PAULO RICARDO ZANONI

UMA METODOLOGIA DE AVALIAÇÃO DE DESEMPENHO DE SISTEMAS PEER-TO-PEER BASEADOS EM TABELAS HASH DISTRIBUÍDAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Luis Carlos Erpen de Bona
Co-orientador: Eduardo Cunha de Almeida

CURITIBA

2011

Z33 Zanoni, Paulo Ricardo
Uma metodologia de avaliação de desempenho de sistemas peer-to-peer baseados em tabelas hash distribuídas / Paulo Ricardo Zanoni. – Curitiba, 2011.
79f. : il., tabs.

Impresso.
Dissertação (mestrado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientador: Luis Carlos Erpen de Bona
Co-orientador: Eduardo Cunha de Almeida

1. Informática. 2. Sistemas operacionais distribuidos (Computadores). I. Bona, Luis Carlos Erpen de. II. Almeida, Eduardo Cunha de. III. Título.

CDD:
004.36

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Paulo Ricardo Zanoni, avaliamos o trabalho intitulado,"UMA METODOLOGIA DE AVALIAÇÃO DE DESEMPENHO DE SISTEMAS PEER-TOPEER BASEADOS EM TABELAS HASH DISTRIBUÍDAS', cuja defesa foi realizada no dia 30 de agosto de 2011, às 14:30 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 30 de agosto de 2011.

Prof. Dr. Luis Carlos Erpen De Bona
DINF/UFPR – Orientador

Prof. Dr. Eduardo Cunha de Almeida
DINF/UFPR – Coorientador

Prof. Dr. Vidal Martins
PUC/PR– Membro Externo

Prof. Dr. Marcos Sfair Sunyé
DINF/UFPR– Membro Interno

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

Agradeço primeiramente a Gilberto e Ivone, pois foram a grande influência que me fez chegar até aqui. Agradeço à Josiani pela enorme paciência e palavras de incentivo. Não posso esquecer dos orientadores Luis e Eduardo, por também terem sido pacientes nos momentos difíceis.

Agradeço também aos professores da UFPR – em especial ao grupo do C3SL – por terem me mostrado tanto o curioso mundo da pesquisa científica quanto as ideologias do mundo do software livre. Neste ponto foram também importantes os inúmeros companheiros de classe, estágio e trabalho que, mesmo sem perceber, me ensinaram muita coisa. Não posso esquecer também dos meus amigos, da Anna e do resto da minha família, que muitas vezes foram trocados por artigos, experimentos e editores de texto.

Por último, mas não menos importantes, não posso esquecer de Bia, Meg e Bob. Eles merecem um parágrafo só pra eles.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

As tabelas hash distribuídas (DHTs, *distributed hash tables*) são redes par-a-par (P2P, *peer-to-peer*) estruturadas que permitem a inserção de dados indexados por chaves. Elas são compostas por um conjunto de participantes (nodos ou pares) dinâmicos sem o controle de uma autoridade central. As DHTs tornaram-se populares ao longo da última década e hoje possuem diversas aplicações, algumas contendo milhões de nodos espalhados ao redor do planeta. Existe uma grande quantidade de DHTs, as quais podem possuir várias implementações e serem configuradas através de diversos parâmetros. Entretanto, não há um consenso sobre a melhor maneira de avaliar o desempenho de uma DHT, o que dificulta a comparação entre o grande número de DHTs existentes e, consequentemente, a escolha da DHT ideal para cada sistema. Esta dissertação apresenta uma revisão de trabalhos que propõem metodologias de avaliação de desempenho de DHTs e de trabalhos que simplesmente avaliam desempenho de DHTs, sem propor metodologias. Com base nesta revisão uma nova metodologia de avaliação de desempenho de DHTs é proposta. Esta metodologia define um conjunto de testes de desempenho composto por métricas e cargas de trabalho. Estas métricas e cargas de trabalho são baseadas nos pontos em comum encontrados nos diversos trabalhos estudados. A metodologia apresentada define também um modelo para a execução de avaliações de desempenho de DHTs composto por três entidades: mestre, controlador e nodo. Este trabalho apresenta também a ferramenta Dhtperf, que além de implementar a metodologia proposta permite facilmente a definição de novas métricas e cargas de trabalho. Esta ferramenta foi utilizada para realizar avaliações de desempenho de diversas DHTs existentes em dois ambientes distintos, validando a metodologia proposta. Os resultados obtidos nas avaliações realizadas são apresentados e discutidos.

ABSTRACT

Distributed hash tables (DHTs) are structured peer-to-peer (P2P) networks that allow the insertion of data indexed by keys. They are composed by a set of dynamic participants (nodes or peers) without the control of a central authority. The DHTs became popular during the last decade and today they have many applications, some containing millions of nodes spread across the planet. There is a big amount of DHTs, which can have many implementations and be configured through many parameters. However, there is no consensus on the best way to evaluate the performance of a DHT, making the comparison between the big number of existent DHTs difficult and, consequently, complicating the choice of the ideal DHT for each system. This dissertation presents a review of works that propose DHT performance evaluation methodologies and works that just evaluate the performance of DHTs, without proposing methodologies. Based on this review a new methodology for DHT performance evaluation is proposed. This methodology defines a set of performance tests composed by metrics and workloads. These metrics and workloads are based on the common points identified in the many works studied. The presented methodology also defines a model for the execution of DHT performance evaluations, composed by three entities: master, controller and node. This work also presents the Dhtperf tool, that besides implementing the proposed methodology, also easily allows the definition of new metrics and new workloads. This tool was used to evaluate the performance of existent DHTs in two distinct environments, validating the proposed methodology. The results obtained in the evaluations are presented and discussed.

CAPÍTULO 1

INTRODUÇÃO

Ao longo da última década as redes par-a-par (P2P, *peer-to-peer*) tornaram-se muito populares [1], o que fez surgir uma grande variedade de aplicações, algumas possuindo até milhões de participantes [53], como no caso da rede Gnutella [18]. Uma rede P2P é composta por um conjunto de participantes (nodos ou *peers*) dinâmicos sem o controle de uma autoridade central. Em geral as redes P2P são projetadas para gerenciarem uma grande quantidade de nodos entrando e saindo do sistema a todo momento.

As redes P2P podem ser classificadas de acordo com suas estruturas em três grupos: as redes P2P *estruturadas*, as *não-estruturadas* e as *fracamente estruturadas* [1]. Nas redes estruturadas os participantes mantém uma topologia pré-definida e seguem regras que definem como as informações são transmitidas e armazenadas. Nas redes P2P não-estruturadas não há regras para a manutenção da topologia e para o local de armazenamento das informações, diminuindo o custo de manutenção da rede, mas impossibilitando o estabelecimento de garantias quanto ao desempenho de certos aspectos da mesma. As redes P2P fracamente estruturadas são caracterizadas por não definirem estritamente a localização do conteúdo armazenado, mas afetarem a sua localização através de seus algoritmos de roteamento.

Os exemplos mais comuns de redes P2P estruturadas são as tabelas hash distribuídas (DHTs, *distributed hash tables*) [3]. As DHTs são estruturas de dados distribuídas que permitem a inserção de dados indexados por chaves. Cada chave é um identificador único que deve ser guardado e utilizado para encontrar os dados previamente inseridos. Dentre as DHTs mais utilizadas estão Chord [59], Pastry [52], Tapestry [66], CAN [48] e Kademlia [35].

Existe uma grande quantidade de DHTs, cada uma com suas próprias características, parâmetros e topologia. Além disso, cada DHT pode possuir diversas implementações, que podem possuir parâmetros que afetam o seu funcionamento. Essa grande diversidade dificulta o processo de comparação entre as DHTs, aumentando o grau de complexidade de tarefas como, por exemplo, a escolha da melhor DHT para um determinada aplicação. Para estas tarefas, metodologias de avaliação de desempenho podem ser utilizadas. Nos últimos anos foram publicados uma série de trabalhos que propõem metodologias de avaliação de desempenho de DHTs [24, 26, 29, 40, 50]. Contudo, ainda não existe um consenso de um conjunto mínimo de testes de desempenho suficientes para realizar esta avaliação, o que dificulta a avaliação e comparação dos resultados obtidos.

Avaliar o desempenho de DHTs não é uma tarefa trivial. Avaliações de desempenho

consistem na escolha de técnicas de avaliação (medição, simulação ou modelagem analítica), métricas, cargas de trabalho, ferramentas para aplicar as cargas de trabalho e obter valores para as métricas escolhidas, além de análise correta dos resultados obtidos [22]. Além disso, fatores intrínsecos às DHTs aumentam a complexidade do processo de avaliação de desempenho, descritos a seguir. Como DHTs são distribuídas, todas as tarefas devem ser implementadas e executadas de maneira distribuída. Como as DHTs são projetadas para permitirem milhões de participantes de maneira escalável, os componentes da avaliação de desempenho também devem ser escaláveis, não interferindo no desempenho da própria DHT durante a avaliação. Por fim, como existem diversos tipos de aplicações que podem utilizar DHTs, é difícil definir conjuntos de cargas de trabalho e métricas que representem todos os casos de uso possíveis.

Esta dissertação apresenta uma revisão de trabalhos que propuseram metodologias de avaliação de desempenho de DHTs [24, 26, 29, 40, 50], bem como de alguns trabalhos que, sem propor metodologias, realizaram avaliações de desempenho de DHTs [5, 6, 20, 52, 59]. Para cada trabalho revisado uma descrição das avaliações de desempenho realizadas é apresentada, incluindo uma descrição do ambiente em que os testes forma executados, as cargas de trabalho utilizadas, as métricas obtidas, as DHTs analisadas e o número de nodos utilizados. Em seguida é apresentada também uma análise comparativa sobre os trabalhos estudados, na qual foram identificadas as semelhanças e diferenças entre cada um dos aspectos analisados.

Com base na análise realizada, uma metodologia para avaliação de desempenho de DHTs é apresentada. Essa metodologia define uma avaliação de desempenho como um conjunto de testes de desempenho, sendo que cada teste é definido através de dois elementos: a *carga de trabalho* e as *métricas* de avaliação. Com base nos pontos comuns identificados entre os trabalhos estudados, são definidas quatro métricas – *latência*, *taxa de sucesso*, *tráfego de rede* e *estado dos nodos* – e cinco cargas de trabalho – *compartilhamento de arquivos*, *sem churn*, *alto churn*, *taxa de sucesso* e *propagação de chaves* – que juntas formam um conjunto de testes de desempenho. A metodologia define também um modelo para a execução destes testes, composto por três entidades, denominadas *mestre*, *controlador* e *nodo*.

Uma ferramenta que implementa a metodologia proposta também é apresentada. Esta ferramenta, denominada Dhtperf, além de implementar a metodologia proposta, permite facilmente a adição de novas métricas e cargas de trabalho, podendo ser utilizada em evoluções ou modificações da metodologia proposta. Ela é um *software livre* [16], portanto estas adições podem ser realizadas por possíveis interessados. Dhtperf pode ser utilizada para avaliar implementações já existentes de DHTs sem a necessidade de alterações nas mesmas. São apresentadas descrições de cada um dos componentes implementados, além de informações sobre como adicionar novas métricas, cargas de trabalho e suporte a novas DHTs.

Dhtperf foi utilizada para realizar avaliações de desempenho de diversas DHTs existentes em dois ambientes, um deles contendo 200 nodos em uma única máquina e o outro contendo cerca de 300 nodos espalhados ao redor do mundo, integrantes da rede PlanetLab [8]. Os resultados obtidos nestas duas avaliações são apresentados e discutidos, validando tanto a metodologia proposta quanto a implementação realizada.

O restante deste trabalho está organizado da seguinte maneira. O Capítulo 2 introduz os conceitos de sistemas distribuídos utilizados neste trabalho, abordando redes P2P e DHTs, além de descrições das DHTs Chord, Pastry e Kad. O Capítulo 3 apresenta e compara diversos trabalhos que realizam avaliações de desempenho de DHTs ou propõem metodologias para tal. Com base na análise realizada, o Capítulo 4 propõe uma metodologia de avaliação de sistemas P2P baseados em DHTs, além de Dhtperf, uma ferramenta que implementa esta metodologia. Os resultados obtidos através da utilização da ferramenta Dhtperf são apresentados e discutidos no Capítulo 5. Por fim, o Capítulo 6 apresenta as considerações finais desta dissertação.

CAPÍTULO 2

SISTEMAS PEER-TO-PEER

Em uma visão abrangente, pode-se dizer que sistemas distribuídos são sistemas compostos por vários computadores – ou processadores – interligados de alguma maneira e interagindo entre si para realizar uma tarefa em comum [3, 11, 56, 60]. Um bom exemplo de sistema distribuído é a Internet, que consiste em milhões de computadores interligados através do protocolo IP (*Internet Protocol*), que juntos provêm uma grande quantidade de serviços que vão desde páginas *web* convencionais até sistemas de busca por inteligência extraterrestre, como o SETI@home [54].

Dentre os diversos tipos de sistemas distribuídos pode-se destacar os sistemas *peer-to-peer* (P2P, par-a-par). Existem muitas definições diferentes – e também até conflitantes – para sistemas P2P, mas uma definição bastante abrangente é apresentada em [1][1]:

> Sistemas par-a-par são sistemas distribuídos que consistem em nodos interconectados capazes de se auto-organizarem em topologias de rede com o propósito de dividirem recursos como conteúdo, ciclos de CPU, armazenamento e largura de banda, capazes de adaptarem-se a falhas e acomodarem populações de nodos transientes enquanto mantém conectividade e desempenho aceitáveis, sem requerer suporte ou intermediação de um servidor ou autoridade global centralizada. [1, p. 337]

Ou seja, sistemas P2P são sistemas distribuídos onde os participantes – também chamados de *peers* ou nodos – formam uma rede com o objetivo de prover algum serviço, sem uma autoridade global controlando o que cada participante pode ou deve fazer. Assim, os participantes devem cooperar para que a rede seja sempre capaz de realizar o seu objetivo, mesmo na presença de uma grande quantidade de nodos entrando e saindo – fenômeno denominado *churn* –, na presença de nodos que não se comportam da maneira esperada ou que não cooperam para o bom funcionamento da rede. Apesar de não haver uma autoridade global controlando os participantes, as redes P2P não são necessariamente isentas de hierarquia: em algumas delas, certos nodos podem ser eleitos para realizar uma tarefa especial – os chamados supernodos, ou superpeers – ou então parte do serviço pode ser centralizado, como por exemplo serviços de autenticação ou indexação de arquivos.

Dentre os exemplos mais conhecidos de sistemas P2P existentes estão diversos sistemas de compartilhamento de arquivos (e.g., BitTorrent [4], eMule [14] e Kazaa [25]), sistemas de backup distribuído (e.g., pStore [2]), sistemas de preservação digital (e.g., Lockss [33])

[1] Traduzida para o português.

e muitos outros. Além disso, sistemas P2P podem ser empregados para implementar funcionalidades de sistemas que não são exclusivamente P2P, como transferências de arquivos em serviços de mensagens instantâneas ou até o roteamento de mensagens de voz, como no caso do Skype [57].

Sistemas P2P podem ser classificados de diversas maneiras, sendo a classificação quanto à estrutura uma das mais comuns, dividindo os sistemas P2P em sistemas *estruturados*, sistemas *não-estruturados* e sistemas *fracamente estruturados* [1].

Em sistemas P2P não-estruturados não há regras definindo a localização do conteúdo, portanto quando um participante deseja encontrar um certo conteúdo na rede ele deve efetuar uma busca, que geralmente resulta na propagação de mensagens entre os nodos, como por exemplo inundação (*flooding*), passeios aleatórios (*random walks*) ou outras técnicas [28]. Dessa maneira, não há como garantir que se um recurso existe na rede ele será definitivamente encontrado. Entretanto, como a rede não precisa possuir uma topologia definida, seu custo de manutenção quando há muitos participantes entrando e saindo é baixo se comparado com o das redes estruturadas.

Já nos sistemas P2P estruturados os participantes formam uma topologia pré-definida e há regras que definem como o conteúdo deve ser armazenado. Em geral cada conteúdo recebe um identificador único que deve ser utilizado para encontrá-lo. Isso faz com que as buscas não gerem tantas mensagens quanto nas redes P2P não-estruturadas e garante que se determinado conteúdo existir ele será encontrado. Por outro lado, apesar de facilitar a busca por conteúdo cujo identificador é conhecido, este processo pode dificultar – ou até impossibilitar – as buscas quando o identificador do conteúdo é desconhecido, como por exemplo em buscas por palavras-chave. Outra desvantagem das redes estruturadas é a manutenção da topologia, pois quando há uma grande quantidade de participantes entrando e saindo da rede a troca de muitas mensagens entre eles pode ser exigida [1].

Nos sistemas P2P fracamente estruturados a localização do conteúdo não é especificada mas é afetada pelos algoritmos de roteamento da rede.

Como exemplos de redes P2P não-estruturadas podemos citar as redes Gnutella [18] e Overnet [44]. Um exemplo de rede P2P fracamente estruturada é a rede Freenet [9]. Já os maiores exemplos de redes P2P estruturadas são as tabelas hash distribuídas (DHTs, do inglês *Distributed Hash Tables*), que são o objeto de estudo deste trabalho. Como exemplos de DHTs podemos citar CAN [48], Chord [59] e Pastry [52].

2.1 Tabelas hash distribuídas

Uma Tabela Hash Distribuída (*Distributed Hash Table* ou DHT) é uma estrutura de dados semelhante à uma tabela hash convencional, com a exceção de que ela não está centralizada na memória de uma máquina, mas sim distribuída em uma rede de participantes – ou nodos – que comportam-se seguindo um determinado protocolo, tornando

a DHT uma abstração de uma tabela hash convencional. A seguir são introduzidas as tabelas hash convencionais, as tabelas hash distribuídas, bem como três das DHTs mais utilizadas: Chord [59], Pastry [52] e Kademlia [35].

Uma tabela *hash* convencional – não-distribuída, também chamada de tabela de dispersão, tabela de espelhamento ou dicionário – é uma estrutura de dados que serve para indexar e armazenar dados, semelhante ao vetor [10]. Cada dado a ser armazenado – também chamado de valor, ou *value* – possui uma chave – *key* –, que é utilizada para indexá-lo na tabela. Sobre a chave é aplicada uma função denominada *função hash*, que serve para obter o índice – ou hash – correspondente ao elemento na tabela. O conjunto composto por todos os índices possíveis é finito e denominado *espaço de chaves*. Assim, uma tabela hash possui duas operações: a inserção, que insere um dado associado a uma chave na tabela – comumente representada pela função `put(key,value)` – e a recuperação, que recupera um dado a partir de sua chave – comumente representada pela expressão `value = get(key)`.

A função hash deve ser elaborada de maneira a evitar ao máximo que dados diferentes possam ser associados à mesma chave. Porém, quase sempre isso é impossível e quando dois ou mais conjuntos de dados diferentes são associados à mesma chave ocorre uma colisão. Há diversas técnicas para o tratamento de colisões, porém quando o espaço de chaves é muito grande a probabilidade de ocorrência de colisões é muito pequena, o que pode levar muitas implementações de tabelas hash a simplesmente ignorarem a possibilidade de colisões.

A Figura 2.1 mostra um exemplo de tabela hash utilizada para implementar uma agenda telefônica. As chaves são representadas pelos nomes e os dados são os números de telefone. A função hash `strlen()` obtém o local de destino dos dados através da quantidade de caracteres de suas respectivas chaves. Note que uma função como essa não é uma boa função hash pois pode, nesse caso, produzir muitas colisões.

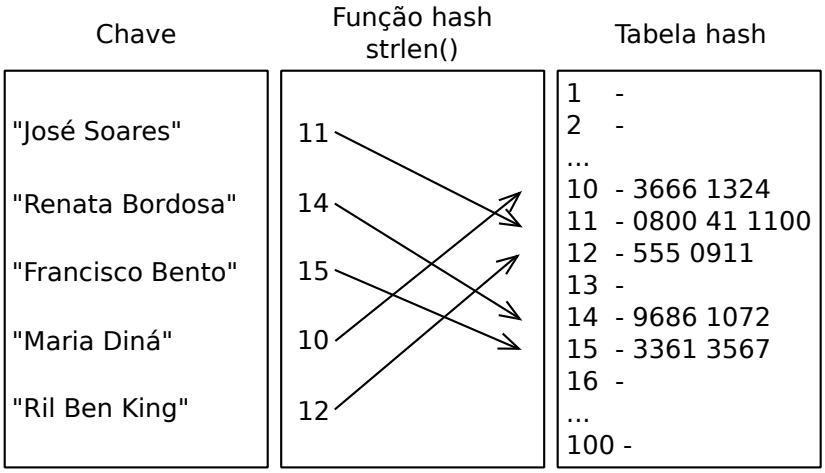

Figura 2.1: Exemplo de tabela hash

Muitas linguagens de programação, como Perl, Python e Ruby costumam ter as tabelas hash como tipos de dados já implementados em suas bibliotecas padrões. Dentre os exemplos de funções hash comumente utilizadas podemos citar as funções SHA-1 [38] e MD5 [51].

Uma tabela hash distribuída é como uma tabela hash convencional, porém ela é formada por diversos nodos, cada um responsável por apenas uma fração do espaço total de chaves. Assim, quando um dado é inserido na tabela o nodo responsável por armazená-lo é escolhido com base no hash da chave desse dado. Em geral DHTs são elaboradas para funcionarem de maneira escalável em grandes redes P2P, podendo ter de dezenas a milhões de participantes, dentre os quais muitos estão constantemente entrando e saindo da rede. Essas asserções criam uma série de problemas a serem resolvidos, explicados adiante.

Os casos mais comuns de uso de DHTs são as redes que envolvem o armazenamento de arquivos, como redes de compartilhamento de arquivos, sistemas de backup e sistemas de preservação digital. Outras aplicações podem envolver desde estruturas para *multicast* [7] até a implementação de sistemas de DNS (*Domain Name System*) não hierárquicos [46].

Como a DHT deve ser capaz de possuir até milhões de nodos, dentre os quais muitos podem estar constantemente entrando e saindo, não há como fazer cada nodo ter conhecimento sobre todos os outros participantes da rede, pois isso ocasiona a troca de uma grande quantidade de mensagens, principalmente quando há a necessidade de notificação de entrada e saída de nodos na rede. Uma maneira de resolver este problema seria através da centralização das informações sobre os participantes da rede em algum servidor, mas isso também pode ser considerado não escalável para uma grande quantidade de nodos. Assim, cada nodo deve conter informações sobre apenas uma parte da rede, fazendo com que a união de todos os nodos forme uma topologia. As informações que cada nodo mantém sobre os outros participantes da DHT são denominadas *tabelas de roteamento*.

A manutenção de uma topologia consistente pode gerar a necessidade de muitas trocas de mensagens de controle entre os nodos, o que pode diminuir a escalabilidade da rede. Ainda, quando houver a necessidade de comunicação de nodos que estão distantes na topologia, o roteamento das mensagens deve ser eficiente, evitando assim que a rede fique inundada com mensagens caso haja muitos nodos se comunicando.

Nas DHTs atuais o número de nodos pelos quais uma mensagem entre quaisquer dois nodos passa é, em geral, $O(\log N)$ para uma DHT com N participantes. Para a DHT CAN, por exemplo, este número é $O(d.N^{\frac{1}{d}})$, onde d é o número de dimensões da rede, um parâmetro da DHT. Para a DHT Chord este número é $O(\log N)$. Já para a DHT Pastry é $O(\log_{2^b} N)$, onde 2^b é a base para o número de bits utilizado nos identificadores dos nodos, um parâmetro da DHT. O tamanho das tabelas de roteamento também é expresso como função do número de participantes da DHT, sendo, por exemplo, $2d$ para a DHT CAN, $\log N$ para a DHT Chord, e $B \log_B N + B \log_B N$ para a DHT Pastry, onde

$B = 2^b$. O trabalho apresentado em [31] apresenta uma comparação entre as ordens de complexidades dos algoritmos e tabelas de roteamento utilizados por diversas DHTs.

Ainda, independentemente da maneira com que os nodos estão organizados, deve haver uma forma determinística de decidir qual a fração do espaço total de chaves sobre a qual cada nodo é responsável. Assim, quando há uma busca por uma determinada chave o nodo correto é encontrado.

Além de dificultar a manutenção da topologia, o fato de haver muitos nodos entrando e saindo da rede ainda pode ocasionar outros problemas. Chaves anteriormente inseridas podem desaparecer da rede, caso os nodos que as possuam saiam da rede. Ou então estas chaves podem acabar sendo armazenadas no nodo errado, como no caso onde um nodo A é responsável por uma chave e um nodo B entra na rede, passando a ser o novo responsável por algumas das chaves do espaço de chaves de A. Esses problemas costumam ser abordados não só com bons métodos para a manutenção da topologia, mas também com técnicas como replicação de dados, múltiplos `get`s, `put`s repetidos implícitos e outros [55].

Como consequência dos problemas apresentados, a implementação de uma DHT consiste não apenas na implementação das funções `put` e `get` presentes nas tabelas hash convencionais, mas também em funções de entrada e saída da DHT e rotinas para detecção de eventos e manutenção da topologia. Não há uma definição formal amplamente discutida e aceita para a interface de uma DHT, porém em geral as DHTs podem ser facilmente mapeadas para alguma interface comum, composta por funções como `put`, `get`, `join` e `leave` que servem para, respectivamente, inserir e recuperar dados da DHT, entrar na DHT e sair da DHT.

O trabalho apresentado em [12] realiza uma discussão inicial no estabelecimento de uma interface comum entre DHTs e redes P2P estruturadas em geral. Ele define um modelo constituído de três camadas, chamadas de *tiers*. O tier 0 consiste na interface de uma camada de roteamento baseado em chaves (*Key-based Routing Layer*, KBR), o tier 1 é composto de interfaces genéricas, como DHT (para tabelas hash distribuídas), CAST (para multicast) e DOLR (para *Decentralized Object Location and Routing*), e por fim o tier 2 é formado pelas aplicações, que podem utilizar tanto o tier 2 quanto o tier 1 diretamente para cumprirem suas tarefas. Este modelo é implementado pelo simulador de redes P2P Overlay Weaver [42].

Dentre as DHTs mais utilizadas podem ser citadas CAN [48], Chord [59], Pastry [52], Tapestry [66], Bamboo [49] e Kademlia [35]. Segundo [27] Chord é a DHT mais citada em trabalhos científicos e Kademlia é a DHT mais utilizada em aplicações que estão em produção.

Afim de melhor exemplificar as informações contidas neste capítulo, a seguir são descritas as DHTs Chord e Pastry. Por terem sido propostas há vários anos, uma série de trabalhos propõem mudanças e melhorias nessas DHTs, porém as descrições apresentadas

a seguir são baseadas em suas propostas originais.

2.1.1 Chord

A DHT Chord foi proposta em 2001 e é uma das DHTs mais citadas e utilizadas até hoje, possuindo variantes como Koorde [23] e Accordion [30]. Nesta DHT os nodos formam uma topologia em anel, onde cada nodo possui informações sobre seu nodo sucessor, seu antecessor e também sobre um conjunto de nodos chamado *finger table*.

O identificador de cada nodo é obtido calculando o hash de seu endereço IP. Quando um nodo entra na rede ele obtém seu nodo sucessor e antecessor, passando a ser o responsável por todas as chaves que estão entre o seu identificador e o identificador de seu nodo antecessor. Assim, diz-se que o nodo responsável por uma chave K é o nodo sucessor da chave K.

Enquanto as informações sobre sucessores e antecessores estiverem consistentes, a corretude do algoritmo será mantida, porém as mensagens roteadas passarão de sucessor em sucessor, fazendo com que o número de nodos pelos quais uma mensagem passe até chegar ao seu destino seja $O(N)$ para uma rede de N nodos. Para fazer que o roteamento seja feito em $O(\log_2 N)$ mensagens é utilizada a finger table.

A finger table é simplesmente uma tabela de roteamento que possui informações sobre no máximo m nodos, onde m é o número de bits do identificador de cada nodo e chave. A i-ésima entrada da finger table de um nodo cujo identificador é n possui informações sobre o nodo sucessor da chave $n + 2^{i-1}$, considerando uma aritmética módulo 2^m.

Para tentar garantir que as tabelas de roteamento – listas de sucessores, antecessores e finger table – mantenham-se consistentes mesmo com nodos entrando e saindo da rede é utilizado um protocolo de estabilização. Este protocolo define que cada nodo deve, periodicamente, executar um algoritmo que visa ajustar os ponteiros de sucessor e antecessor, corrigir as finger tables e finalizar a inicialização de nodos que entraram recentemente. De acordo com os autores, enquanto o tempo necessário para ajustar as tabelas de roteamento for menor que o tempo que a rede leva para duplicar seu tamanho, o número de nodos sobre o qual uma mensagem passa continuará a ser $O(\log_2 N)$. Apesar de tudo, é impossível garantir que a rede esteja sempre consistente, portanto é esperado que em raros casos as mensagens não sejam entregues ou então passem por muito mais nodos do que a quantidade usual.

A Figura 2.2 ilustra uma rede Chord composta por um identificador de 4 bits. Os nodos cujos identificadores são 1, 3, 6, 10 e 12 estão presentes e a finger table do nodo 3 é representada pelas setas. A tabela 2.1 mostra as finger tables de todos os nodos presentes na rede, uma por linha.

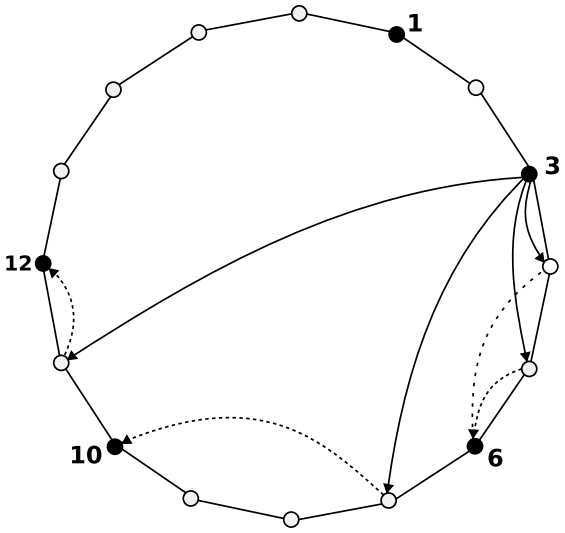

Figura 2.2: Representação de uma DHT Chord e finger table do nodo 3

Tabela 2.1: Finger tables de uma DHT Chord

Nodo	Espaço de chaves	Antecessor	Sucessor	Finger $i = 0$	Finger $i = 1$	Finger $i = 2$	Finger $i = 3$
1	13..1	12	3	3	3	6	10
3	2..3	1	6	6	6	10	12
6	4..6	3	10	10	10	10	1
10	7..10	6	12	12	12	1	3
12	11..12	10	1	1	1	1	6

2.1.2 Pastry

O diferencial da DHT Pastry é que ela leva em conta a localidade espacial dos nodos. Cada nodo possui a informação de sua distância em relação a outros nodos na rede subjacente e a utiliza para tentar fazer com que cada mensagem percorra a menor distância possível. O método para a obtenção dessa distância não é definido, mas uma sugestão dos autores é o número de saltos (*hops*) no roteamento IP.

A DHT Pastry possui um espaço de chaves de 128 bits, no qual cada nodo recebe um identificador (*nodeId*) aleatório dentro desse espaço e é responsável pelas chaves numericamente mais próximas de seu identificador. Cada identificador é definido como uma sequência de dígitos descritos em base 2^b, sendo que b é um dos parâmetros da DHT. A ideia por trás do roteamento é encaminhar as mensagens para nodos que possuam um prefixo (em dígitos) com cada vez mais dígitos iguais aos do identificador da chave.

Cada nodo da DHT possui três tabelas utilizadas no roteamento de mensagens, denominadas *routing table*, *leaf set* e *neighborhood set*. Cada uma das n linhas da *routing table* possui $2^b - 1$ entradas, sendo que cada entrada da i-ésima linha possui i dígitos em comum com o identificador do nodo em questão (sendo n o número de dígitos do identificador

do nodo). A tabela denominada *leaf set* possui um conjunto de L nodos cujos identificadores são numericamente mais próximos do nodo em questão. A tabela *neighborhood set* possui um conjunto de M nodos que são mais próximos do nodo em questão levando-se em consideração a métrica de proximidade espacial. L e M são parâmetros da DHT. A Tabela 2.2 ilustra as tabelas de roteamento de um nodo cujo identificador é 02130211 em uma rede onde $b = 2$, $L = 8$ e $M = 8$.

Tabela 2.2: Tabelas de rodeamento de nodo da rede Pastry

Identificador 02130211, $b = 2$, $L = 8$ e $M = 8$

Routing table			
	*1*2323212	*2*2331231	*3*0132113
0*0*223333	**0***1*012300		**0***3*332211
02*0*12301		**02***2*21021	**02***3*23232
021*0*1123	**021***1*3212	**021***2*3210	
	0213*1*132	**0213***2*333	**0213***3*000
02130*0*01	**02130***1*03		**02130***3*12
021302*0*0		**021302***2*0	**021302***3*1
0213021*0*		**0213021***2*	**0213021***3*
Leaf Set			
02130210	02130220	02130212	02130203
02130200	02130202	02130213	02130233
Neighbordhood set			
03232100	02312301	33001212	01011020
21323221	11112222	01230322	23010123

Quando um nodo precisa rotear uma mensagem com chave D, primeiro ele verifica se D faz parte de seu *leaf set*, roteando a mensagem diretamente para D neste caso. Caso contrário, ele procura em sua *routing table* pelo nodo que possui o prefixo com o maior número de dígitos em comum com D (sendo *prefixo* considerado como os dígitos a esquerda do número), roteando para este nodo nesse caso.

As tabelas de roteamento são atualizadas toda vez que um nodo entra na rede ou toda vez que um nodo falho é detectado. Outra característica importante do Pastry é que ao preencher sua *routing table* um nodo sempre escolherá nodos que tem maior proximidade espacial, caso haja mais de um a escolher para algum campo da tabela.

2.1.3 Kademlia

Na DHT Kademlia cada nodo recebe um identificador aleatório de 160 bits, sendo que a distância entre dois nodos é o resultado da operação *ou exclusivo* (XOR) entre os identificadores destes nodos. As tabelas de roteamento de cada nodo da DHT são formadas por 160 listas denominadas *k-buckets*. Para cada k-bucket i ($0 <= i < 160$) são armazenadas informações sobre k nodos – endereço IP, porta UDP e identificador – cujas

distâncias estão entre 2^i e 2^{i+1} do nodo que as armazena, sendo que k é um parâmetro da DHT. Estas listas são ordenadas de acordo com a última vez em que cada nodo foi visto e são atualizadas toda vez que um nodo recebe alguma mensagem de qualquer outro nodo.

A vantagem da DHT Kademlia é que quando um nodo precisa encontrar outro nodo (ou o valor de alguma chave) ele efetua buscas paralelas em α nodos diferentes, sendo que α é um parâmetro da DHT. Cada nodo deve responder a operação de busca com um conjunto formado pelos k nodos conhecidos mais próximos do nodo a ser encontrado. Esta operação é efetuada diversas vezes, recursivamente, até que cada um dos k nodos mais próximos encontrados tenham recebido e respondido a operação de busca.

Para inserir uma chave na DHT o nodo que a insere deve primeiro efetuar uma busca pela chave, encontrando os k nodos mais próximos da mesma. Este nodo deve então enviar uma mensagem denominada *store* para cada um dos k nodos encontrados, fazendo assim com que cada um possua uma cópia da chave. A DHT define também que cada chave inserida deve ser periodicamente re-inserida para não ser considerada expirada.

A operação de entrada na rede também é implementada através de buscas. Quando um nodo i deseja conectar-se à DHT ele precisa entrar em contato com um nodo j, adicionando-o no k-bucket apropriado. O nodo i então faz uma busca pelo seu próprio identificador, o que faz com que outros nodos tomem conhecimento de i e o adicionem em seus respectivos k-buckets.

CAPÍTULO 3

AVALIAÇÃO DE DESEMPENHO DE SISTEMAS BASEADOS EM DHTS

Desde as propostas das primeiras DHTs muitas outras foram propostas, cada uma com suas características e topologia, além de modificações e extensões nas já existentes. Cada uma destas DHTs pode também possuir uma série de parâmetros que podem ser utilizados para alterar o seu funcionamento, como por exemplo os parâmetros k e α da DHT Kademlia, explicados no Capítulo 2. Ainda, uma mesma DHT pode possuir diversas implementações, além de parâmetros específicos da implementação, como por exemplo o nível de replicação, que é um parâmetro das DHTs implementadas pela ferramenta Overlay Weaver. Essa grande quantidade de opções dificulta o processo de comparação entre DHTs, aumentando o grau de complexidade de tarefas como, por exemplo, a escolha da DHT ideal – incluindo parâmetros e implementação – para um determinado sistema. Para facilitar este processo, avaliações de desempenho podem ser utilizadas.

Estão presentes na literatura uma série de trabalhos que de alguma forma realizam avaliações de desempenho de DHTs. Enquanto alguns desses trabalhos utilizam essas avaliações apenas como forma de demonstrar os resultados de suas pesquisas [5, 6, 20, 52, 59], outros são especificamente sobre avaliações de desempenho de DHTs, podendo chegar até a propor suas próprias metodologias para tal [24, 26, 29, 40, 50]. Este capítulo apresenta uma análise comparativa sobre esses trabalhos.

O primeiro ponto em comum a ser ressaltado é que as avaliações de desempenho apresentam em geral a mesma estrutura [22]. Cada avaliação é composta de um ou mais componentes que chamaremos de *testes de desempenho*. Cada teste de desempenho é independente e tem como objetivo efetuar a análise ou medição de uma ou mais *métricas*. Essa medição deve ser efetuada sobre uma DHT cujos nodos seguem um determinado comportamento. Denominamos a descrição deste comportamento a ser seguido pelos nodos da DHT – incluindo inserções, buscas, entradas e saídas – de *carga de trabalho* ou *workload*. Portanto, podemos dizer que uma avaliação de desempenho é composta de um ou mais testes de desempenho e que um teste de desempenho consiste na aplicação de uma carga de trabalho à uma rede e na obtenção de diversas métricas enquanto a mesma está sendo aplicada. A Figura 3.1 apresenta um diagrama que representa os conceitos apresentados.

A descrição da carga de trabalho deve ser a mais completa possível, visando garantir que uma mesma carga de trabalho, ao ser aplicada diversas vezes, resulte em redes com o mesmo o padrão de comportamento. Dentre as informações incluídas devem estar o

número de nodos, definições de suas entradas e saídas na rede, operações realizadas por cada nodo (como `put` e `get`) e quaisquer outros tipos de eventos que possam influenciar o comportamento da DHT. Detalhes como a organização topológica dos nodos, tanto na DHT (camada de aplicação) quanto nas outras camadas de rede, também podem ser necessários em alguns casos, como por exemplo na avaliação de certas métricas de DHTs que levam em consideração a localidade espacial dos nodos, como Pastry.

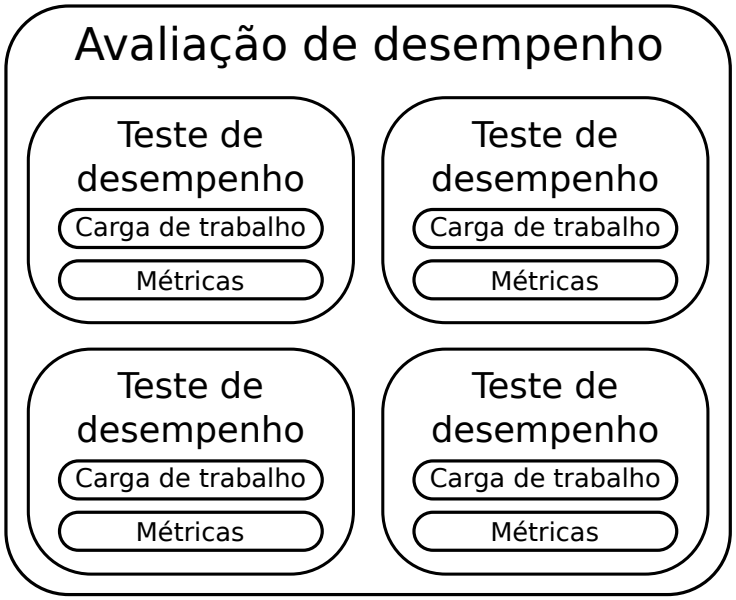

Figura 3.1: Componentes de uma avaliação de desempenho

3.1 Descrição da análise

A seguir será apresentada uma revisão de trabalhos que propõem metodologias de avaliação de desempenho de DHTs e também de trabalhos que apenas realizam avaliações de desempenho de DHTs sem propor metodologias. Para cada um dos trabalhos estudados foram analisadas as avaliações de desempenho propostas e foi elaborado um resumo contendo as seguintes informações: as DHTs – ou redes – utilizadas, o ambiente sobre o qual as avaliações foram realizadas, o tamanho das redes em questão, as cargas de trabalho e as métricas utilizadas. Esses dados são apresentados ao final deste capítulo, na Tabela 3.1.

Cada uma das *redes* utilizadas pelos trabalhos analisados foi associada a uma das seguintes categorias: (i) implementações de DHTs (i.e., prontas para serem utilizadas por aplicações reais); (ii) DHTs providas por simuladores; (iii) infraestruturas que possibilitam a criação de DHTs, como no caso de Plaxton [47], que inspirou as DHTs Pastry [52] e Tapestry [66]; ou (iv) modelos teóricos de DHTs ou de infraestruturas que possibilitam a criação de DHTs, como no caso de [26].

Os *ambientes* utilizados nos diversos trabalhos analisados foram classificados como: (i) *reais*, para os casos mais próximos da utilização de DHTs ou de um ambiente real de execução, como o PlanetLab [8]; (ii) *simulação*, para os casos nos quais simuladores de redes P2P foram utilizados; (iii) *emulação*, para os casos nos quais foram utilizadas implementações reais em ambientes restritos (e.g., centenas de nodos virtuais em um único nodo físico ou uma pequena rede local) ou (iv) *analíticos*, para os casos nos quais os estudos realizados foram puramente teóricos e os resultados apresentados são, por exemplo, provas matemáticas.

O *tamanho da rede* descreve a quantidade de nodos utilizados nas avaliações realizadas pelos trabalhos apresentados. Uma exceção é o trabalho [26], onde a análise teórica é realizada para o caso no qual o tamanho da rede tende ao infinito.

A *carga de trabalho* descreve o comportamento adotado pelos nodos na rede e as operações realizadas pelos mesmos, o que inclui as operações `put` e `get`, além das entradas e saídas de nodos da rede (*churn*). Para os casos onde houve vários testes de desempenho com comportamentos diferentes, os diversos tipos de comportamento são listados. Na maioria dos casos analisados o comportamento dos nodos foi simplesmente realizar um certo número de requisições em um determinado intervalo de tempo. Em alguns casos os comportamentos foram baseados em cargas de trabalho observadas em aplicações reais, como em [6]. Em outros casos os comportamentos foram apenas inspirados nos possíveis comportamentos reais, ou então a escolha do comportamento adotado simplesmente não foi justificada. Houve também avaliações onde foram analisadas apenas as mensagens de controle geradas pelas DHTs (e.g., mensagens de manutenção da topologia), portanto os nodos não realizaram requisições, como em [5].

Cada trabalho realizado possui seu próprio método para analisar desempenho e medir os resultados. Entretanto certas *métricas* como a latência foram analisadas de maneiras diferentes em diversos trabalhos (e.g., em [29] a latência é simplesmente o intervalo de tempo entre uma requisição `get` e sua resposta, mas em [50] a latência é expressa como a relação entre o intervalo de tempo obtido e o tempo de *ping* entre o nodo que busca a chave e o nodo que a possui). Apesar dessas pequenas diferenças, a Tabela 3.1 apresenta definições simplificadas das métricas utilizadas. As descrições completas são apresentadas nas Seções 3.2 e 3.3.

3.2 Trabalhos que propõem metodologias de avaliação de desempenho de DHTs

O trabalho apresentado em [50], realizado em 2003, propõe dois testes a serem utilizados para comparar o desempenho de DHTs. O primeiro teste consiste em uma aplicação que faz uma série de buscas pelos nodos responsáveis por chaves escolhidas aleatoria-

mente. A métrica analisada nesse teste é a razão entre o tempo de duração dessa busca e o tempo de envio e resposta (*round trip*) de uma mensagem direta entre o nodo que realiza a busca e o nodo responsável pela chave. O segundo teste consiste na busca por arquivos que possuem réplicas na rede. Um conjunto de 10 objetos é inserido na DHT, sendo que cada um possui 4 réplicas. O teste consiste em fazer cada nodo realizar uma busca por cada um desses objetos e então medir o tempo de *round trip* entre o nodo que realiza a busca e o nodo responsável pela réplica encontrada, além do tempo de round trip entre o nodo que realiza a busca e o nodo responsável pela réplica mais próxima. A métrica a ser analisada é a razão entre os dois tempos obtidos.

Os experimentos foram realizados com aproximadamente 80 nodos da rede PlanetLab [8]. As DHTs utilizadas foram Chord, em sua versão implementada pelo *Massachusetts Institute of Technology* (MIT) [61] e Tapestry, em sua versão implementada pela Universidade de Berkeley [62]. Não foi mencionada a presença de churn durante a realização dos testes.

Apesar desse trabalho ser especificamente sobre avaliação de desempenho em DHTs, apenas dois testes de desempenho foram propostos e não há discussão sobre o porquê desses dois testes terem sido escolhidos ao invés de outros. Além disso, houve necessidade de modificação nas DHTs utilizadas para a realização das avaliações.

Li et al. [29] propõem uma metodologia de avaliação de desempenho de DHTs em situações de *churn*. Essa metodologia é baseada na análise da troca entre desempenho e custo de comunicação e possui apenas duas métricas. Os autores propõem o uso da latência como métrica unificada para a análise de desempenho, enquanto o "custo" de comunicação pode ser resumido ao cálculo de bytes transferidos. Além disso, apenas uma carga de trabalho é definida e utilizada nas análises.

O objetivo dos testes de desempenho é comparar a influência dos parâmetros de cada DHT em seu desempenho. Neste trabalho são realizados diversos testes de desempenho em cada DHT, sendo que em cada teste os parâmetros utilizados pela DHT são diferentes, de maneira que o conjunto total dos testes reflete uma fração considerável de todas as possibilidades de parâmetros. As métricas obtidas foram o número médio de bytes transmitidos e a latência média de cada busca.

O padrão de comportamento seguido pelos nodos é fixo, sendo que os nodos fazem buscas em média a cada dez minutos e falham (*crash*) em média a cada uma hora. Os testes foram realizados através do simulador p2psim [45] e a rede era composta por 1024 nodos. As DHTs analisadas foram Chord, Tapestry, Kelips [19] e Kademlia.

O trabalho apresentado em [40] propõe um *framework* para a realização de benchmarks de robustês (*robustness benchmarks*) em sistemas distribuídos (não apenas DHTs), cujo objetivo é medir a qualidade de serviço (*Quality of Service*, ou QoS) de um sistema durante e depois da ocorrência de perturbações. Os autores definem o benchmark de robustês como sendo composto de quatro componentes: a carga de trabalho (*workload*),

a carga de perturbação (*perturbation load*), as métricas e o ambiente de testes.

A carga de trabalho é definida por aplicação e contém, por exemplo, o tipo de busca a ser realizada por cada nodo. O ambiente de testes inclui o tipo e número de nodos a serem utilizados, além de sua topologia e capacidades de transmissão. As métricas são divididas em métricas genéricas e métricas que dependem do serviço (*service-specific*). Por fim, as perturbações também podem ser genéricas ou específicas para cada serviço, e como exemplos podem ser citadas a adição de *churn* na rede, diversos tipos de falhas, picos de atividade, erros de configuração, rede ou até de programação. Portanto, pode-se dizer que a definição de *carga de trabalho* que apresentamos no início deste capítulo é um conjunto que envolve a carga de trabalho, carga de perturbação e, de certa forma, o ambiente de testes como definidos em [40]. A implementação do benchmark é realizada através do framework ACME [41].

Apesar das descrições e exemplos, não há a formalização de um conjunto de cargas de trabalho, cargas de perturbação ou métricas ideais para uma avaliação de robustês. Os autores mencionam um teste realizado com 150 nodos porém não oferecem detalhes sobre o experimento. Dos exemplos citados no artigo pode-se apenas inferir que uma possível carga de trabalho seria a realização de buscas aleatórias e dentre as métricas estariam a latência das requisições, a porcentagem de requisições completadas, a quantidade da largura total de banda utilizada, a consistência das respostas quando múltiplos nodos realizam a mesma requisição ao mesmo tempo e a corretude dos resultados. As DHTs utilizadas nos experimentos foram Chord, Pastry e Tapestry.

O trabalho apresentado em 2006 por Kong et al. [26], diferentemente dos outros aqui apresentados, propõe um método puramente analítico para comparar o desempenho de DHTs. Este método é denominado *Reachable Component Method* (RCM, método dos componentes alcançáveis) e serve para expressar a porcentagem de caminhos falhos em uma DHT como função da probabilidade de falha dos nodos em uma rede de tamanho que tende ao infinito. A métrica obtida na análise realizada foi a *roteabilidade* (*routability*) do sistema. A carga de trabalho consiste em uma DHT completa sem buscas e com nodos com probabilidade de falha uniforme q. O trabalho mostra o RCM aplicado às DHTs CAN, Chord, Kademlia e Symphony [34], além de sistemas baseados no roteamento Plaxton [47].

Realizado em 2007, o trabalho apresentado em [24] propõe uma metodologia para a avaliação de desempenho de DHTs em redes emuladas. Essa metodologia é composta por cenários de avaliação (chamados de *scenarios* no artigo) e métricas de desempenho. A definição dos cenários de avaliação é equivalente à definição de carga de trabalho apresentada no início deste capítulo. A metodologia permite facilmente a criação de novos cenários, porém o conjunto de métricas é fixo e não há discussão sobre a escolha e relevância do conjunto de cenários e métricas apresentados.

As DHTs Accordion, Bamboo, Chord e Pastry são analisadas sob 5 diferentes cargas

de trabalho, sendo elas: (i) rede estática, sem nodos entrando e saindo, (ii) rede com nodos entrando e saindo, (iii) rede onde uma chave pode possuir diferentes valores, (iv) rede dividida em dois grupos, onde um só faz operações do tipo `put` e o outro só faz operações do tipo `get` e (v) rede dividida em dois grupos, onde um não faz nada e o outro faz tanto operações `put` quanto `get`. As métricas analisadas em cada teste são a taxa de sucesso das operações `get`, o atraso de resposta das mesmas e também o tráfego de rede gerado.

Para a realização dos testes uma rede como a Internet foi emulada em uma pequena rede local, onde cada máquina possuía uma grande quantidade de nodos que executavam implementações não modificadas das DHTs. O número de nodos utilizados nos testes variou de 91 a 991.

3.3 Trabalhos que realizam avaliações de desempenho de DHTs

Existem muitos trabalhos que realizam avaliações de desempenho de DHTs sem propor metodologias para tal. Como seria inviável descrever todos estes trabalhos, esta seção apresenta apenas alguns, escolhidos por apresentarem exemplos das práticas mais comuns entre os trabalhos deste tipo.

O trabalho que apresentou a definição inicial da DHT Chord, em 2001, realizou alguns testes de desempenho visando demonstrar as características da DHT então proposta [59]. Dos cinco testes realizados, quatro foram simulações compostas de 500 até 16000 nodos e um dos testes utilizou um ambiente emulado, composto de até 200 nodos.

No primeiro teste chaves foram inseridas na DHT e a métrica utilizada foi o número de chaves por nodo. O segundo teste teve como métrica o número de nodos contactados nas operações `get` e foi realizado em uma rede sem *churn*, com nodos apenas realizando buscas. A carga de trabalho do terceiro teste é composta das seguintes etapas: (i) a DHT é criada, (ii) chaves são inseridas, (iii) espera-se a estabilização da rede, (iv) uma porcentagem nodos é removida da rede e (v) os nodos realizam operações `get` nas chaves previamente inseridas. A métrica analisada nesse teste foi a porcentagem de buscas que falharam. Utilizando a mesma métrica do terceiro teste, o quarto teste tem como carga de trabalho nodos que tanto entram e saem da DHT como realizam buscas segundo um processo de Poisson. Por fim, o quinto teste foi realizado em um ambiente emulado em 10 máquinas reais e teve como objetivo analisar a escalabilidade da rede. A latência das operações `get` foi avaliada para quantidades crescentes de nodos na DHT – de 10 a 200.

Também em 2001, o trabalho [52] apresentou a DHT Pastry. Além da definição da DHT foram realizados seis testes de desempenho em redes emuladas de 1000 a 100000 nodos.

Nos três primeiros testes os nodos da rede apenas fazem buscas aleatórias e não há *churn*. As métricas são o número de nodos contatados em cada busca (*hops*), a distância

percorrida por cada mensagem (distância real) o número de mensagens por segundo que cada nodo pode enviar. O quarto e o quinto teste foram realizados em três versões do Pastry com diferentes algoritmos de atualização de tabelas de roteamento. A carga de trabalho do quarto teste envolveu a entrada de 5000 nodos na rede, um por um e a métrica analisada foi o número de entradas nas tabelas de roteamento. O quinto teste foi caracterizado por nodos buscando chaves e teve como métrica o número de mensagens que chegaram em nodos próximos dos nodos em questão. Por fim, no sexto teste uma rede com 5000 nodos foi formada e então 500 desses nodos foram removidos. Esse procedimento foi realizado duas vezes: uma para uma versão do Pastry sem os algoritmos de reparo de tabelas de roteamento e outra para uma versão convencional do Pastry. As métricas analisadas foram o número de entradas nas tabelas de roteamento, o número de nodos pelos quais cada mensagem passou e o número de mensagens necessárias para reparar as tabelas de roteamento.

O trabalho proposto em 2004 por Bjurefors et al. [5] realizou três tipos de testes de desempenho com o objetivo de estudar o desempenho da rede Pastry em sistemas heterogêneos. O primeiro teste realizado mede a intensidade das mensagens de controle na inicialização dos nodos. O segundo teste é semelhante ao primeiro, porém os nodos, após entrarem, efetuam buscas por chaves. O terceiro teste é idêntico ao segundo mas utiliza tabelas de roteamento particionadas. As métricas analisados foram o tamanho médio dos caminhos percorridos pelas mensagens e o tempo de entrega das mensagens.

Cada um dos três testes foi realizado diversas vezes e com o número de nodos participantes variando de 30 até 3000 nodos por teste. Os testes foram realizados através de um simulador, o Microsoft Research Pastry Simulator v3.0A, e os nodos simulados foram divididos em dois grupos: os nodos fortes, que representam computadores com conexões de 100 Mbit/s e baixa latência e os nodos fracos, que são equivalentes a celulares conectados a redes GPRS (*General Packet Radio Service*).

Em Castro et al. [6] são apresentadas técnicas que podem ser utilizadas para desfazer algumas das supostas "desvantagens" das redes P2P estruturadas sobre as redes P2P não-estruturadas e testes de desempenho que mostram os resultados obtidos. Dos quatro testes de desempenho utilizados, três utilizam cargas de trabalho retiradas de aplicações reais e um utiliza uma carga de trabalho própria. Nos dois primeiros testes os nodos da rede não realizam buscas ou inserções, e a métrica analisada é o tráfego de rede. No terceiro teste são também realizadas buscas e as métricas são a taxa de sucesso, a latência e o número de mensagens transmitidas por nodo. Por fim, o quarto teste não utiliza cargas de trabalho reais, e sim cargas de trabalho com comportamentos baseados na distribuição de Poisson e com diferentes níveis de *churn*. As métricas analisadas são as mesmas do terceiro teste. As redes analisadas foram Pastry, HeteroPastry e SuperPastry, os testes foram simulados e possuiam de 10000 a 37000 nodos.

Harvesf e Blough [20] apresentaram técnicas de inserção de réplicas em DHTs e de-

monstraram seus resultados através testes de desempenho em uma rede Pasty simulada com 1024 nodos. Os dois primeiros testes tiverem como carga de trabalho buscas em DHTs com número crescente de nodos falhos e como métricas a taxa de sucesso das buscas. O primeiro teste comparou técnicas de inserção de réplicas e o segundo teste comparou técnicas de roteamento em ambientes com réplicas. O terceiro teste comparou técnicas de busca em conjuntos de réplicas tendo como métrica o tempo de resposta e carga de trabalho buscas periódicas em um sistema com 25% dos nodos falhos.

3.4 Discussão

Trabalho	**Rede(s)**	**Ambiente**	**Tamanho da rede**	**Cargas de trabalho**	**Métricas**
Stoica et al. 2001 [59]	Chord	Simulação e emulação	10 – 16000 nodos	(i) inserções sem churn, (ii) buscas sem churn, (iii) buscas com churn	(i) chaves por nodo, (ii) *hops*, (iii) taxa de sucesso (iv) latência
Rowstron e Druschel 2001 [52]	Pastry	Emulação	1000 – 100000 nodos	(i) buscas sem churn, (ii) buscas com churn	(i) *hops*, (ii) distância real, (iii) vazão, (iv) entradas das tabelas de roteamento, (v) taxa de sucesso (vi) tráfego de rede.
Rhea et al. 2003 [50]	Chord e Tapestry	Real: PlanetLab	79 – 83 nodos	(i) buscas sem churn	(i) latência, (ii) proximidade de réplicas
Bjurefors et al. 2004 [5]	Pastry	Simulação: Microsoft Research Pastry Simulator v3.0A	30-3000 nodos	(i) inicialização, (ii) *i* com buscas, (iii) *ii* com tabelas particionadas	(i) tráfego de rede
Li et al. 2004 [29]	Chord, Tapestry, Kelips e Kademlia	Simulação: p2psim	1024 nodos	(i) buscas com churn	(i) tráfego de rede, (ii) latência
Oppenheimer et al. 2004 [40]	Chord, Pastry e Tapestry	Emulação: ACME	150 nodos	(i) buscas com churn	(i) latência, (ii) taxa de sucesso, (iii) tráfego de rede, (iv) consistência
Castro et al. 2005 [6]	Pastry, Hetero-Pastry e Super-Pastry	Simulação	10000 – 37000 nodos	(i) workload real sem queries, (ii) workload real, (iii) churn	(i) tráfego de rede, (ii) taxa de sucesso, (iii) latência
Kong et al. 2006 [26]	CAN, Chord, Kademlia, Symphony e Plaxton	Analítico	Infinito	(i) rede com nodos falhos	(i) caminhos falhos
Kato e Kamiya 2007 [24]	Accordion, Bamboo, Chord e Pastry	Emulação: rede local	91 – 991 nodos	(i) buscas sem churn, (ii) buscas com churn, (iii) outros modelos mais complexos	(i) taxa de sucesso, (ii) latência, (iii) tráfego de rede
Harvesf e Blough 2007 [20]	Pastry	Simulação	1024 nodos	(i) buscas com nodos falhos e replicação	(i) taxa de sucesso, (ii) latência

Tabela 3.1: Trabalhos que medem o desempenho de DHTs

Observamos que algumas DHTs como Chord e Pastry foram utilizadas em quase todos os trabalhos analisados, portanto podem ser consideradas como DHTs “mais populares”. Além disso, quase todos os trabalhos analisados utilizaram ambientes simulados ou emulados, sendo apenas um deles baseado em um ambiente real e também somente um deles realizado de forma puramente analítica. Diversos trabalhos que utilizaram simuladores não mencionaram qual o simulador utilizado. Outro fato observado foi que apesar das

discussões sobre escalabilidade e aplicações que podem possuir até milhões de nodos, boa parte dos trabalhos não analisou mais do que apenas algumas centenas ou milhares de nodos. Ainda, apesar das cargas de trabalho das avaliações realizadas envolver quase somente a realização de buscas periódicas, as características das redes envolvidas variaram bastante, principalmente com relação aos algoritmos e técnicas utilizados internamente pelas DHTs. Todas as cargas de trabalho observadas em aplicações reais foram retiradas de apliações de compartilhamento de arquivos. Observamos também que as métricas mais utilizadas nas avaliações de desempenho foram latência, tráfego de rede e taxa de sucesso. Por fim, cada um dos trabalhos que propõem metodologias de avaliação de desempenho cria sua própria metodologia, sem reutilizar as metodologias propostas em outros trabalhos.

CAPÍTULO 4

METODOLOGIA PROPOSTA

Apesar da grande quantidade de trabalhos que efetuam avaliações de desempenho de DHTs, não existe um consenso sobre um conjunto padrão de testes de desempenho que seja suficiente para realizar uma avaliação de desempenho de uma DHT, assim como também não existe uma metodologia padrão para a realização destes testes. Assim, cada novo trabalho define e utiliza seus próprios métodos, o que dificulta a avaliação e comparação dos resultados apresentados nos diferentes trabalhos.

Este capítulo propõe uma metodologia de avaliação de DHTs baseada nos pontos comuns identificados entre os diversos trabalhos existentes na literatura. Esta metodologia consiste em um conjunto padrão de cargas de trabalho e métricas, além de definições sobre como realizar os testes de desempenho. Até onde pudemos identificar, nosso trabalho é o primeiro a propor uma metodologia de avaliação de desempenho de DHTs baseada nas metodologias já propostas e avaliações já realizadas por outros trabalhos.

É apresentada também uma implementação desta metodologia, denominada Dhtperf. Esta implementação, além de fornecer as cargas de trabalho e métricas propostas, permite facilmente a implementação de novas cargas de trabalho e novas métricas, portanto pode ser utilizada em possíveis evoluções ou variações da metodologia proposta por este trabalho.

O restante deste capítulo está organizado da seguinte maneira. A primeira parte da metodologia, referente à *definição* dos testes de desempenho, composta pelas métricas e cargas de trabalho, é apresentada na Seção 4.1. A segunda parte da metodologia, referente à *execução* dos testes de desempenho, definida através das entidades *mestre*, *controlador* e *nodo* é apresentada na Seção 4.2. Por fim, a Seção 4.3 apresenta Dhtperf, uma implementação para a metodologia proposta

4.1 Definição dos testes de desempenho

A metodologia proposta define cada teste de desempenho como uma combinação de dois elementos: a *carga de trabalho* e as *métricas* a serem analisadas. A carga de trabalho é a definição das ações a serem executadas pelos nodos da DHT, como buscas, inserções, entradas e saídas da DHT. Como o objetivo é que cada teste possa ser utilizado por diversas DHTs e em diversos ambientes, algumas informações como por exemplo a disposição dos nodos na DHT ou os detalhes das camadas de rede mais inferiores devem ser omitidas da carga de trabalho. As métricas definem os dados que devem ser coletados pelos nodos, como latência das operações, taxa de sucesso e outras. O conjunto de dados a serem

coletados deve ser independente da DHT a ser utilizada e a obtenção desses dados não pode exigir modificações na DHT.

Em uma avaliação de desempenho de DHT há uma série de fatores que devem ser analisados, portanto cada avaliação deve ser composta por diversos testes de desempenho. Uma das maiores dificuldades na avaliação é que cada uma das inúmeras possíveis aplicações que utilizam DHTs pode apresentar cargas de trabalho diferentes. Portanto o conjunto de testes de desempenho relevantes para uma determinada aplicação pode ser completamente diferente do conjunto de testes de outra. Com isso, a determinação de um conjunto de testes a ser adotado como padrão para a maioria dos casos não é trivial.

Identificamos dois tipos de cargas de trabalho: as cargas de trabalho baseadas em aplicações reais – que podem ser simplesmente cópias das cargas originais ou então geradas a partir da análise do comportamento observado nestas aplicações, expressas através de distribuições estatísticas ou de outras maneiras – e as cargas de trabalho que visam exercitar alguma operação ou funcionalidade específica da DHT. Um exemplo muito comum do primeiro grupo são as cargas de trabalho de aplicações de compartilhamento de arquivos, como as que podem ser encontradas no *The Peer-to-Peer Trace Archive* [63]. Já sobre segundo grupo pode-se citar as cargas de trabalho com alto grau de *churn*, que visam testar a estabilidade das DHTs, e as cargas de trabalho onde os nodos efetuam grandes quantidades de buscas, visando obter medidas como latência, corretude e outras.

As métricas também podem ser divididas em dois grupos: as métricas gerais, que podem ser medidas em todos os tipos de testes de desempenho e as métricas específicas, que só podem ser medidas em testes que possuem cargas de trabalho específicas. Do primeiro grupo pode-se citar as três métricas identificadas como mais analisadas: latência das mensagens, tráfego de rede gerado e taxa de sucesso das operações. Do segundo grupo pode-se citar, por exemplo, as métricas que avaliam a distância e disponibilidade das réplicas armazenadas em DHTs com suporte a réplicas.

O conjunto básico de métricas e cargas de trabalho dos testes de desempenho definidos pela metodologia proposta contempla as métricas e cargas de trabalho mais utilizadas encontradas na literatura e é apresentado a seguir. Como é impossível abranger todos os casos de uso e possíveis funcionalidades das DHTs, a implementação dessa metodologia, apresentada na Seção 4.3, permitirá facilmente a adição de novas métricas e cargas de trabalho, possibilitando avaliações mais completas e evoluções da metodologia proposta neste trabalho.

4.1.1 Métricas

Para o grupo das métricas que podem ser obtidas em todos os testes de desempenho inicialmente escolhemos apenas as mais utilizadas nos trabalhos analisados: latência, taxa de sucesso, e tráfego de rede gerado. Das outras métricas observadas na literatura, quase

todas foram utilizadas em apenas um dos trabalhos, sendo que dentre as razões pelas quais elas não foram escolhidas estão: algumas podem requerer modificações nas DHTs para serem obtidas (como por exemplo o número de saltos nas buscas), outras podem ser consideradas equivalentes a uma das três métricas selecionadas (como a porcentagem de caminhos falhos, semelhante à taxa de sucesso e o número de saltos, equivalente à latência) e outras são dependentes de cargas de trabalho específicas (como a vazão).

Chamaremos de latência o tempo decorrido entre o início de execução de uma operação e a obtenção de sua resposta final, o que inclui todas as comunicações necessárias entre os diversos possíveis nodos durante a operação. A latência pode ser obtida não somente para operações de `put` e `get`, como também para `join` e `leave`.

A obtenção da taxa de sucesso consiste em decidir se a resposta obtida pela operação da DHT retornou um resultado correto ou incorreto. A metodologia proposta define esta métrica apenas para as operações `join` e `get`. Para a operação `join`, caso o nodo consiga conectar-se à DHT a operação pode ser classificada como sucesso, caso contrário, como falha. Já para a operação `get` a taxa de sucesso pode ser definida e obtida de diversas maneiras.

Idealmente, o resultado correto de uma operação `get` em uma chave existente é o valor da própria chave caso ela exista, ou uma notificação de chave inexistente no caso contrário. Entretanto, caso as operações `put` e `get` estejam sendo executadas concorrentemente por diversos nodos diferentes, possivelmente espalhados em diversas máquinas, obter o resultado correto para a operação `get` não é uma tarefa simples: a chave buscada não foi encontrada pois ela não existe ou pois o nodo simplesmente não conseguiu localizá-la? O nodo que executou a operação `get` não tem como responder a esta pergunta se ele não tiver conhecimento do estado completo da DHT, o que inclui informações como as chaves previamente inseridas por todos os nodos, os nodos que eram responsáveis por chaves e desconectaram-se da DHT, o grau de replicação e muitas outras. Testes de desempenho de DHTs podem tentar resolver ou contornar esse problema através de diversas técnicas.

Uma das possibilidades é fazer com que cada nodo saiba, antes de efetuar uma busca, se a chave em questão existe na rede. Porém, em alguns casos manter os nodos com informações consistentes sobre o estado das chaves pode exigir muita comunicação e sincronia, sendo esta uma boa opção apenas para utilizar em simuladores. Outra alternativa é registrar o momento em que cada operação é realizada por cada nodo e o resultado obtido, para que alguma entidade possa fazer a análise da consistência após a realização dos testes, com base nos dados fornecidos por todos os nodos. Porém, uma série de problemas relacionados à sincronização destas operações também pode dificultar a análise dos resultados. Por fim, para certas cargas de trabalho – principalmente onde há pouco ou nenhum *churn* – pode ser possível prever eficientemente quais chaves fazem parte da DHT, facilitando a obtenção da métrica mas tornando-a dependente da carga de trabalho.

Para esta metodologia a taxa de sucesso para a operação `get` foi definida de uma

maneira simplificada, facilitando as implementações mas tornando a métrica dependente de cargas de trabalho específicas. A cada `get` realizado pela DHT um sucesso é registrado se a operação retornou algum valor qualquer. Assim, caso uma chave não exista na DHT e uma operação `get` não consiga obter um resultado – que é o comportamento esperado – esta métrica registrará uma falha. Portanto, a métrica só tem significado relevante se analisada em conjunto com uma carga de trabalho feita especificamente para ela, como por exemplo uma em que todas as operações `get` sejam realizadas em chaves que já foram previamente inseridas na DHT.

O tráfego de rede gerado por uma DHT pode ser medido através tanto do número de bytes transmitidos por cada nodo quanto do número de mensagens trocadas entre os nodos. A obtenção da métrica na forma de número de mensagens trocadas nem sempre pode ser realizada facilmente, podendo em alguns casos ser impossível obtê-la sem modificações no código-fonte das DHTs a serem analisadas. Além disso, o tamanho de cada mensagem pode variar entre as diversas DHTs e suas implementações, tornando a comparação potencialmente injusta. Já a obtenção da métrica em razão do número de bytes pode ser mais facilmente implementada, possivelmente através apenas dos recursos do sistema operacional, sem necessidade de modificação nas DHTs analisadas.

Durante as avaliações de desempenho que realizamos (descritas no Capítulo 5) detectamos que nodos das DHTs avaliadas comumente apresentam problemas que os inutilizam por todo o resto do teste de desempenho. Em alguns casos os processos de alguns nodos entram em *loop infinito*, possivelmente devido a bugs no software da DHT analisada. Em outros casos os processos simplesmente terminam, mesmo durante operações como *get*. Ou ainda, as falhas detectadas não são na entidade nodo, mas sim nas máquinas onde os testes de desempenho estão sendo executados ou até mesmo na rede – caso mais comum em ambientes como o PlanetLab. Para uma correta interpretação dos resultados das métricas obtidas é imprescindível que haja informações sobre os nodos que não conseguiram completar as suas cargas de trabalho, portanto criamos a métrica *estado dos nodos*.

A métrica *estado dos nodos* organiza os nodos em três grupos: *terminados*, *falhos* e *não terminados*. Nodos que executam a carga de trabalho corretamente e reportam os resultados de suas métricas ao final do teste são caracterizados como terminados. Somente nodos terminados têm suas outras métricas processadas e mostradas no resultado final. Nodos que, por algum problema, têm seus processos interrompidos de maneira inesperada, consequentemente encerrando a aplicação da carga de trabalho e não reportando os resultados de suas métricas são associados ao grupo de nodos falhos. Nodos que não encerram inesperadamente mas que também não reportam os resultados de suas métricas após o final do teste de desempenho são classificados como não terminados.

4.1.2 Cargas de trabalho

A metodologia proposta define uma carga de trabalho como uma descrição de como cada nodo da DHT efetuará as quatro operações básicas: `join`, `leave`, `put` e `get`. Cada carga de trabalho possui a definição de um ou mais *perfis* de nodos. Cada perfil possui quatro *sub-perfis*, um para cada operação. Estes sub-perfis possuem as definições de como o nodo que segue aquele perfil efetuará a operação em questão durante a execução da carga de trabalho. Esta definição pode conter tanto descrições exatas do comportamento (e.g., insira a chave *c1* com valor *v1* 50 segundos após o início) quanto descrições mais genéricas (e.g., insira chaves selecionadas aleatoriamente dentro de um dicionário pré-definido com uma frequência que segue a distribuição de Poisson, com lambda 5, em segundos). A Figura 4.1 apresenta uma representação esquemática de uma carga de trabalho conforme proposta pela metodologia.

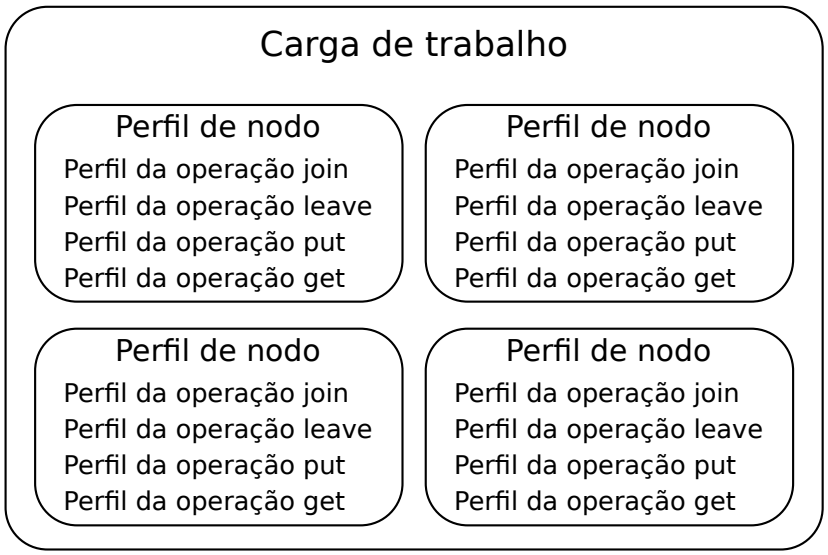

Figura 4.1: Representação esquemática de uma carga de trabalho

A análise da literatura permitiu a identificação de uma grande quantidade de cargas de trabalho. Foram detectados diversos tipos de cargas de trabalho, algumas baseadas em aplicações reais, outras com diferentes graus de churn, outras feitas para a obtenção de métricas específicas, além de cargas cujo objetivo é exercitar funcionalidades específicas das DHTs. Com base na análise realizada elaboramos cinco cargas de trabalho para esta metodologia: *compartilhamento de arquivos*, baseada em aplicações existentes que utilizam DHTs; *sem churn* e *alto churn*, que têm como objetivo comparar diferentes graus de churn; *taxa de sucesso*, cujo objetivo principal é obter resultados para a métrica de mesmo nome e *propagação de chaves*, um exemplo de carga de trabalho que serve para exercitar uma funcionalidade específica da DHT.

A carga de trabalho *compartilhamento de arquivos* é baseada no comportamento observado em aplicações reais de DHTs. Aplicações de compartilhamento de arquivos foram

escolhidas por serem o caso de uso mais comum e com maior número de nodos dentro das aplicações existentes que utilizam DHTs. A parte correspondente a entrada e saída dos nodos é baseada no estudo publicado por Moritz et al. [58], que observou a DHT Kad da rede eDonkey por seis meses. Este estudo sugere que o número de nodos observados na DHT é constante. Este comportamento foi simulado em nossa carga de trabalho através do perfil da operação `join`, que define que sempre que um nodo sair da DHT ele deverá retornar imediatamente. Outra informação relevante é que o tempo que um nodo permanece conectado (denominado pelo estudo de *session time*) segue a distribuição estatística de Weibull (há diversos valores sugeridos para os argumentos da distribuição, mas os valores utilizados foram 0.56 para *forma* e 97.63 para *escala*), portanto o perfil da operação `leave` define que após um nodo entrar na DHT ele deverá deixá-la em uma quantidade de tempo aleatória baseada nesta distribuição.

Para a operação `put` foi observado que algumas implementações existentes publicam suas chaves a cada 4 horas, portanto a carga de trabalho *compartilhamento de arquivos* deve seguir este comportamento através do perfil da operação `put`. O estudo publicado em [36] observou que 95% das palavras-chave publicadas nunca são pesquisadas, comportamento que também deve ser seguido. Não foi encontrado na literatura um estudo que caracterize as operações `get` realizadas em DHTs, portanto definimos que estas operações são realizadas com uma frequência que segue a distribuição estatística de Poisson (com lambda 100). Outra alternativa à proposta apresentada seria utilizar uma carga de trabalho baseada nas que estão presentes no *The Peer-to-Peer Trace Archive*, porém antes de serem utilizadas estas cargas teriam que ser de alguma maneira adaptadas e convertidas para o formato de carga de trabalho utilizado pela nossa metodologia.

O impacto de churn nas DHTs pode ser medido através da comparação das métricas obtidas com as cargas de trabalho *sem churn* e *alto churn*. Ambas as cargas realizam suas operações de `put` e `get` seguindo a distribuição estatística de Poisson (lambda 50). Na carga *sem churn* o perfil de `join` define que todos os nodos conectam-se à DHT no início da aplicação da carga de trabalho e o perfil de `leave` não define nenhuma operação, portanto os nodos não se desconectam até o final da aplicação da carga. Já na carga *alto churn* o perfil da operação `leave` define que os nodos saem da DHT seguindo a distribuição estatística de Weibull (com parâmetros de forma e escala iguais ao da carga de trabalho *compartilhamento de arquivos*) e o perfil de `join` define que os nodos retornam imediatamente após saírem. Níveis intermediários de churn podem ser testados através de modificações na carga *alto churn*.

A carga de trabalho *taxa de sucesso* tem como objetivo medir a taxa de sucesso das operações `get`: o resultado esperado é que todas as operações retornem o valor definido para cada uma das chaves. O perfil da operação `join` define que no início desta carga de trabalho todos os nodos conectam-se à DHT. Em seguida um único nodo insere um conjunto de chaves pré-definidas, portanto o perfil de `put` deste nodo realiza as operações

descritas e o perfil de `put` dos outros nodos define que nenhuma operação será realizada. Depois disso todos os nodos realizam operações `get`, uma para cada chave que foi previamente inserida. O objetivo desta carga é que 100% das operações `get` sejam classificadas como sucesso. O perfil de `leave` define que todos os nodos saem da DHT ao final do experimento.

Semelhante à *taxa de sucesso*, a carga de trabalho *propagação de chaves* verifica se as chaves inseridas na DHT são repassadas para outros nodos quando estes entram na DHT e passam a ser responsáveis por um determinado espaço de chaves. Primeiramente, um único nodo cria a DHT, sem a presença de outros nodos. Este nodo então insere chaves nesta DHT. Somente após este processo que os outros nodos devem conectar-se à DHT. Posteriormente todos os nodos buscam as chaves previamente inseridas. O objetivo desta carga de trabalho também é que todas as operações `get` sejam classificadas como sucesso. Os perfis de `join`, `leave`, `put` e `get` desta carga de trabalho são praticamente iguais aos da carga *taxa de sucesso*, as únicas diferenças são as definições de quando as operações ocorrem.

4.1.3 Testes de desempenho

Como já definido, um teste de desempenho é composto de uma carga de trabalho e métricas a serem analisadas. A metodologia proposta define quatro métricas, denominadas *latência*, *taxa de sucesso*, *tráfego de rede* e *estado dos nodos*, além de cinco cargas de trabalho, denominadas *compartilhamento de arquivos*, *sem churn*, *alto churn*, *taxa de sucesso* e *propagação de chaves*. Todas as métricas podem ser obtidas em todas as cargas de trabalho, porém a métrica *taxa de sucesso* para a operação `get` só pode ser corretamente interpretada nas cargas de trabalho *taxa de sucesso* e *propagação de chaves*.

Esta metodologia propõe cinco testes de desempenho, um para cada carga de trabalho definida. Estes testes são referenciados através dos nomes de suas respectivas cargas de trabalho: *compartilhamento de arquivos*, *sem churn*, *alto churn*, *taxa de sucesso* e *propagação de chaves*. Todas as métricas são obtidas em todas as cargas de trabalho, independentemente de serem ou não relevantes para a carga de trabalho em questão: a interpretação do resultado deve ser feita por quem realiza as avaliações.

4.2 Execução dos testes de desempenho

A execução de um teste de desempenho consiste em aplicar uma carga de trabalho a uma DHT e obter as métricas necessárias. Este processo deve ser realizado de maneira que sua própria execução exerça o mínimo possível de influência sobre as métricas a serem obtidas. Para a metodologia proposta, adotamos três princípios básicos que devem ser seguidos na execução dos testes: *escalabilidade*, *adaptabilidade* e *não-intrusividade*.

Para seguir o princípio da escalabilidade, a execução do teste de desempenho deve ser realizada de maneira no mínimo tão escalável quando as DHTs a serem analisadas: a adição de nodos não deve dificultar a realização dos testes de desempenho. O principio da adaptabilidade exige que a maneira com a qual os testes de desempenho são realizados seja igual para todas as DHTs a serem analisadas, facilitando a comparação dos resultados. Uma consequência deste princípio é que nenhum dos testes pode exigir informações específicas das DHTs a serem analisadas, como por exemplo listas de nodos vizinhos. Por fim, o princípio da não-intrusividade estabelece que a realização dos testes de desempenho não deve exigir modificações nas DHTs a serem avaliadas, pois essas próprias modificações, quando realizadas, podem afetar os resultados das avaliações.

Com base nos princípios estabelecidos acima, um modelo para a execução dos testes de desempenho foi elaborado. Este modelo é formado por três entidades: o *mestre*, os *controladores* e os *nodos*. O mestre é a entidade que gerencia a execução do teste, organizando os controladores e recebendo seus resultados. O controlador é responsável por aplicar a carga de trabalho em um nodo e obter as métricas necessárias. Por fim, a entidade nodo é um nodo da DHT que está sendo analisada. A relação entre estas três entidades está ilustrada na Figura 4.2. As linhas contínuas representam a comunicação entre as entidades realizada através da ferramenta que executa os testes. As linhas pontilhadas representam a comunicação entre as entidades *nodo*, realizada através das DHTs.

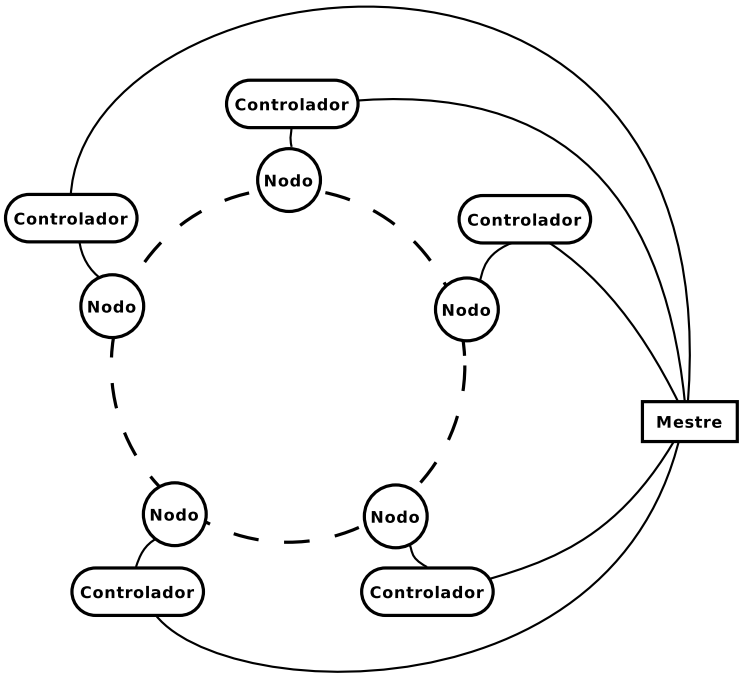

Figura 4.2: Representação esquemática da metodologia proposta

O mestre deve gerenciar toda a execução de um teste de desempenho. Ele possui a definição da carga de trabalho a ser aplicada na rede e deve definir as cargas de trabalho

a serem aplicadas por cada controlador. Ele também é responsável por gerenciar o início da execução do teste e por receber as métricas reportadas pelos controladores ao final da execução. O mestre também pode obter suas próprias métricas, como no caso da métrica *estado dos nodos*. O mestre comunica-se apenas com os controladores, nunca com os nodos.

A troca de informações entre o mestre e os controladores deve funcionar de maneira externa à DHT a ser avaliada e ser rápida e confiável o suficiente para não ferir o princípio da escalabilidade, afetando negativamente o resultado dos testes de desempenho realizados. Apesar da metodologia definir a entidade mestre como uma só, sua implementação pode ser feita de maneira distribuída, tanto através de redes P2P como de estruturas hierárquicas ou quaisquer outras. Outra maneira de evitar os problemas de escalabilidade é fazer com que toda a comunicação entre o mestre e os controladores seja realizada antes e depois (não durante) a aplicação da carga de trabalho. Esta última é a alternativa escolhida pela implementação da nossa metodologia, apresentada adiante.

Para cada nodo existente na DHT deve existir uma entidade chamada controlador. O controlador é uma aplicação que utiliza uma DHT. Ele recebe do mestre as informações sobre a carga de trabalho que deverá aplicar ao nodo e deve aplicar esta carga ao nodo, obtendo os resultados das métricas que devem ser posteriormente reportadas ao mestre. O controlador e o nodo devem estar preferencialmente na mesma máquina, sendo que a comunicação entre os dois se dá através dos comandos `join`, `leave`, `put` e `get`. As métricas a serem obtidas pelo controlador devem ser definidas em torno destas quatro operações, garantindo que o princípio da não-intrusividade seja seguido.

A entidade nodo é uma abstração de um nodo da DHT que está sendo avaliada. Ele recebe do controlador comandos que representam operações a serem realizadas na DHT e deve traduzir estes comandos para operações suportadas pela DHT que está sendo avaliada. A interface entre o nodo e o controlador é sempre a mesma (são as operações `join`, `leave`, `put` e `get`), porém cada implementação diferente de DHT a ser avaliada exige uma implementação diferente da entidade nodo. A interface simples entre o nodo e o controlador, apesar de potencialmente limitante, garante que o princípio da adaptabilidade seja respeitado.

4.3 Dhtperf

Dhtperf (ou simplesmente *dhtperf*, nome derivado de *DHT PERFormance*) é uma ferramenta que implementa a metodologia proposta neste capítulo. Ela foi desenvolvida para avaliar o desempenho de quaisquer DHTs que estejam prontas para serem utilizadas em aplicações reais, portanto suporta avaliações em ambientes reais e emulados, mas não simulados ou analíticos. Além dos princípios básicos de escalabilidade, adaptabilidade e não-intrusividade, Dhtperf tem também como princípio básico a extensibilidade: adaptar

a ferramenta para permitir novas métricas, novos tipos de cargas de trabalho ou novas DHTs deve ser fácil e realizado de maneira modular. Assim, Dhtperf deve ser facilmente adaptável tanto para diversos tipos de cargas de trabalho e métricas quanto também para evoluções ou variações da metodologia proposta por este trabalho.

Dhtperf é uma ferramenta escrita na linguagem C++, possui seu código-fonte aberto, licenciado através da versão 3 ou superior da *GNU General Public License* (GPLv3+) [17], e está disponível em [13]. Atualmente, somente os sistemas Unix são suportados e testados, porém suporte para outros sistemas operacionais é uma funcionalidade planejada para as próximas versões. Dhtperf pode ser considerada uma evolução da ferramenta Multidhtshell, publicada em [65].

4.3.1 Mestre, controlador e nodo

A implementação da entidade mestre se dá através de um único processo no sistema operacional, denominado `dhtperf-master`. Este processo é responsável por ler as definições das cargas de trabalho e comunicar-se com os controladores. Toda a comunicação com os controladores é feita através de conexões TCP (uma por controlador) e todas as trocas de mensagens são feitas somente antes e depois dos controladores aplicarem as suas cargas de trabalho sobre os nodos, i.e., enquanto os controladores estão aplicando suas cargas de trabalho e obtendo métricas, não há comunicação com o mestre. Em sua implementação atual, as conexões TCP entre o mestre e os controladores ficam abertas durante todo o teste de desempenho (mesmo não havendo transmissão de dados durante a aplicação das cargas de trabalho), porém é possível fazer com que até mesmo esta conexão seja fechada para a aplicação das cargas de trabalho.

A entidade controlador é implementada através do processo `dhtperf-controller`, sendo que há uma instância deste processo para cada nodo da DHT. O controlador deve necessariamente estar na mesma máquina da entidade nodo, porém pode haver mais de uma instância de controlador – e, consequentemente, nodo – por máquina. Os principais componentes da implementação do controlador são as classes do tipo `Profile` e as classes do tipo `DhtNode`. As classes do tipo `Profile` estão relacionadas à aplicação da carga de trabalho, possuindo as implementações dos tipos de comportamento a serem seguidos por cada nodo da DHT. Cada classe do tipo `DhtNode` possui uma implementação capaz de controlar um tipo diferente de DHT. Este controle pode ser feito tanto através de bibliotecas C e C++ quanto também de processos do sistema operacional.

A representação da entidade nodo é definida pela implementação das classes do tipo `DhtNode`. Cada implementação diferente de DHT suportada pela ferramenta exige a implementação de uma classe derivada da classe `DhtNode`. Essas classes são responsáveis por instanciar os nodos das DHTs analisadas, o que atualmente ocorre através da criação de processos separados no sistema operacional. A comunicação entre as classes derivadas

de `DhtNode` atualmente implementadas e os processos das DHTs se dá pelas entradas e saídas padrão desses processos, através de *pipes* do Unix. Apesar das implementações atuais serem dessa maneira, não há restrição nenhuma para a maneira com a qual as classes derivadas de *DhtNode* iniciam e comunicam-se com suas DHTs.

Os nodos suportados pela implementação atual são os seguintes: *dummy*, *kaddemo*, *openchord*, *owshellchord*, *owshellkad*, *owshellpastry* e *pastconsole*. *Dummy* é apenas utilizado para testes, não opera nenhuma DHT. *Kaddemo* utiliza o programa `kaddemo_static`, um console para a DHT Kademlia, fornecido pela implementação *MaidSafe DHT* [32], escrita na linguagem C++. *Openchord* utiliza a classe `chord.console`, um console para a DHT Chord, fornecido pela implementação *Open Chord* [39], realizada em Java. *Owhsellchord*, *owshellkad* e *owshellpastry* utilizam o `owdhtshell`, um console para as diversas implementações de DHTs fornecidas pelo *toolkit Overlay Weaver* [43], escrito em Java. Por fim, *pastconsole* utiliza o programa `past-console`, uma pequena implementação de um console para a DHT Pastry, que utiliza as bibliotecas Java do projeto *Free Pastry* [15].

Adicionar suporte ao um nodo tipo de DHT é simples: basta implementar uma nova classe derivada de `DhtNode`. Se a DHT prover algum tipo de biblioteca C ou C++, a própria classe implementada poderá chamar as funções da biblioteca diretamente. Caso a DHT seja implementada em outra linguagem, será necessário escrever um programa que recebe comandos da classe `DhtNode` e os executa na DHT. Neste caso, há uma outra classe denominada `Wrapper` com funcionalidades que podem facilitar a implementação: basta fazer com que a classe implementada seja derivada tanto de `DhtNode` quanto de `Wrapper`.

O diagrama de sequência UML apresentado na Figura 4.3 representa a comunicação entre as entidades mestre e controlador. A mensagem rotulada como *Abre conexão TCP* é simplesmente uma abertura de conexão TCP convencional, e não é uma mensagem de protocolo. Quando esta conexão é aberta o mestre define como o nodo associado ao controlador se comportará durante a execução do teste de desempenho. O mestre então envia a mensagem *Tipo de DHT* especificando qual DHT terá seu desempenho avaliado, seguida da mensagem *Perfil*, que especifica o comportamento do nodo durante a execução do teste. Depois que todos os controladores esperados conectarem-se com o mestre e receberem suas mensagens de *Perfil*, o mestre envia, simultaneamente, para cada controlador, uma mensagem de *Início*, indicando o início do teste de desempenho. Os controladores então executam a avaliação e, após terminarem, enviam mensagens *Resultados* indicando os resultados obtidos pelas métricas. Um controlador pode enviar diversas mensagens *Resultados*, uma para cada métrica obtida. Depois que terminar de enviar seus resultados, o controlador envia a mensagem *Fim*, indicando que seu papel na execução do teste terminou. Por fim, o mestre faz todo o processamento necessário para transformar as métricas obtidas por cada nodo nos resultados exibidos ao usuário da ferramenta.

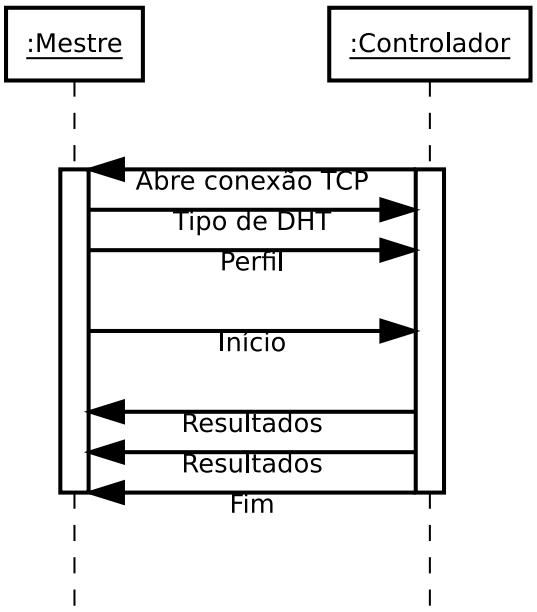

Figura 4.3: Diagrama de sequência das mensagens entre o mestre e um controlador

4.3.2 Cargas de trabalho

Cada carga de trabalho é definida em um arquivo texto e passada ao mestre como argumento da linha de comando. Este arquivo possui definições do número de nodos esperado, a duração do teste de desempenho, os diversos perfis de nodos e, por fim, a definição de qual perfil será atribuído a cada nodo. O número de nodos é apenas o número esperado, podendo ser alterado através de parâmetros passados pela linha de comando ao mestre. A duração do teste de desempenho é atualmente fornecida em segundos.

A definição de um perfil de comportamento de nodo é composta por quatro sub-perfis, um para cada operação (`join`, `leave`, `put` e `get`). Estes sub-perfis são implementados por classes do tipo `Profile`. Cada um desses sub-perfis é representado por uma sequência de caracteres composta de uma ou mais palavras. A primeira palavra identifica uma classe do tipo `Profile` e as demais palavras são passadas como argumentos para o construtor desta classe. Os únicos requisitos para a classe do tipo `Profile` são que ela possua uma fila de eventos (do tipo `join`, `leave`, `put` ou `get`, de acordo com o sub-perfil) e que implemente uma função que é chamada periodicamente e é responsável por inserir eventos nesta fila, retornando quando será o próximo evento. Esta interface simples confere grande flexibilidade para os sub-perfis, permitindo uma grande quantidade de implementações diferentes.

Dentro da definição da carga de trabalho podem ser especificados um ou mais perfis de comportamento de nodos. Um desses perfis deve ser definido como o *perfil padrão*, que é atribuído a todos os nodos. Para atribuir um outro perfil a algum nodo, o número do nodo deve ser especificado. Os números dos nodos são números inteiros sequenciais que começam do valor zero. Na implementação atual, o nodo associado ao primeiro

controlador a conectar-se com o mestre recebe o identificador 0, o segundo nodo recebe o identificador 1 e assim por diante.

As classes do tipo `Profile` presentes na implementação atual são as seguintes: `None`, `Exact`, `Now`, `Interval`, `NTimesRandom`, `FullDictionary`, `PoissonRandom` e `WeibullRandom`. Nodos com o perfil `None` simplesmente não realizam a operação atribuída ao perfil. O perfil `Exact` permite especificar exatamente os momentos nos quais as operações serão executadas, portanto pode ser utilizado para fazer um mapeamento direto entre as cargas de trabalho observadas em aplicações reais e as cargas de trabalho desta ferramenta. O perfil `Now` executa a operação associada sempre que possível, tendo sua maior utilidade quando associado às operações `join`, possibilitando que os nodos, após saírem da DHT, retornem assim que possível.

Algumas cargas de trabalho podem exigir que as operações de `put` e `get` sejam efetuadas utilizando um conjunto de chaves comum a todos os nodos. Para resolver este problema, os controladores possuem um dicionário de chaves e valores pré-definidos que são referenciados pelos perfis a seguir. Cada perfil recebe como um de seus argumentos o tamanho do conjunto, especificando a quantidade de chaves a ser utilizada. Quando a operação associada for `put` ou `get`, uma chave escolhida aleatoriamente do dicionário pré-definido é utilizada. O perfil `Interval` recebe, além dos argumentos já mencionados, dois outros argumentos: o intervalo no qual as operações serão executadas e o número de operações a serem executadas entre cada intervalo. O perfil `NTimesRandom` é semelhante ao `Exact`, admitindo que sejam especificadas duplas contendo o momento em que a ação será realizada, e o número de operações que serão realizadas naquele momento. O perfil `FullDictionary` recebe uma lista contendo diversos momentos, sendo que em cada momento a operação associada será realizada uma vez para cada chave do dicionário pré-definido. Por fim, os perfis `PoissonRandom` e `WeibullRandom` fazem com que as operações associadas sejam executadas em uma frequência que segue as distribuições estatísticas de Poisson e Weibull, respectivamente.

Nem todos os perfis definidos podem ser associados aos quatro tipos de operações: alguns podem ser associados somente às operações `put` e `get`, enquanto outros podem ser associados somente às operações de `join` e `leave`.

O conjunto de perfis desenvolvido é apenas inicial e suficiente para suprir as necessidades dos testes de desempenho realizados no desenvolvimento deste trabalho. Adicionar novos perfis é bastante simples: o desenvolvedor precisa apenas implementar uma classe derivada de `Profile` que gerencia a fila de eventos já mencionada.

A Tabela 4.1 mostra o conteúdo de um arquivo que define uma carga de trabalho. Esta carga de trabalho tem duração de 100 segundos e o número sugerido de nodos é 4. Há dois perfis de nodos: o perfil `prof0`, que está atribuído somente ao nodo 0, e o perfil `prof1`, que é o perfil padrão, portanto está atribuído a todos os outros nodos.

O único nodo do perfil `prof0` tentará entrar na DHT, caso possa, em dois momentos:

1 e 70 segundos após o início do teste de desempenho, respectivamente. O mesmo nodo sairá da DHT em dois momentos exatos: 50 e 95 segundos após o início, respectivamente. Dez segundos após o início do teste de desempenho este nodo inserirá através da operação `put` todos os primeiros 50 pares de chave/valor pré-definidos no dicionário comum a todos os nodos. A partir do vigésimo segundo este mesmo nodo buscará aleatoriamente uma das 50 primeiras chaves pré-definidas pelo dicionário já mencionado com uma frequência que segue a distribuição estatística de Poisson, com lambda igual a 5.

Os outros nodos da DHT seguirão perfil denominado `prof1`. Este perfil estabelece que os nodos deverão tentar entrar na DHT sempre que puderem, mas, assim que entrarem, sairão da DHT após uma quantidade de tempo que segue a distribuição estatística de Weibull com parâmetros *forma* igual a 0.56 e *escala* igual a 97.62. Sessenta segundos após o início do teste, estes nodos inserirão na DHT a chave denominada `key1` com valor `value1`. O par de chave/valor `key2`/`value2` será inserido na DHT 65 segundos depois do início do teste de desempenho. Por fim, os nodos que seguem este perfil não utilizarão a operação `get`.

Tabela 4.1: Exemplo de arquivo de carga de trabalho

```
nodes 4
duration 100

profile prof0
join exact 1 70
leave exact 50 95
put fulldictionary 50 10
get poissonrandom 50 5 20

profile prof1
join now 1
leave weibullrandom 1 0.56 97.62
put exact 60 key1 value1 \
          65 key2 value2
get none

default-profile prof1
node 0 prof0
```

Para criar cargas de trabalho, basta ao desenvolvedor escrever um arquivo seguindo a mesma sintaxe do arquivo demonstrado na Tabela 4.1. Caso haja a necessidade de um comportamento não previsto pela implementação atual, basta implementar uma nova classe derivada de `Profile` seguindo as instruções já mencionadas e então referenciá-la no arquivo da carga de trabalho.

A implementação atual da ferramenta possui as cinco cargas de trabalho definidas na metodologia: *compartilhamento de arquivos*, *sem churn*, *alto churn*, *taxa de sucesso* e *propagação de chaves*, que foram nomeadas, respectivamente, `filesharing`, `no-churn`, `high-churn`, `successrate` e `key-spreading`.

Tabela 4.2: Cargas de trabalho implementadas

(a) filesharing

```
nodes 2
duration 600

profile prof0
join now 10-50
leave weibullrandom 1 0.56 97.62
put interval 100 10 240 50-90
get poissonrandom 5 100 60

profile prof1
join exact 1
leave none
put interval 100 10 240 50-90
get poissonrandom 5 100 60

default-profile prof0
node 0 prof1
```

(b) no-churn

```
nodes 5
duration 600

profile prof0
join exact 5
leave none
put poissonrandom 50 10 10
get poissonrandom 50 10 10

profile prof1
join exact 15-80
leave none
put poissonrandom 50 10 80
get poissonrandom 50 10 80

default-profile prof1
node 0 prof0
```

(c) high-churn

```
nodes 5
duration 600

profile prof0
join exact 5
leave none
put poissonrandom 50 10 10
get poissonrandom 50 10 10

profile prof1
join now 15-80
leave weibullrandom 1 0.56 97.62
put poissonrandom 50 10 80
get poissonrandom 50 10 80

default-profile prof1
node 0 prof0
```

(d) successrate

```
nodes 5
duration 600

profile prof0
join exact 5
leave exact 590
put fulldictionary 100 100
get fulldictionary 100 200-300

profile prof1
join exact 15-80
leave exact 590
put none
get fulldictionary 100 200-300

default-profile prof1
node 0 prof0
```

(e) key-spreading

```
nodes 5
duration 600

profile prof0
join exact 5
leave exact 590
put fulldictionary 100 10
get fulldictionary 100 250-350

profile prof1
join exact 150-240
leave exact 590
put none
get fulldictionary 100 250-350

default-profile prof1
node 0 prof0
```

4.3.3 Métricas

A implementação das métricas não é tão modular quanto a das cargas de trabalho ou das DHTs suportadas. Cada métrica exige que código para obtê-la seja inserido em pontos específicos da implementação, tanto na entidade mestre quanto na entidade controlador, o que dificulta o processo de modularização. Não há, portanto, restrições quanto à forma de implementação das métricas. Todas as métricas programadas são obtidas em todos os testes de desempenho, por isso cabe ao responsável pela execução dos testes decidir quais métricas são relevantes para cada carga de trabalho. Há, porém, uma mensagem definida pelo protocolo de comunicação entre mestre e controlador que serve para transportar dados arbitrários relacionados a métricas: a mensagem *Resultados*, ilustrada na Figura 4.3. A implementação atual da ferramenta Dhtperf possui três métricas: latência, taxa de sucesso e estado dos nodos, que são referenciadas pela ferramenta através dos nomes `Latency`, `SuccessRate` e `NodesStatus`, respectivamente. A métrica tráfego de rede, definida pela metodologia, ainda não possui uma implementação na ferramenta.

As métricas `Latency` e `SuccessRate` são obtidas pelo controlador e têm implementações triviais, seguindo as definições da metodologia. Cada controlador obtém um conjunto de resultados para cada uma das operações e, ao final da aplicação da carga de trabalho, transmite estes resultados para o mestre que, por fim, realiza o processamento final dos resultados.

Já a métrica `NodesStatus` é obtida pelo mestre. Após o envio das mensagens de *Início* o mestre espera que os controladores reportem seus resultados e encerrem seus resultados por uma quantidade de tempo equivalente à duração do teste de desempenho e mais alguns segundos (90 na implementação atual). Se, dentro deste intervalo de tempo, o controlador reportar o resultado de suas métricas e encerrar seu processamento corretamente (através das mensagens *Resultados* e *Fim*), este controlador é classificado como terminado (*Finished*). Se durante este intervalo de tempo a conexão TCP que o controlador mantém com o mestre for fechada inesperadamente, este controlador é classificado como falho (*Failed*). Por fim, se a conexão TCP não for encerrada mas os resultados não forem reportados a tempo, o controlador é classificado como não terminado (*Unfinished*).

4.3.4 Desenvolvimentos futuros

Em seu estado atual, a ferramenta Dhtperf já pode ser considerada pronta para ser utilizada em diversos tipos de avaliações de desempenho de DHTs, possuindo bastante flexibilidade para extensões e adaptações a novas DHTs, métricas e cargas de trabalho. Entretanto, ainda há algumas limitações e pontos que podem ser melhorados em versões futuras.

A limitação cujo impacto sobre os resultados das avaliações de desempenho é maior está relacionada com a operação `join`. Toda vez que um nodo necessita conectar-se

a uma DHT, o primeiro passo a ser realizado é entrar em contato com um outro nodo integrante da mesma DHT e utilizá-lo como *ponto de entrada*. Na implementação atual da ferramenta, o primeiro controlador a conectar-se com o mestre torna-se automaticamente o nodo *ponto de entrada* utilizado por todos os outros nodos que desejarem entrar na DHT. Assim, este nodo torna-se mais importante do que todos os outros, tendo maior influência sobre os resultados dos testes de desempenho. Se este nodo falhar, os resultados podem ser seriamente comprometidos. Se muitos outros nodos tentarem conectar-se à DHT simultaneamente, eles podem inundar o ponto de entrada com requisições, causando indiretamente um ataque de negação de serviço (*Denial of Service*, DoS).

Uma possível solução para este problema é fazer com que o mestre envie para todos os controladores uma lista com todos os possíveis nodos da DHT. Assim, toda vez que um nodo desejar conectar-se à DHT, ele pode escolher aleatoriamente um membro desta lista e tentar utilizá-lo como ponto de entrada. Caso o nodo escolhido como ponto de entrada não esteja conectado à DHT, cuidado especial deve ser tomado para que o nodo em questão aborte o processo de conexão ao invés de criar uma nova DHT paralela.

Outro problema da implementação atual está relacionado com a mensagem de *Início*, ilustrada na Figura 4.3. Após todos os controladores conectarem-se ao mestre e receberem suas mensagens de *Tipo de DHT* e *Perfil*, cabe ao mestre enviar aos controladores mensagens de *Início*, indicando o início da aplicação da carga de trabalho. O problema é que estas mensagens são enviadas sequencialmente a todos os controladores, portanto o primeiro a recebê-la iniciará a aplicação da carga de trabalho antes dos demais. Isto pode tornar-se um problema caso haja uma grande quantidade de nodos participantes no teste de desempenho ou então caso haja muita latência na rede.

Uma possível solução para este problema seria fazer com que a mensagem de *Início*, ao invés de indicar o início imediato da aplicação da carga de trabalho, agende este início para algum tempo no futuro. Assim, à medida que o mestre envia as mensagens aos controladores, ele pode ajustar o tempo agendado de acordo com o tempo decorrido desde o envio da primeira mensagem. Outra possível solução pode envolver o uso do *Network Time Protocol* [37] ou tecnologia semelhante. Ainda assim, nenhuma destas duas soluções garante que todos os controladores iniciarão as aplicações das cargas de trabalho exatamente no mesmo instante.

A métrica *tráfego de rede* foi identificada como uma das mais utilizadas na literatura. Entretanto, ela não está presente na implementação atual. A implementação necessária para obter esta métrica não é trivial, principalmente considerando o fato de que um dos princípios básicos da metodologia apresentada é a *não-intrusividade*: não é permitido alterar o código das DHTs analisadas. Versões futuras da ferramenta poderão conter a implementação desta métrica.

Como outra melhoria pode-se citar a troca do formato dos arquivos de carga de trabalho por um formato baseado em XML. Este formato, além de ser amplamente utili-

zado, permite definições da linguagem aceita através do uso de *Document Type Definition* (DTD) ou *XML Schema Definition* (XSD) [64].

CAPÍTULO 5

RESULTADOS EXPERIMENTAIS

Este capítulo descreve os resultados obtidos nos dois experimentos realizados com a ferramenta Dhtperf. Cada experimento consiste na execução de cada um dos cinco testes de desempenho descritos na Seção 4.1.3 sobre cada uma das seguintes DHTs: Open Chord, Overlay Weaver – em suas implementações de Chord, Kad e Pastry – e PastConsole – uma aplicação que utiliza as bibliotecas implementadas pelo projeto Free Pastry. Os experimentos são referenciados ao longo deste capítulo por nomes que representam os ambientes nos quais eles foram realizados: *Dinf* e *PlanetLab*. Os experimentos foram realizados em Julho de 2011, portanto as descrições dos ambientes referem-se às configurações destes na data mencionada.

O ambiente Dinf é composto por apenas uma máquina, localizada no Departamento de Informática da Universidade Federal do Paraná (UFPR). Portanto as métricas obtidas não são influenciadas pelos mesmos fatores que normalmente exercem influência em sistemas distribuídos. A máquina utilizada possui processadores AMD Opteron(tm) Processor 6136, com um total de 32 núcleos e 128GB de memória RAM. O sistema operacional utilizado é Debian GNU/Linux wheezy/sid com Kernel Linux versão 2.6.38. A máquina é compartilhada com outros usuários, sendo que no momento da realização dos experimentos cujos resultados são aqui apresentados havia diversos processos de outros usuários rodando, alguns deles consumindo completamente diversos núcleos de processador. Durante o experimento 200 controladores foram utilizados.

O ambiente PlanetLab é composto por uma rede de mais de 1000 máquinas espalhadas em mais de 500 sítios ao redor do mundo, conectadas através da Internet [8]. As máquinas do PlanetLab são heterogêneas, sendo que a maioria delas possui hardware comparável com desktops modernos do ano de 2011. Estas máquinas rodam o sistema operacional Fedora 8 e são compartilhadas entre diversos pesquisadores através de máquinas virtuais construídas com a tecnologia Linux VServer. Os experimentos foram realizados com um conjunto de 304 máquinas diferentes do PlanetLab, cada uma contendo no máximo um controlador. Apesar do conjunto possuir 304 máquinas (1 mestre e 303 controladores), cada teste de desempenho foi realizado com apenas 272 desses controladores (90% de 303). Esta margem foi estabelecida pois alguns controladores eventualmente apresentam problemas em conectarem-se ao mestre, devido a características do ambiente PlanetLab.

As versões das DHTs avaliadas foram as mesmas nos dois ambientes: Open Chord 1.0.5, Overlay Weaver 0.10.2 e Free Pastry 2.1. A DHT *kaddemo*, fornecida pela implementação MaidSafe DHT, e atualmente suportada pela ferramenta Dhtperf não foi

avaliada nos ambientes Dinf e PlanetLab. Em ambos os casos os conjuntos de bibliotecas e compiladores fornecidos pelos sistemas operacionais (Debian Sid e Fedora 8, respectivamente) não possibilitaram a compilação do software.

O restante deste capítulo está organizado da seguinte maneira. As Seções 5.1, 5.2 e 5.3 apresentam os resultados das métricas obtidas em cada um dos cinco testes de desempenho dos dois experimentos realizados, bem como as conclusões obtidas a partir da análise dos dados. A Seção 5.4 apresenta uma discussão geral sobre os resultados obtidos. Por fim, a Seção 5.5 apresenta uma avaliação da metodologia proposta e da ferramenta Dhtperf.

5.1 Métrica NodesStatus

A Tabela 5.1 sumariza os resultados obtidos pela métrica `NodesStatus` em cada uma das cinco cargas de trabalho de cada um dos dois ambientes. Cada linha representa uma implementação de DHT em uma carga de trabalho e cada coluna representa a quantidade de nodos classificada em cada um dos três estados (*Finished*, *Failed* e *Unfinished*), para cada um dos dois experimentos (Dinf e PlanetLab). Os dados da tabela também estão apresentados na forma de gráficos, presentes no Apêndice A (Figuras A.29 a A.38). Cada gráfico representa uma carga de trabalho de um ambiente e cada barra de cada gráfico representa uma das DHTs avaliadas. As barras são divididas em três partes, cada uma correspondente a cada um dos possíveis estados.

A DHT Open Chord claramente apresentou os piores resultados, mostrando-se inutilizável para as cargas de trabalho e número de nodos escolhidos. Nenhum dos nodos foi capaz de terminar nenhuma das cargas de trabalho, com exceção da carga `key-spreading` do ambiente Dinf, na qual 84% dos nodos foram classificados como *Finished*. Como em 9 dos 10 testes nenhum nodo foi classificado como *Finished*, os resultados desta DHT para as outras métricas destes 9 testes não foram reportados, portanto aparecem nas tabelas como nulos e são desconsiderados das análises realizadas e gráficos gerados. Uma análise realizada permitiu constatar que o problema da implementação utilizada do Open Chord é que, durante a realização dos experimentos, o software da DHT simplesmente trava ao realizar algumas operações `put` ou `get`, fazendo com que o controlador passe todo o resto do tempo esperando o resultado da operação. Detectamos que este problema começa a ocorrer com alguns nodos em DHTs que possuem aproximadamente 40 nodos, atingindo frações maiores de nodos conforme o tamanho da DHT cresce. Experimentos com 100 nodos ou mais resultam em praticamente todos os nodos detectados como *Unfinished*.

As DHTs implementadas pela ferramenta Overlay Weaver – Chord, Kad e Pastry – obtiveram a maior quantidade de nodos classificados como *Finished* em praticamente todas as cargas de trabalho de todos os ambientes, podendo assim ser consideradas as melhores DHTs sob o ponto de vista desta métrica. Apesar disso, somente estas DHTs apresentaram nodos caracterizados como *Failed*. Alguns nodos recebem este estado pois

Tabela 5.1: Resultados da métrica `NodesStatus`

DHT	Dinf			PlanetLab		
	Fini.	Fail.	Unfi.	Fini.	Fail.	Unfi.
filesharing						
Openchord	0	0	200	0	0	272
Owshellchord	199	0	1	234	25	13
Owshellkad	199	0	1	247	12	13
Owshellpastry	195	1	4	238	24	10
Pastconsole	181	0	19	170	0	102
no-churn						
Openchord	0	0	200	0	0	272
Owshellchord	200	0	0	266	0	6
Owshellkad	200	0	0	263	0	9
Owshellpastry	200	0	0	265	0	7
Pastconsole	200	0	0	241	0	31
high-churn						
Openchord	0	0	200	0	0	272
Owshellchord	197	0	3	237	17	18
Owshellkad	191	0	9	242	12	18
Owshellpastry	198	0	2	239	25	8
Pastconsole	181	0	19	139	0	133
successrate						
Openchord	0	0	200	0	0	272
Owshellchord	200	0	0	241	24	7
Owshellkad	199	1	0	265	0	7
Owshellpastry	199	1	0	239	25	8
Pastconsole	200	0	0	217	0	55
key-spreading						
Openchord	168	0	32	0	0	272
Owshellchord	199	1	0	245	21	6
Owshellkad	200	0	0	265	0	7
Owshellpastry	200	0	0	241	25	6
Pastconsole	198	0	2	247	0	25

Legenda: Fini. = Finished, Fail. = Failed, Unfi. = Unfinished.

em operações como `put` e `get` ocorre um erro e o programa `owdhtshell` é encerrado. Em todos os experimentos os resultados destas três DHTs foram muito semelhantes, sendo que em nenhum caso a diferença entre o número de nodos classificados como *Finished* nas três foi maior do que 10%. A maior diferença entre as três DHTs é que é que a implementação Kad tende a apresentar um menor número de nodos classificados como *Failed*, principalmente no ambiente PlanetLab.

A DHT Past Console apresentou um alto número de nodos *Unfinished*, principalmente nas duas cargas de trabalho com alto grau de churn. Os piores casos observados foram no ambiente PlanetLab com as cargas de trabalho `high-churn` e `filesharing`, onde o número de nodos classificados como *Unfinished* chegou a 48,8% e 37,5% do total, respectivamente. Isto ocorre pois eventualmente as operações `join` desta DHT demoram um grande tempo para terminar, fazendo com que o controlador fique esperando o resultado, estourando o tempo alocado para aplicar a carga de trabalho e retornar os resultados.

Quando conseguem terminar a tempo, em geral estas operações resultam em falhas.

A comparação entre as cinco cargas de trabalho permite observar que as cargas com maior grau de churn (`filesharing` e `high-churn`) apresentaram o maior número de nodos classificados como *Failed* e *Unfinished*. Por outro lado, a carga que apresentou a maior quantidade de nodos classificados como *Finished* foi a `no-churn`, que além de apresentar os melhores resultados para o ambiente PlanetLab não apresentou nenhum nodo *Failed* ou *Unfinished* no ambiente Dinf (desconsiderando a DHT Open Chord).

Por fim, ao comparar os resultados obtidos nos dois experimentos podemos constatar que o ambiente Dinf apresentou, proporcionalmente, mais nodos classificados como *Finished* do que o ambiente PlanetLab. Este resultado é esperado, já que o ambiente Dinf consiste em apenas uma máquina com grande capacidade de processamento, enquanto o ambiente PlanetLab está distribuído ao redor do planeta, sujeito a uma quantidade muito maior de fatores que podem comprometer os experimentos.

5.2 Métrica SuccessRate

A métrica `SuccessRate` foi obtida para as operações `join` e `get`, cujos resultados são apresentados nas Tabelas 5.2 e 5.3, respectivamente. Estas tabelas mostram, para cada ambiente avaliado, a taxa de sucesso detectada pelos nodos que obtiveram a menor e a maior taxa, além da média de todas das taxas obtidas em cada operação de cada nodo.

5.2.1 Operação `join`

A taxa de sucesso das operações `join` é relevante para todas as cargas de trabalho, porém mais relevante para `filesharing` e `high-churn`, pois estas cargas possuem alto grau de churn e consequentemente um maior número de operações `join` realizadas. Apesar disso não houve diferença significativa entre os resultados destas cargas de trabalho se comparadas com as outras: em todos os casos onde houve nodos classificados como *Finished*, exceto um, a média da taxa de sucesso das operações de todos os nodos foi igual ou maior a 98%. A única exceção foi a carga de trabalho `successrate` no ambiente PlanetLab com a DHT Past Console, onde a média das taxas de sucesso observadas foi de 41%.

Todos os nodos da DHT Open Chord foram classificados como *Unfinished* pela métrica `NodesStatus` em quase todos os testes realizados. Para estes testes os resultados das métricas não foram reportados por nenhum nodo, portanto as taxas de sucesso aparecem com o valor zero. O único resultado existente é da carga de trabalho `key-spreading` no ambiente Dinf, onde todas as operações `join` de todos os nodos classificados como `Finished` obtiveram sucesso.

Os resultados observados nas DHTs implementadas pelo Overlay Weaver tendo média

Tabela 5.2: Resultados da métrica `SuccessRate` para a operação `join`

DHT	Dinf			PlanetLab		
	Mín.	Méd.	Máx.	Mín.	Méd.	Máx.
filesharing						
Openchord	0	0	0	0	0	0
Owshellchord	100	100	100	27	98	100
Owshellkad	100	100	100	16	98	100
Owshellpastry	100	100	100	33	99	100
Pastconsole	100	100	100	88	99	100
no-churn						
Openchord	0	0	0	0	0	0
Owshellchord	100	100	100	25	98	100
Owshellkad	100	100	100	14	98	100
Owshellpastry	100	100	100	8	98	100
Pastconsole	100	100	100	100	100	100
high-churn						
Openchord	0	0	0	0	0	0
Owshellchord	100	100	100	40	99	100
Owshellkad	50	99	100	13	99	100
Owshellpastry	100	100	100	31	98	100
Pastconsole	100	100	100	100	100	100
successrate						
Openchord	0	0	0	0	0	0
Owshellchord	100	100	100	10	98	100
Owshellkad	100	100	100	11	99	100
Owshellpastry	100	100	100	25	99	100
Pastconsole	100	100	100	25	41	100
key-spreading						
Openchord	100	100	100	0	0	0
Owshellchord	100	100	100	11	98	100
Owshellkad	100	100	100	33	98	100
Owshellpastry	100	100	100	3	98	100
Pastconsole	100	100	100	100	100	100

Legenda: Mín. = Mínimo, Méd. = Médio, Máx. = Máximo, valores em porcentagem (%).

superior a 98% de sucesso em todos os testes. Para o ambiente Dinf todos os nodos de todas as cargas de trabalho obtiveram 100% de taxa de sucesso, exceto para a carga `high-churn`, onde a DHT Kad apresentou uma média de 99%, com um resultado mínimo de 50%. Para o ambiente PlanetLab a taxa de sucesso mínima em todas as cargas de trabalho foi menor ou igual a 40%, chegando a 3% no pior caso, obtido pela DHT Pastry na carga de trabalho `key-spreading`.

Já a DHT Past Console apresentou os melhores resultados para esta métrica, obtendo 100% de sucesso para todos os nodos em 8 dos 10 testes de desempenho. As únicas cargas de trabalho onde o sucesso absoluto não foi atingido foram as cargas `filesharing` e `successrate` do ambiente PlanetLab, tendo como resultados mínimos 88% e 25% e como resultados médios 99% e 41%, respectivamente. Entretanto, apesar dos resultados desta métrica para a DHT aparentarem ser melhores do que os das outras DHT, deve-se lembrar que algumas operações `join` realizadas demoram muito tempo para terminar, o que em

muitos casos estoura o tempo limite estabelecido para os testes de desempenho, fazendo os nodos em questão serem classificados como *Unfinished* e terem suas métricas descartadas. Assim, os problemas da operação `join` desta DHT apresentam suas consequências na métrica `NodesStatus` e não na métrica `SuccessRate`.

Por fim, a comparação entre os resultados dos ambientes Dinf e PlanetLab mostrou o que já era esperado: as taxas de sucesso em um ambiente onde todos os nodos encontram-se na mesma máquina são maiores do que em um ambiente distribuído ao redor do planeta, conectado através da Internet.

5.2.2 Operação `get`

A taxa de sucesso das operações `get`, apesar de ter sido medida para todas as cargas de trabalho, tem relevância somente para as cargas `successrate` e `key-spreading`, pois espera-se que nestas cargas o resultado obtido seja de 100% de sucesso. Nas outras cargas de trabalho as operações `put` e `get` são realizadas com chaves escolhidas aleatoriamente, portanto ao realizar um `get` não há como saber se a chave já foi inserida ou não, logo não há como prever qual o resultado esperado para a métrica.

A carga de trabalho `successrate` apresentou bons resultados, tanto para as DHTs implementadas pelo Overlay Weaver quanto para a DHT Past Console. A menor média obtida foi de 91%, com a DHT Chord do Overlay Weaver no ambiente PlanetLab. Dentre os valores mínimos obtidos, os piores são das DHTs implementadas pelo Overlay Weaver no ambiente PlanetLab, estando um pouco abaixo dos 20%. O único caso onde o valor máximo não foi 100% foi para a DHT Chord do Overlay Weaver, no qual o resultado obtido foi 97%. Para o ambiente Dinf os melhores resultados foram obtidos com as DHTs implementadas pelo Overlay Weaver, onde foi obtido 100% de sucesso em todos os casos. Ainda no ambiente Dinf, a DHT Past Console obteve um valor mínimo de 97% e médio de 99%. Já para o ambiente PlanetLab a DHT Past Console mostrou-se superior, obtendo uma taxa de sucesso mínima de 88% e uma média de 99%. A DHT Open Chord não pode ser comparada pois não apresentou nenhum nodo classificado como *Finished* nas duas aplicações desta carga de trabalho.

Já a carga de trabalho `key-spreading` apresentou resultados abaixo dos esperados. Para o ambiente Dinf, a DHT Openchord obteve 168 nodos classificados como *Finished*, portanto os resultados para métrica foram obtidos. Entretanto, todos os 168 nodos reportaram uma taxa de sucesso de 0% em suas operações `get`, o que indica que, apesar dos nodos terem conseguido realizar este teste, algum outro problema existe, pois nenhuma das chaves inseridas foi encontrada.

Para as DHTs implementadas pelo Overlay Weaver outro problema foi detectado: a melhor média obtida para a carga de trabalho `key-spreading` foi de 1% e a maior taxa de sucesso foi de apenas 3%. Isto ocorre pois as DHTs utilizadas neste trabalho utilizam as

Tabela 5.3: Resultados da métrica `SuccessRate` para a operação `get`

DHT	Dinf			PlanetLab		
	Mín.	Méd.	Máx.	Mín.	Méd.	Máx.
filesharing						
Openchord	0	0	0	0	0	0
Owshellchord	0	55	100	0	41	100
Owshellkad	20	75	100	0	73	100
Owshellpastry	0	59	100	0	60	100
Pastconsole	25	80	100	20	87	100
no-churn						
Openchord	0	0	0	0	0	0
Owshellchord	86	99	100	18	91	100
Owshellkad	87	99	100	22	96	100
Owshellpastry	86	99	100	26	98	100
Pastconsole	92	99	100	91	99	100
high-churn						
Openchord	0	0	0	0	0	0
Owshellchord	75	89	100	18	75	94
Owshellkad	80	94	100	33	92	100
Owshellpastry	79	94	100	27	93	100
Pastconsole	88	97	100	80	96	100
successrate						
Openchord	0	0	0	0	0	0
Owshellchord	100	100	100	18	91	97
Owshellkad	100	100	100	16	96	100
Owshellpastry	100	100	100	17	97	100
Pastconsole	97	99	100	88	99	100
key-spreading						
Openchord	0	0	0	0	0	0
Owshellchord	0	0	0	0	0	1
Owshellkad	1	1	1	0	0	0
Owshellpastry	0	0	0	0	0	3
Pastconsole	97	98	100	0	99	100

Legenda: Mín. = Mínimo, Méd. = Médio, Máx. = Máximo, valores em porcentagem (%).

configurações padrões de seus desenvolvedores e a configuração padrão do Overlay Weaver não é adaptada para ter uma boa tolerância a churn. O autor do Overlay Weaver divulga um *patch* que, se aplicado, aumenta o grau de tolerância a churn, também aumentando o tráfego de rede gerado [21]. Este patch, além de habilitar a inserção de réplicas das chaves adicionadas, habilita uma rotina periódica de re-inserção das réplicas (operação denominada *reput* pela ferramenta). Os efeitos da operação de *reput* do Overlay Weaver em uma carga de trabalho muito semelhante à `key-spreading` já foram estudados em uma versão inicial deste trabalho e publicados em [65].

Por fim, o único caso onde os resultados obtidos com a carga de trabalho `key-spreading` foram próximos do esperado foi com a DHT Past Console, que obteve um valor mínimo de 97%, médio de 98% e máximo de 100% para o ambiente Dinf. Já no ambiente PlanetLab, o valor mínimo foi de 0%, o médio de 99% e o máximo de 100% para a mesma DHT.

5.3 Métrica Latency

Os resultados da métrica Latency são apresentados na forma de gráficos de função distribuição acumulada. Estes gráficos encontram-se no Apêndice A. Cada linha de cada gráfico representa uma operação de uma DHT em uma carga de trabalho. Para uma dada latência L, o valor do eixo *Distribuição acumulada* representa a porcentagem de operações que obtiveram uma latência menor ou igual a L. Para facilitar a leitura, as linhas do gráfico possuem diferentes formas geométricas associadas. Cada forma geométrica representa uma DHT e nos casos onde há mais de uma operação ou carga de trabalho para a mesma DHT, a forma geométrica permanece a mesma, alterando apenas o seu preenchimento. Um detalhe a ser observado é a escala de cada gráfico: os valores mínimo e máximo eixo X variam de acordo com o ambiente (Dinf ou PlanetLab) e de acordo com as operações plotadas (`join` e `leave` ou `put` e `get`).

Como exemplo de interpretação dos gráficos podemos citar a Figura A.1. Nesta figura as linhas que possuem formas geométricas preenchidas de preto representam as latências das operações `join` e as linhas que possuem formas geométricas não preenchidas representam as latências das operações `leave`. Para o caso das operações `leave`, 100% das operações realizadas em todas as DHTs apresentaram latência menor do que 750 milissegundos. Pode-se observar também que, por exemplo, aproximadamente 45% das operações `join` realizadas pela DHT Overlay Weaver Pastry apresentaram latência inferior a 1250 milissegundos. O fato de nenhuma das linhas que representam as operações `join` chegar ao topo do gráfico significa que todas as DHTs apresentaram algumas operações `join` com latências maiores do que 7500 milissegundos. Ainda sobre as operações `join`, pode-se dizer que a DHT Overlay Weaver Pastry apresentou os melhores resultados pois ela possui uma quantidade maior de operações que apresentaram baixas latências. Pode-se dizer também que a DHT Overlay Weaver Chord apresentou o segundo melhor resultado para a operação `join`, seguida da DHT Overlay Weaver Kad, em terceiro lugar, e da DHT Past Console, em quarto.

As latências das operações `join` e `leave` observadas nas DHTs para cada uma das cargas de trabalho são apresentadas nas Figuras A.1 até A.5 para o ambiente Dinf e A.11 até A.15 para o ambiente PlanetLab. As latências das operações `put` e `get` observadas nas DHTs para cada uma das cargas de trabalho são apresentadas nas Figuras A.6 até A.10 para o ambiente Dinf e A.16 até A.20 para o ambiente PlanetLab. Comparações das latências de cada uma das operações entre as cargas de trabalho `no-churn` e `high-churn` são apresentadas nas Figuras A.21 até A.24 para o ambiente Dinf e A.25 até A.28 para o ambiente PlanetLab.

Ao comparar os desempenhos das operações `join` no ambiente Dinf (Figuras A.1 a A.5) não é possível identificar claramente uma DHT pior ou melhor. A implementação Pastry do Overlay Weaver obteve sempre baixas latências se comparada com as outras

DHTs, sendo a única não que apresentou o pior resultado em nenhuma das cinco cargas de trabalho. A DHT Kad do Overlay Weaver também obteve baixas latências, porém teve um desempenho significativamente inferior ao das outras DHTs para a carga de trabalho `high-churn` (Figura A.3). Na única carga de trabalho onde a DHT Open Chord apresentou resultados (`key-spreading`, ambiente Dinf) seu desempenho foi o pior para a operação `join` em comparação ao das outras DHTs (Figura A.5).

Já para o ambiente PlanetLab a o resultado da latência das operações `join` (Figuras A.11 até A.15) apresentou um padrão perceptível em todos os gráficos: as menores latências foram observadas com a DHT Pastry implementada pelo Overlay Weaver, seguida da DHT Chord implementada pela mesma ferramenta. O terceiro melhor desempenho foi observado na DHT Kad do Overlay Weaver e as maiores latências foram observadas na DHT Past Console.

As cargas de trabalho `filesharing` e `high-churn`, por possuírem alto grau de churn, apresentam um maior número de operações `join` do que as outras cargas de trabalho (Figuras A.1, A.3, A.11 e A.13). Para estas duas cargas a DHT que sempre apresentou as menores latências nos dois ambientes foi a implementação Pastry do Overlay Weaver. O segundo melhor resultado foi também sempre da DHT Chord implementada pelo Overlay Weaver. Em três dos quatro casos (Figuras A.1, A.11 e A.13) a implementação Kad do Overlay Weaver obteve o terceiro lugar, seguida pela DHT Past Console.

As operações `leave` foram implementadas na ferramenta em nível de sistema operacional, através de sinais que finalizam os processos das DHTs. Consequentemente, a latência observada para essas operações foi extremamente baixa, o que dificulta as comparações (Figuras A.1 até A.5 e A.11 até A.15). Os únicos casos onde a latência das operações `leave` foi maior do que o usual foram nas cargas de trabalho `successrate` e `key-spreading` do ambiente Dinf (Figuras A.4 e A.5). Nessas cargas todos os nodos saem da DHT exatamente 590 segundos após o início da aplicação da carga de trabalho. Como neste ambiente todos os nodos estão executando na mesma máquina, todos os 200 processos dos nodos tentam encerrar ao mesmo tempo e acabam sobrecarregando os recursos do sistema operacional, resultando em latências acima das convencionais. Não há gráficos de latência para esta operação na carga de trabalho `no-churn` pois nesta carga nenhum nodo executa esta operação (Figuras A.2 e A.12).

Para a latência das operações `put` pode-se observar dois padrões de resultados, um para o ambiente Dinf (Figuras A.6 a A.10) e outro para o PlanetLab (Figuras A.16 a A.20). No ambiente Dinf as menores latências foram observadas na implementação Pastry do Overlay Weaver. As DHTs Past Console e Overlay Weaver Chord apresentaram resultados bastante semelhantes, sendo também próximos da Overlay Weaver Pastry, porém ligeiramente piores. Por fim, os piores resultados foram observados na DHT Overlay Weaver Kad. Já no ambiente PlanetLab as duas implementações da DHT Pastry obtiveram as menores latências, apresentando resultados semelhantes. Em terceiro e quarto

lugar encontram-se, respectivamente, as DHTs Chord e Kad implementadas pelo Overlay Weaver.

Dentre os casos discrepantes da operação `put` destacam-se dois. Na carga de trabalho `key-spreading` todas as chaves são inseridas enquanto a DHT possui apenas um nodo conectado, portanto as latências apresentadas são muito menores do que as usuais (Figuras A.10 e A.20). Outro fato relevante é que na carga de trabalho `successrate` do ambiente PlanetLab o único nodo responsável pela operação `put` não conseguiu reportar suas métricas, portanto o resultado não é mostrado na Figura A.19. Apesar dos resultados não terem sido reportados, pode-se afirmar que as operações foram realizadas corretamente ao observar que os resultados da métrica `SuccessRate` para a operação `get`, presentes na Tabela 5.3, foram maiores do que zero.

Com relação à operação `get`, ao comparar os resultados obtidos e ordenar as DHTs de melhor a pior, pode-se chegar a uma ordem relativa igual à observada na operação `put`, tanto para o ambiente Dinf (Figuras A.6 a A.10) quanto para o ambiente PlanetLab (Figuras A.16 a A.20). É importante ressaltar a carga de trabalho `key-spreading` do ambiente Dinf, ilustrada na Figura A.10, que é a única carga a apresentar resultados da operação `get` para a DHT Open Chord. Neste caso as latências observadas pela DHT foram ligeiramente inferiores do que da DHT Kad implementada pelo Overlay Weaver, porém foram bem maiores do que das outras três DHTs analisadas.

Ao comparar os resultados das latências das operações `put` com as operações `get` (Figuras A.6 a A.10 e A.16 a A.20) pode-se observar que de maneira geral as operações `put` apresentam latências um pouco maiores, porém com curvas semelhantes às das operações `get` de suas respectivas DHTs. A principal exceção é a carga de trabalho `key-spreading`, na qual as latências de `put` são muito menores pois estas operações são realizadas quando a DHT possui somente um nodo conectado (Figuras A.10 e A.20).

Por fim podemos comparar os resultados da métrica para as quatro operações entre as cargas de trabalho `no-churn` e `high-churn` (Figuras A.21 a A.28). Com relação à operação `join` (Figuras A.21 e A.25), em 5 das 8 medições os resultados obtidos com a carga de trabalho `no-churn` apresentam menores latências (todos os casos exceto Overlay Weaver Chord para os ambiente Dinf e PlanetLab e Overlay Weaver Pastry para o ambiente PlanetLab). Já a operação `leave` (Figuras A.22 e A.26) apresentou latências muito baixas para todos os resultados, dificultando maiores comparações. Tanto nas operações `put` (Figuras A.23 e A.27) como nas operações `get` (Figuras A.24 e A.28) as latências obtidas com a carga de trabalho `no-churn` foram menores do que as obtidas com a carga de trabalho `high-churn`.

5.4 Discussão

Uma análise geral dos resultados obtidos permite identificar pontos fortes e fracos em cada uma das DHTs. A DHT Overlay Weaver Pastry, por exemplo, apresentou baixas latências se comparada com as outras implementações avaliadas, mas apresentou *bugs* nas operações `put` e `get` que levaram o software da DHT terminar sua execução. Além disso, os resultados obtidos por esta DHT foram menores do que os esperados para a taxa de sucesso da operação `get` na carga de trabalho `key-spreading`, o que mostra que a configuração padrão da DHT pode apresentar problemas quando a rede apresenta churn. A DHT Past Console, apesar de apresentar bons resultados em relação às outras DHTs para as taxas de sucesso e as latências das operações `get`, apresentou problemas de latência com a operação `join`. Já a DHT Overlay Weaver Kad, apesar de apresentar altas latências, obteve um alto número de nodos classificados como *Finished* na métrica `NodesStatus`, o que indica que ela pode ser uma das menos afetadas por *bugs* no software. Por outro lado, a DHT Open Chord não mostrou-se melhor em nenhum caso.

Outro fato observado é que o que importa não é somente o tipo de DHT escolhido (Chord, Pastry, Kad ou outras), mas também a implementação realizada. Enquanto a implementação Chord realizada pelo Overlay Weaver mostrou-se bastante regular – na maioria dos casos analisados ela não foi nem a melhor nem a pior –, a implementação do Open Chord apresentou diversos *bugs*, tornando-a inutilizável em praticamente todos os testes de desempenho realizados. Já as duas implementações da DHT Pastry analisadas, apesar de terem obtido bons resultados, apresentaram pontos fortes e fracos distintos. Ainda, acreditamos que simples mudanças nos códigos de todas as DHTs analisadas podem fazer os resultados melhorarem significativamente: Open Chord possui bugs que fazem com que operações `put` e `get` nunca terminem; Overlay Weaver possui bugs que fazem com que o software da DHT encerre inesperadamente, além de possuir uma configuração padrão com baixo grau de tolerância a churn e a DHT Past Console apresenta problemas com a operação `join`.

Apesar disso, concluímos também que o tipo de DHT escolhido faz diferença. As três DHTs da ferramenta Overlay Weaver analisadas compartilham grande parte de seus códigos-fonte, mesmo assim os resultados obtidos entre essas DHTs apresentaram diferenças significativas, como por exemplo nos resultados da métrica `Latency`.

5.5 Avaliação da metodologia e da ferramenta

O principal objetivo das avaliações de desempenho realizadas pela ferramenta Dhtperf apresentadas neste capítulo é a validação da metodologia proposta. Conseguimos mostrar que a metodologia pode ser implementada e é capaz de permitir a avaliação de desempenho de diversas implementações de DHTs, realizadas por diferentes autores, sem exigir

modificações nas mesmas. Mostramos também que a ferramenta pode ser utilizada tanto em ambientes emulados como o Dinf quanto em ambientes reais, como o PlanetLab.

O conjunto escolhido de métricas e cargas de trabalho foi suficiente para detectar uma série de pontos fortes e fracos nas diversas DHTs analisadas, além de pontos que podem ser melhorados em cada uma delas. Apesar de esperarmos que um conjunto maior de métricas e cargas nos permita identificar ainda mais diferenças, acreditamos que o conjunto definido pela nossa metodologia já é suficiente para constituir uma avaliação de desempenho de DHTs que possa ser utilizada por possíveis interessados. É importante ressaltar que a métrica *tráfego de rede*, definida pela metodologia, ainda não foi implementada na ferramenta. Acreditamos também que muitos dos testes de desempenho realizados pelos trabalhos estudados no Capítulo 3 podem ser reproduzidos com a ferramenta Dhtperf, através da definição de cargas de trabalho baseadas nas originais.

Por fim, é importante também ressaltar a facilidade de utilização da ferramenta. Tanto para o ambiente Dinf como para o ambiente PlanetLab a realização de uma avaliação de desempenho completa, incluindo todas as DHTs e cargas de trabalho, pode ser realizada através de simples scripts Bash. Os scripts utilizados nos dois ambientes estão disponíveis junto com o código fonte da ferramenta, bem como os pacotes RPM utilizados. Assim, reproduzir os experimentos realizados é uma tarefa simples. Além disso, o código fonte da ferramenta está disponível e sob a licença GPL, o que permite que qualquer um possa modificá-lo, adaptando-o a novas DHTs, implementando novas métricas, cargas de trabalho e melhorando-o de maneira geral.

CAPÍTULO 6

CONCLUSÃO

As DHTs são sistemas P2P estruturados que tornaram-se bastante populares ao longo dos últimos anos, possuindo diversos tipos de aplicações, algumas delas com milhões de nodos. Existem diversos tipos de DHTs, cada tipo pode possuir diversas implementações e cada uma dessas implementações pode ser configurada através de diversos parâmetros que alteram o seu funcionamento. Ainda não há um consenso sobre a melhor maneira de avaliar o desempenho de uma DHT, o que dificulta a comparação entre o grande número de DHTs existentes e seus parâmetros.

Nos últimos anos foram apresentados diversos trabalhos que realizam avaliações de desempenho de DHTs, além de alguns trabalhos que propõem metodologias para tal. Esta dissertação estudou esses diversos trabalhos, apresentando suas semelhanças e diferenças. Com base nestes pontos comuns identificados foi proposta uma nova metodologia de avaliação de desempenho de DHTs. Esta metodologia define um conjunto de testes de desempenho que possui métricas e cargas de trabalho que são baseadas nas mais comuns dentre os trabalhos estudados. A metodologia define também um método para a execução dos testes de desempenho, composto por três entidades: *mestre*, *controlador* e *nodo*. Até onde pudemos identificar, nosso trabalho é o primeiro a propor uma metodologia de avaliação de desempenho de DHTs baseada nas metodologias já propostas e avaliações já realizadas por outros trabalhos.

Este trabalho apresentou também Dhtperf, uma implementação para a metodologia proposta. Esta implementação permite a avaliação de desempenho de DHTs existentes em sistemas reais e emulados sem a necessidade de modificação no software das mesmas. Além de implementar as métricas e cargas de trabalho definidas pela metodologia, Dhtperf permite também a definição de novas métricas e novas cargas de trabalho de uma maneira simples. Dhtperf é um software livre, portanto estas novas definições podem ser realizadas por qualquer pessoa.

Dhtperf foi utilizada para avaliar o desempenho de DHTs existentes – Open Chord, Overlay Weaver e Free Pastry – em dois ambientes, um deles composto por apenas uma máquina possuindo 200 nodos e o outro composto por aproximadamente 300 máquinas espalhadas ao redor do mundo – integrantes da rede PlanetLab –, cada uma responsável por um nodo. Os resultados obtidos permitiram identificar pontos fortes e fracos em cada DHT analisada, validando a metodologia proposta e sua implementação. Como cada implementação tem seus pontos fortes e fracos, não é possível identificar uma das implementações como sendo a melhor ou pior. Identificamos também problemas nas im-

plementações analisadas que, se corrigidos, podem melhorar os resultados das avaliações. Por fim, os experimentos nos levam a acreditar que tanto o tipo de DHT escolhida quanto a implementação realizada fazem diferença nos resultados das avaliações.

Como possíveis trabalhos futuros podemos citar a resolução dos problemas identificados na ferramenta implementada, além da definição de novas métricas, cargas de trabalho e DHTs suportadas. Esperamos também que tanto a metodologia quanto a ferramenta sejam utilizadas em trabalhos que de alguma forma necessitam avaliar o desempenho de DHTs.

BIBLIOGRAFIA

[1] Stephanos Androutsellis-Theotokis e Diomidis Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

[2] Christopher Batten, Kenneth Barr, Arvind Saraf, e Stanley Trepetin. pStore: A Secure Peer-to-Peer Backup System. Relatório Técnico LCS Technical Memo 632, Massachusetts Institute of Technology Laboratory for Computer Science, 2001.

[3] Kenneth P. Birman. *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer, 2005.

[4] Bittorrent. `http://www.bittorrent.com/`. Acessado em 7 de agosto de 2011.

[5] Frederik Bjurefors, Lars A. Larzon, e Richard Gold. Performance of pastry in a heterogeneous system. *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, páginas 278–279, 2004.

[6] Miguel Castro, Manuel Costa, e Antony Rowstron. Debunking some myths about structured and unstructured overlays. *Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation (NSDI'05)*, páginas 85 – 98, Berkeley, CA, USA, 2005. USENIX Association.

[7] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, e Antony Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):100 – 110, 2002.

[8] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, e Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[9] Ian Clarke, Oskar Sandberg, Brandon Wiley, e Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, páginas 46–66, 2001.

[10] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, e Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[11] George Coulouris, Jean Dollimore, e Tim Kindberg. *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[12] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, e Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, February de 2003.

[13] Dhtperf. `https://gitorious.org/dhtperf`. Acessado em 9 de agosto de 2011.

[14] eMule. `http://www.emule-project.net`. Acessado em 7 de agosto de 2011.

[15] Free Pastry. `http://www.freepastry.org/`. Acessado em 7 de agosto de 2011.

[16] Free Software Definition. `http://www.gnu.org/philosophy/free-sw.html`. Acessado em 7 de agosto de 2011.

[17] GNU General Public License Version 3. `http://www.gnu.org/licenses/gpl-3.0.html`. Acessado em 7 de agosto de 2011.

[18] Gnutella Protocol Development. `http://rfc-gnutella.sourceforge.net/`. Acessado em 7 de agosto de 2011.

[19] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, e Robert Van Renesse. Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. *Lecture Notes in Computer Science*, páginas 160–169, 2003.

[20] Cyrus Harvesf e Douglas M. Blough. The Design and Evaluation of Techniques for Route Diversity in Distributed Hash Tables. *Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing*, páginas 237 – 238. Citeseer, 2007.

[21] Histórico da lista de e-mails overlayweaver-discuss. `http://sourceforge.net/mailarchive/forum.php?forum_name=overlayweaver-discuss`. Acessado em 7 de agosto de 2011.

[22] Raj Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[23] M. Frans Kaashoek e David R. Karger. Koorde: A simple degree-optimal distributed hash table. *Lecture Notes in Computer Science*, páginas 98–107, 2003.

[24] Daishi Kato e Toshiyuki Kamiya. Evaluating DHT Implementations in Complex Environments by Network Emulator. *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS 2007)*, 2007.

[25] Kazaa. `http://www.kazaa.com`. Acessado em 7 de agosto de 2011.

[26] Joseph S. Kong, Jesse S. A. Bridgewater, e Vwani P. Roychowdhury. A General Framework for Scalability and Performance Analysis of DHT Routing Systems. *Proceedings of the International Conference on Dependable Systems and Networks*, páginas 343–354, Washington, DC, USA, 2006. IEEE Computer Society.

[27] Aleksandra Kovacevic, Kalman Graffi, Sebastian Kaune, Christof Leng, e Ralf Steinmetz. Towards Benchmarking of Structured Peer-to-Peer Overlays for Network Virtual Environments. *Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems*, páginas 799–804, Washington, DC, USA, 2008. IEEE Computer Society.

[28] Xiuqi Lee e Jie Wu. Searching Techniques in Peer-to-Peer Networks. Jie Wu, editor, *Handbook of Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, páginas 617 – 643. Auerbach Publications, Boca Raton, 2006.

[29] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris, e M. Frans Kaashoek. Comparing the performance of distributed hash tables under churn. *Proceedings of the 2nd Bertinoto Workshop on Future Directions in Distributed Computing (FuDiCo II): Survivability: Obstacles and Solutions, Bertinoro, Italy*, páginas 87–99. Citeseer, 2004.

[30] Jinyang Li, Jeremy Stribling, Robert Morris, e M. Frans Kaashoek. Bandwidth-efficient management of DHT routing tables. *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation*, NSDI'05, páginas 99–114, Berkeley, CA, USA, 2005. USENIX Association.

[31] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, e Steven Lim. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Communications Surveys and Tutorials*, 7(1 – 4):72 – 93, 2005.

[32] MaidSafe DHT. `http://code.google.com/p/maidsafe-dht/`. Acessado em 7 de agosto de 2011.

[33] Petros Maniatis, Mema Roussopoulos, T. J. Giuli, David S. H. Rosenthal, e Mary Baker. The LOCKSS Peer-to-Peer Digital Preservation System. *ACM Transactions on Computer Systems (TOCS)*, 23(1):50, 2005.

[34] Gurmeet Singh Manku, Mayank Bawa, e Prabhakar Raghavan. Symphony: Distributed Hashing in a Small World. *USENIX Symposium on Internet Technologies and Systems (USITS)*. Stanford InfoLab, 2003.

[35] Petar Maymounkov e David Mazières. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS02)*, 1:53 – 65, 2002.

[36] Ghulam Memon. Characterizing Traffic in Widely-Deployed DHT. Relatório técnico, University of Oregon, 2008.

[37] NTP: The Network Time Protocol. `http://www.ntp.org/`. Acessado em 7 de agosto de 2011.

[38] National Institute of Standards e Technology. Federal information processing standards publication 180-1. Relatório técnico, United States Department of Commerce, May de 1995.

[39] Open Chord. `http://open-chord.sourceforge.net/`. Acessado em 7 de agosto de 2011.

[40] David Oppenheimer, Vitaliy Vatkovskiy, e David A. Patterson. Towards a framework for automated robustness evaluation of distributed services. *Proceedings of the 2nd Bertinoto Workshop on Future Directions in Distributed Computing (FuDiCo II): Survivability: Obstacles and Solutions, Bertinoro, Italy*. Citeseer, 2004.

[41] David Oppenheimer, Vitaliy Vatkovskiy, Hakim Weatherspoon, Jason Lee, David A. Patterson, e John Kubiatowicz. Monitoring, Analyzing, and Controlling Internet-scale Systems with ACME. Relatório técnico, UC Berkeley Technical Report UCB-CSD-03-1276, 2003.

[42] Overlay Weaver. `http://overlayweaver.sourceforge.net/doc/overview/#runtime`. Acessado em 7 de agosto de 2011.

[43] Overlay Weaver. `http://overlayweaver.sourceforge.net`. Acessado em 7 de agosto de 2011.

[44] Overnet. `http://www.overnet.org`. Acessado em 7 de agosto de 2011.

[45] p2psim. `http://pdos.csail.mit.edu/p2psim/`. Acessado em 7 de agosto de 2011.

[46] Vasileios Pappas, Dan Massey, Andreas Terzis, e Lixia Zhang. A Comparative Study of the DNS Design with DHT-Based Alternatives. *Proceedings of IEEE International Conference on Computer Communications (INFOCOM'06)*, páginas 23 – 29, 2006.

[47] C. Greg Plaxton, Rajmohan Rajaraman, e Andréa W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. *Theory of Computing Systems*, 32(3):241–280, 1999.

[48] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, e Scott Schenker. A Scalable Content-Addressable Network. *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, páginas 161 – 172. ACM, 2001.

[49] Sean Rhea, Dennis Geels, Timothy Roscoe, e John Kubiatowicz. Handling Churn in a DHT. *Proceedings of the USENIX Annual Technical Conference*, páginas 127–140, 2004.

[50] Sean C. Rhea, Timothy Roscoe, e John Kubiatowicz. Structured peer-to-peer overlays need application-driven benchmarks. *Lecture notes in computer science*, páginas 56–67, 2003.

[51] R. Rivest. RFC1321: The MD5 Message-Digest Algorithm. *RFC Editor*, 1992.

[52] Antony Rowstron e Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 11, páginas 329 – 350. Citeseer, 2001.

[53] Stefan Saroiu, Krishna P. Gummadi, e Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. *Multimedia Computing and Networking (MMCN)*, January de 2002.

[54] SETI@home. `http://setiathome.berkeley.edu/`. Acessado em 7 de agosto de 2011.

[55] Kazuyuki Shudo. Churn Tolerance Improvement Techniques in an Algorithm-neutral DHT. `http://www.shudo.net/publications/AIMS-2009-churn-resilience/shudo-AIMS2009-slides-churn-resilience.pdf`. Acessado em 7 de agosto de 2011.

[56] Abraham Silberschatz, Peter Baer Galvin, e Greg Gagne. *Operating System Concepts*. Wiley Publishing, 2001.

[57] Skype: P2P Telephony Explained. `https://support.skype.com/pt-br/faq/FA10983/O-que-sao-comunicacoes-P2P`. Acessado em 7 de agosto de 2011.

[58] Moritz Steiner, Taoufik En-Najjary, e Ernst W. Biersack. Long Term Study of Peer Behavior in the KAD DHT. *IEEE/ACM Transactions on Networking*, 17(5):1371 – 1384, 2009.

[59] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, e Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, páginas 149 – 160. ACM, 2001.

[60] Andrew S. Tanenbaum e Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[61] The Chord/DHash Project. `http://pdos.csail.mit.edu/chord/`. Acessado em 7 de agosto de 2011.

[62] The Oceanstore Project. `http://oceanstore.cs.berkeley.edu/`. Acessado em 7 de agosto de 2011.

[63] The Peer-to-Peer Trace Archive. `http://p2pta.ewi.tudelft.nl/pmwiki/`. Acessado em 7 de agosto de 2011.

[64] W3C: XML Schema. `http://www.w3.org/standards/xml/schema`. Acessado em 7 de agosto de 2011.

[65] Paulo Ricardo Zanoni, Luis Carlos Erpen de Bona, e Eduardo Cunha de Almeida. Proposta de uma metodologia de avaliação de sistemas peer-to-peer baseados em tabelas hash distribuídas. *Anais do XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2010.

[66] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, e John D. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on selected areas in communications*, 22(1):41 – 53, 2004.

APÊNDICE A

GRÁFICOS

Este apêndice contém os gráficos gerados pela ferramenta Dhtperf nos experimentos descritos no Capítulo 5.

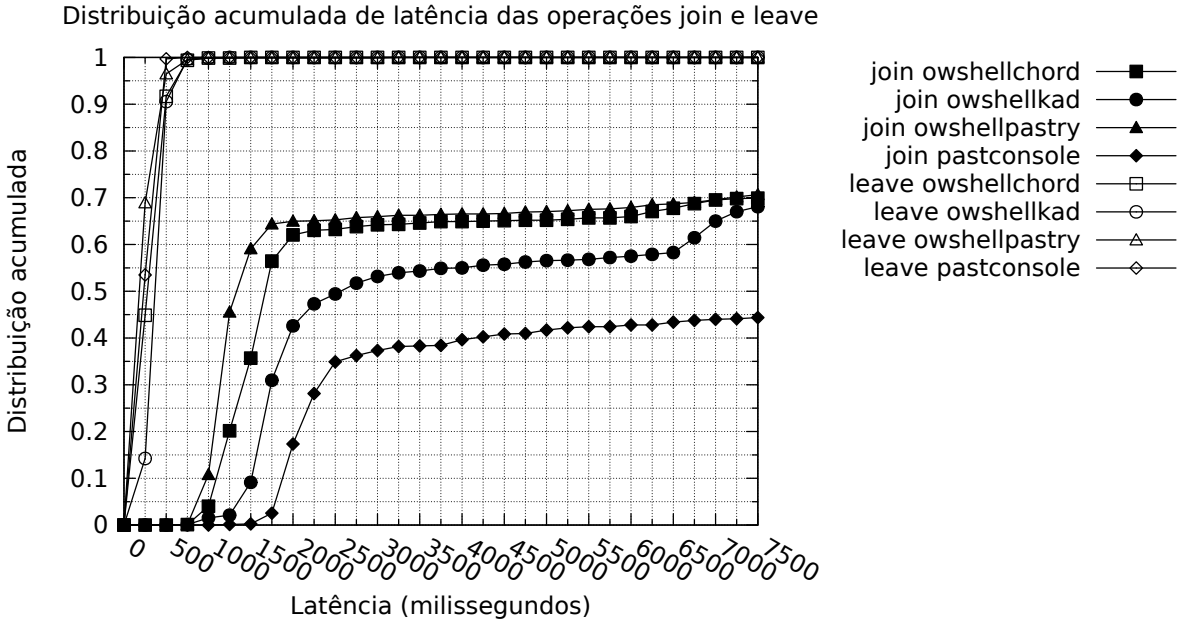

Figura A.1: Latência das operações `join` e `leave` na carga de trabalho `filesharing`, ambiente Dinf

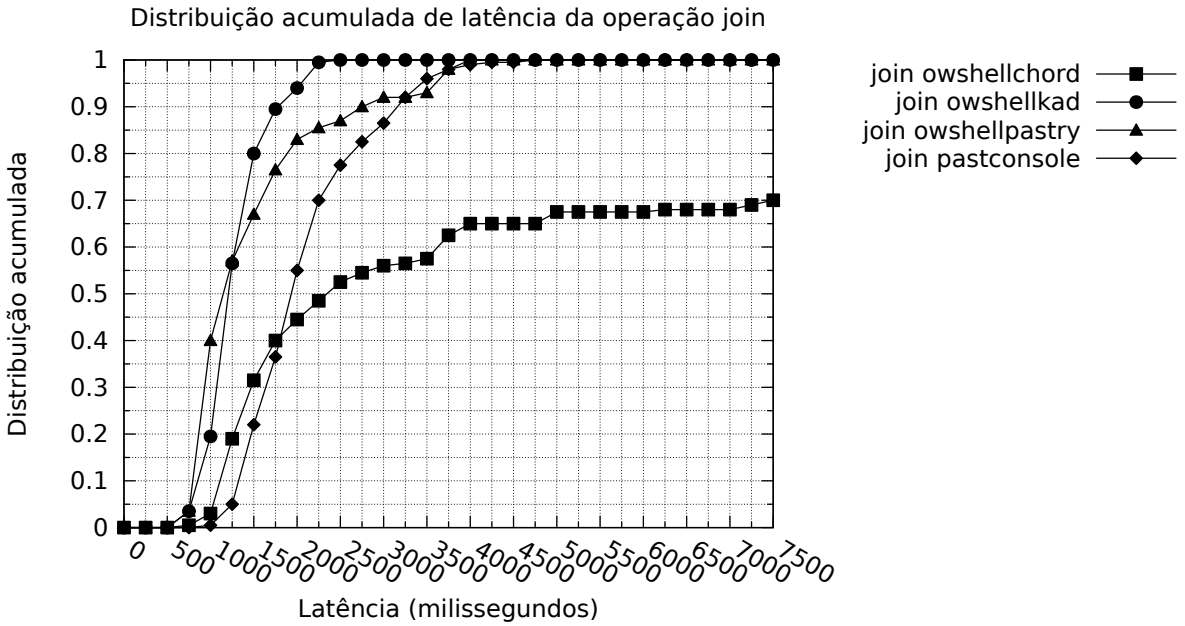

Figura A.2: Latência da operação `join` na carga de trabalho `no-churn`, ambiente Dinf

Figura A.3: Latência das operações `join` e `leave` na carga de trabalho `high-churn`, ambiente Dinf

Figura A.4: Latência das operações `join` e `leave` na carga de trabalho `successrate`, ambiente Dinf

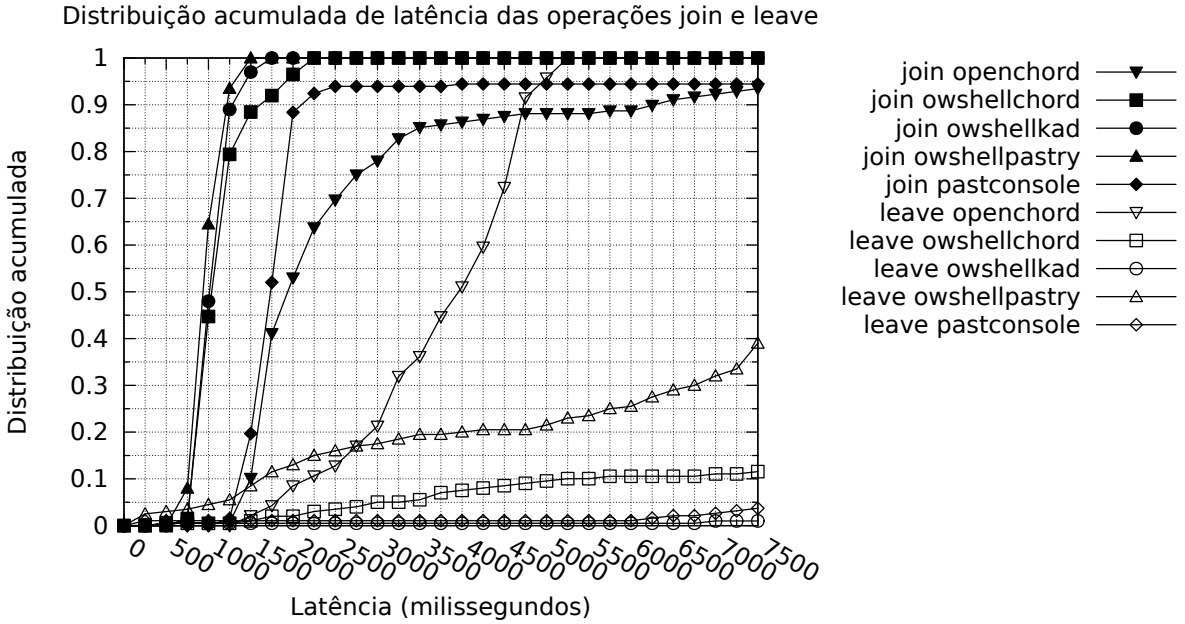

Figura A.5: Latência das operações `join` e `leave` na carga de trabalho `key-spreading`, ambiente Dinf

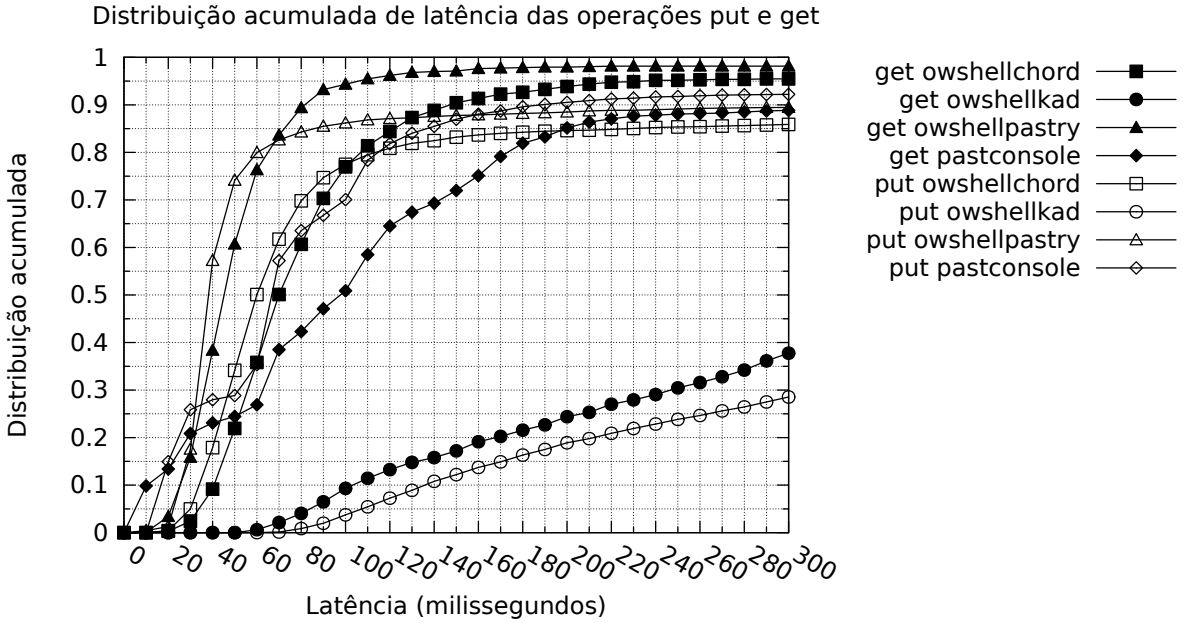

Figura A.6: Latência das operações `put` e `get` na carga de trabalho `filesharing`, ambiente Dinf

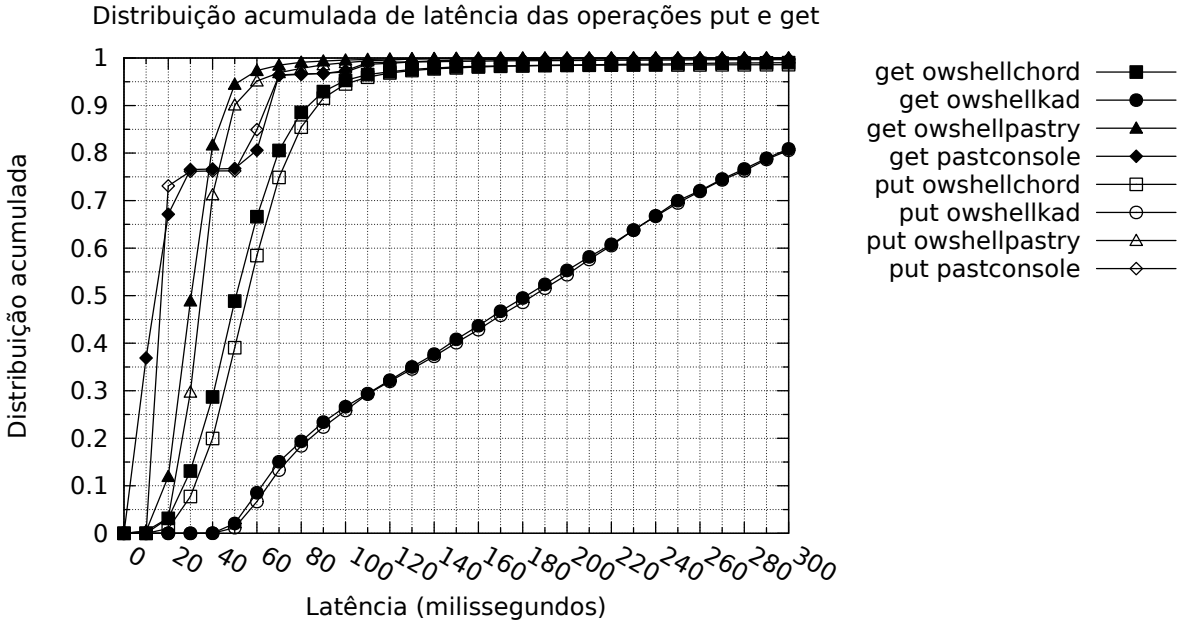

Figura A.7: Latência das operações `put` e `get` na carga de trabalho `no-churn`, ambiente Dinf

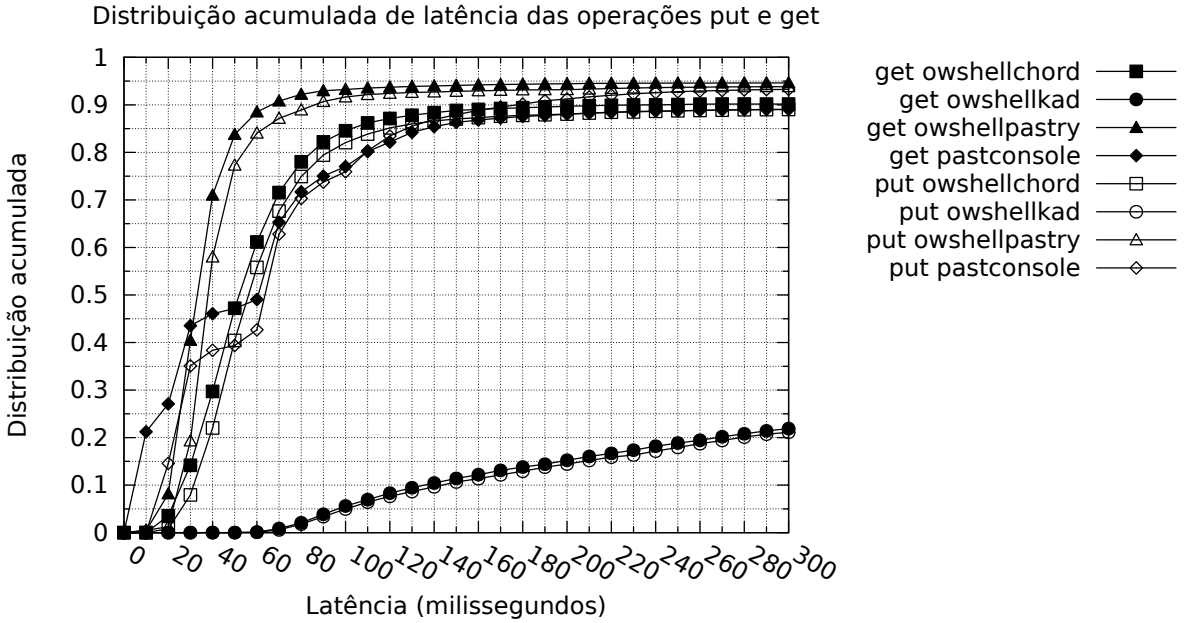

Figura A.8: Latência das operações `put` e `get` na carga de trabalho `high-churn`, ambiente Dinf

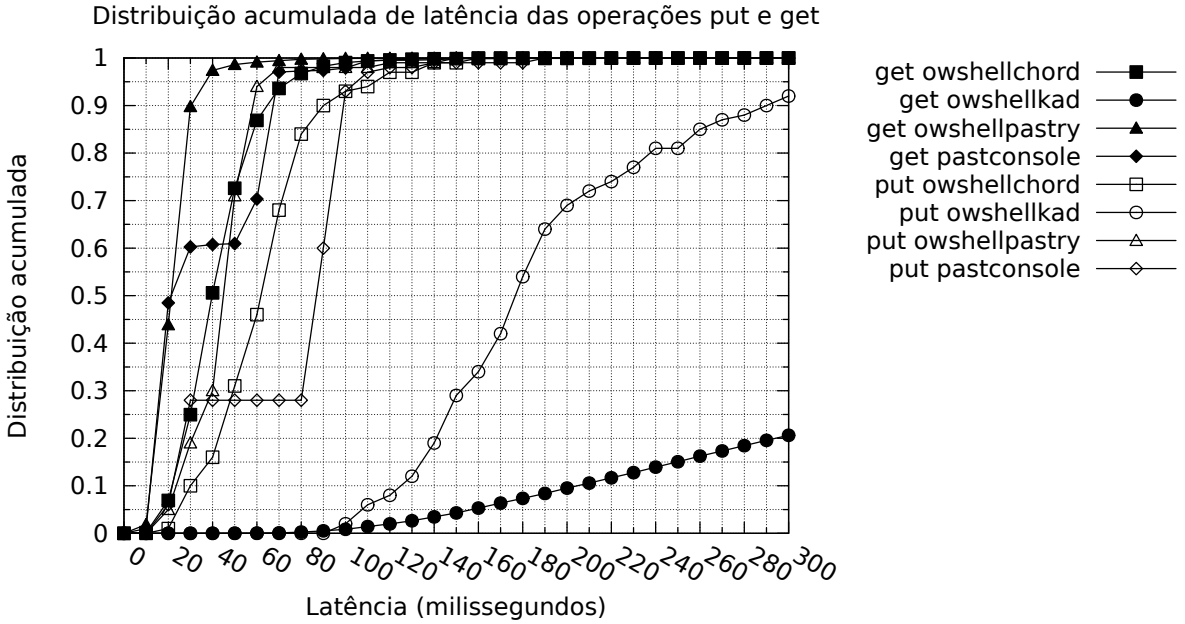

Figura A.9: Latência das operações `put` e `get` na carga de trabalho `successrate`, ambiente Dinf

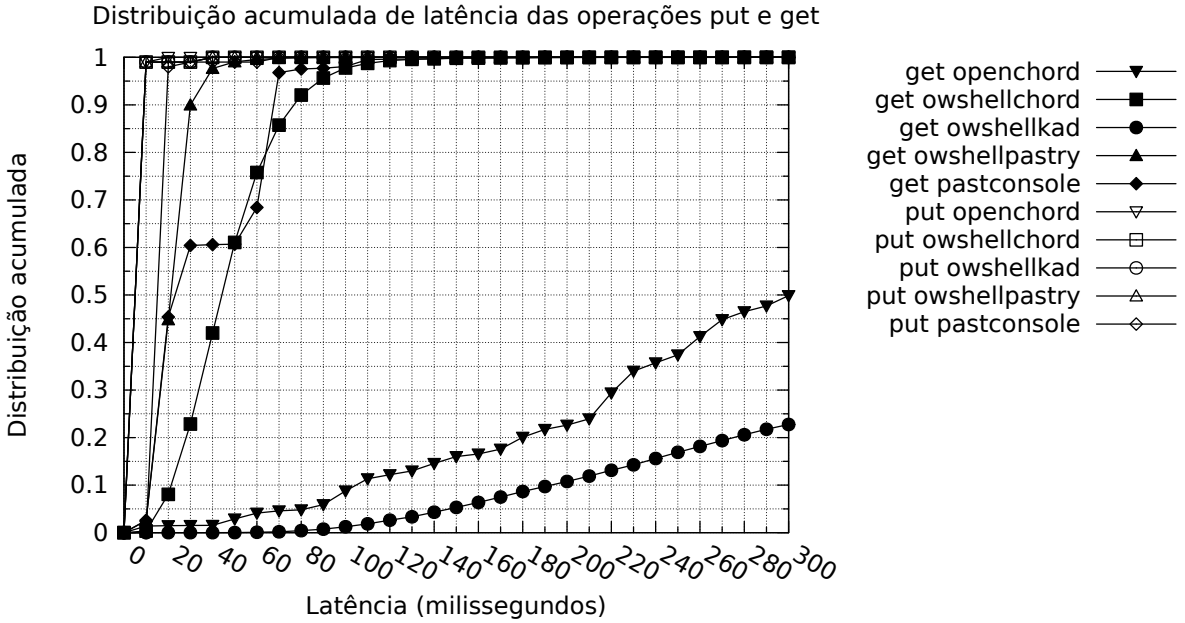

Figura A.10: Latência das operações `put` e `get` na carga de trabalho `key-spreading`, ambiente Dinf

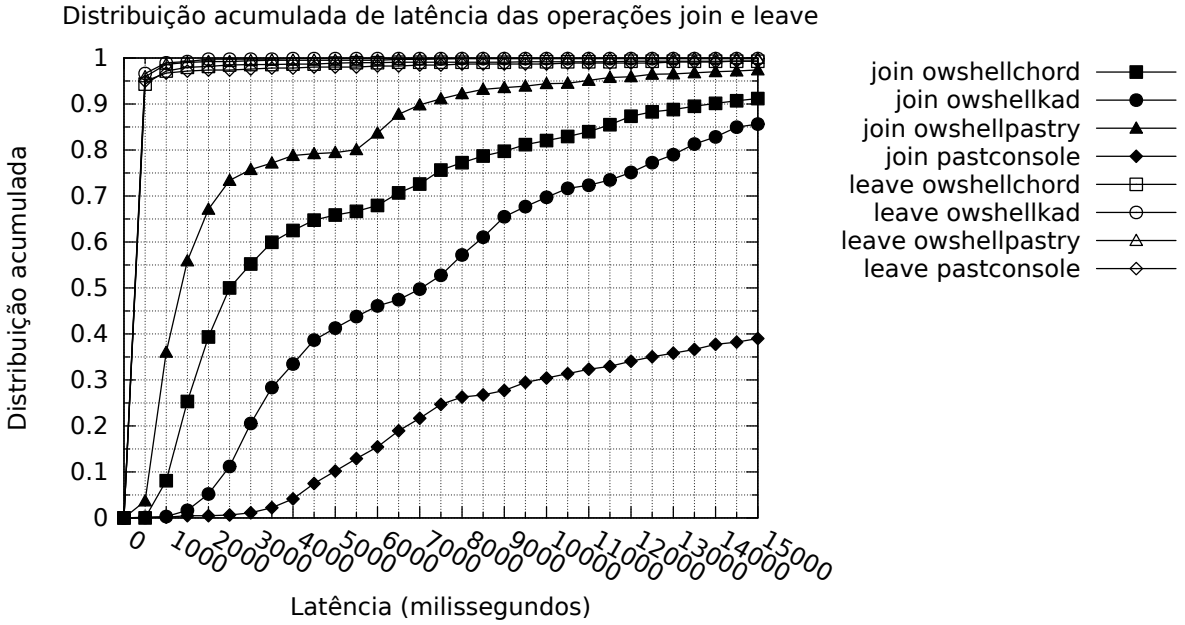

Figura A.11: Latência das operações `join` e `leave` na carga de trabalho `filesharing`, ambiente PlanetLab

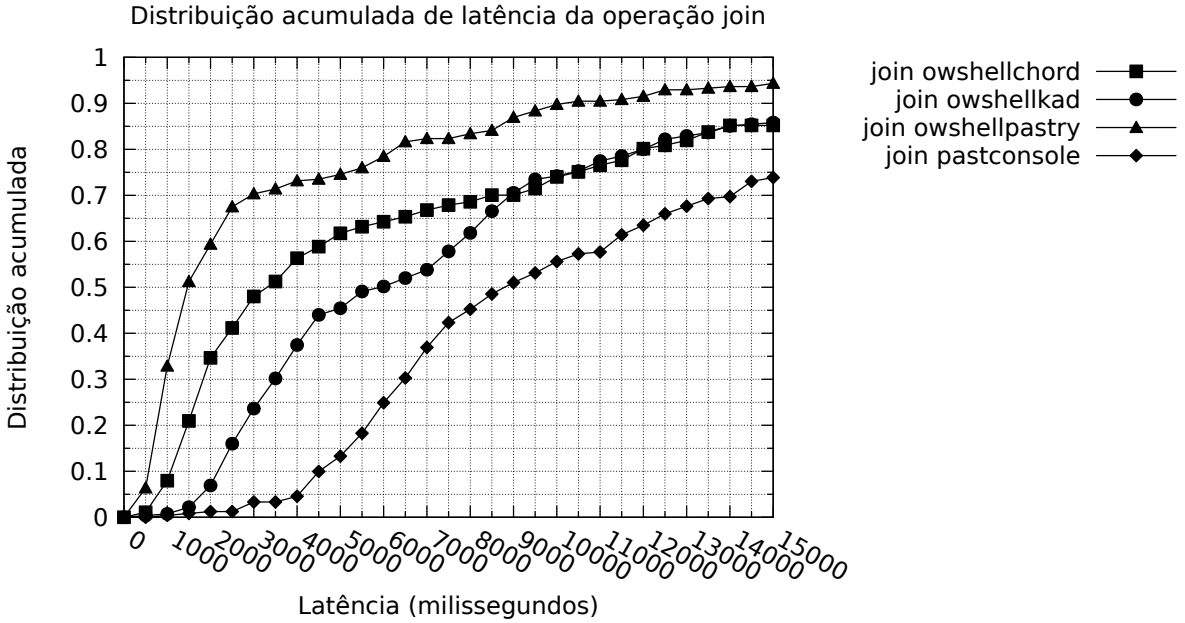

Figura A.12: Latência da operação `join` na carga de trabalho `no-churn`, ambiente PlanetLab

Figura A.13: Latência das operações `join` e `leave` na carga de trabalho `high-churn`, ambiente PlanetLab

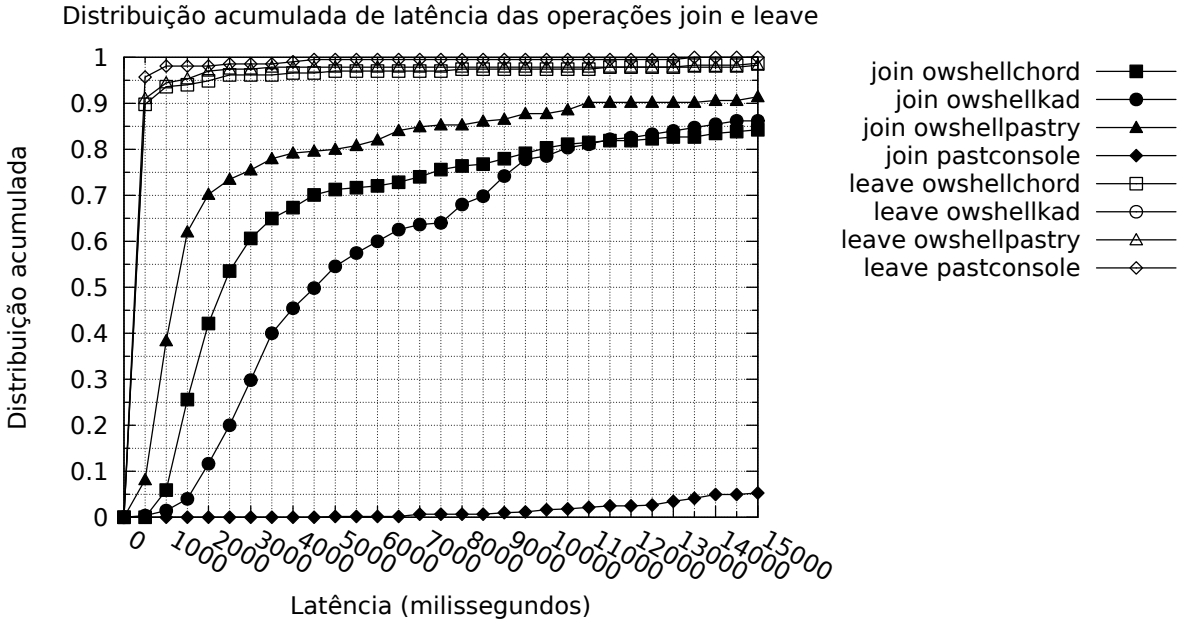

Figura A.14: Latência das operações `join` e `leave` na carga de trabalho `successrate`, ambiente PlanetLab

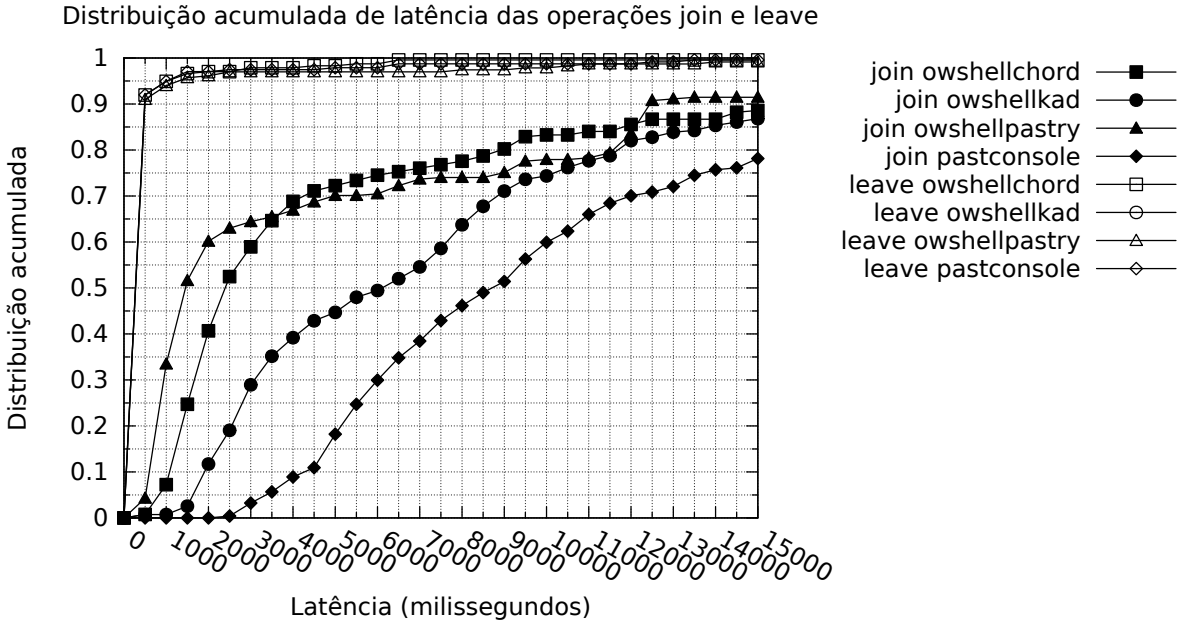

Figura A.15: Latência das operações `join` e `leave` na carga de trabalho `key-spreading`, ambiente PlanetLab

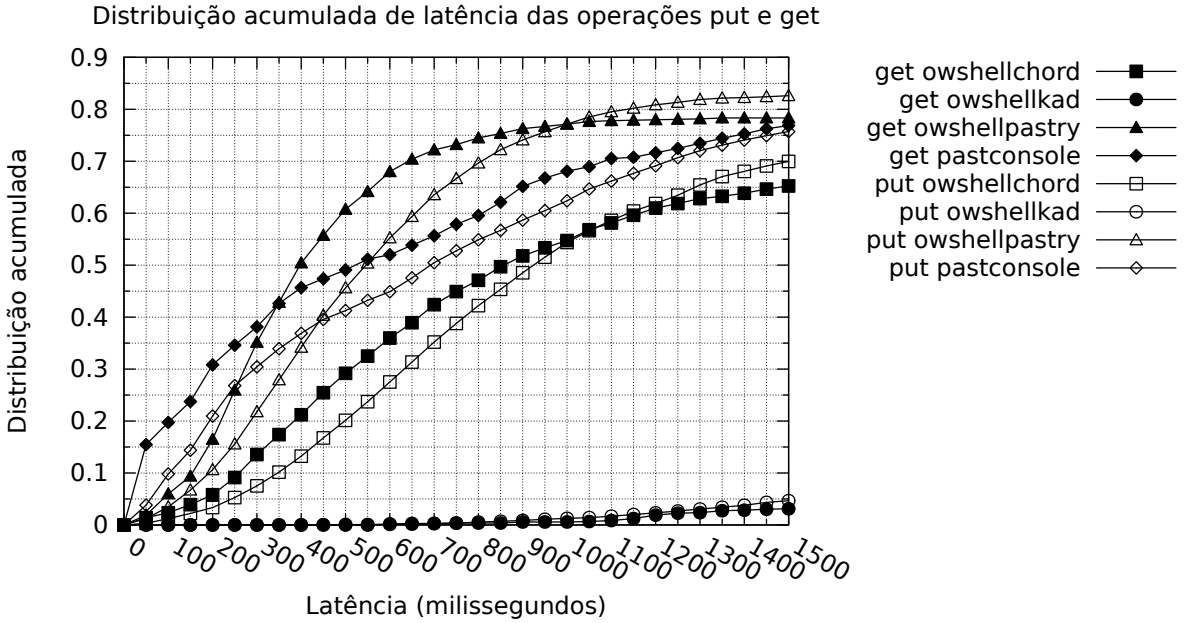

Figura A.16: Latência das operações `put` e `get` na carga de trabalho `filesharing`, ambiente PlanetLab

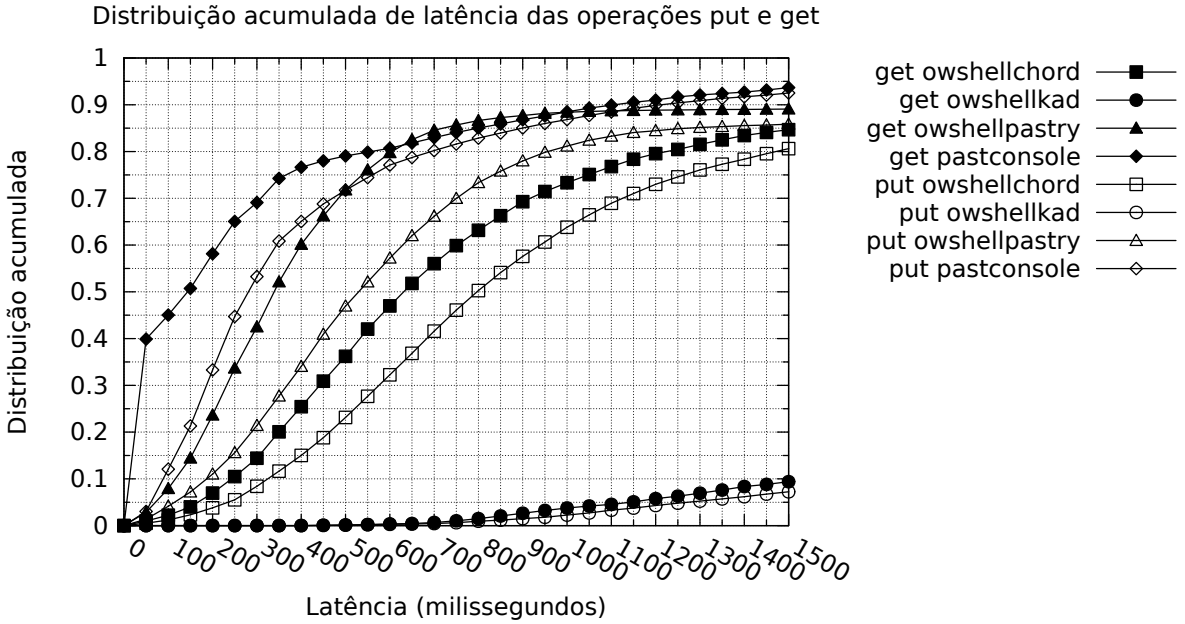

Figura A.17: Latência das operações `put` e `get` na carga de trabalho `no-churn`, ambiente PlanetLab

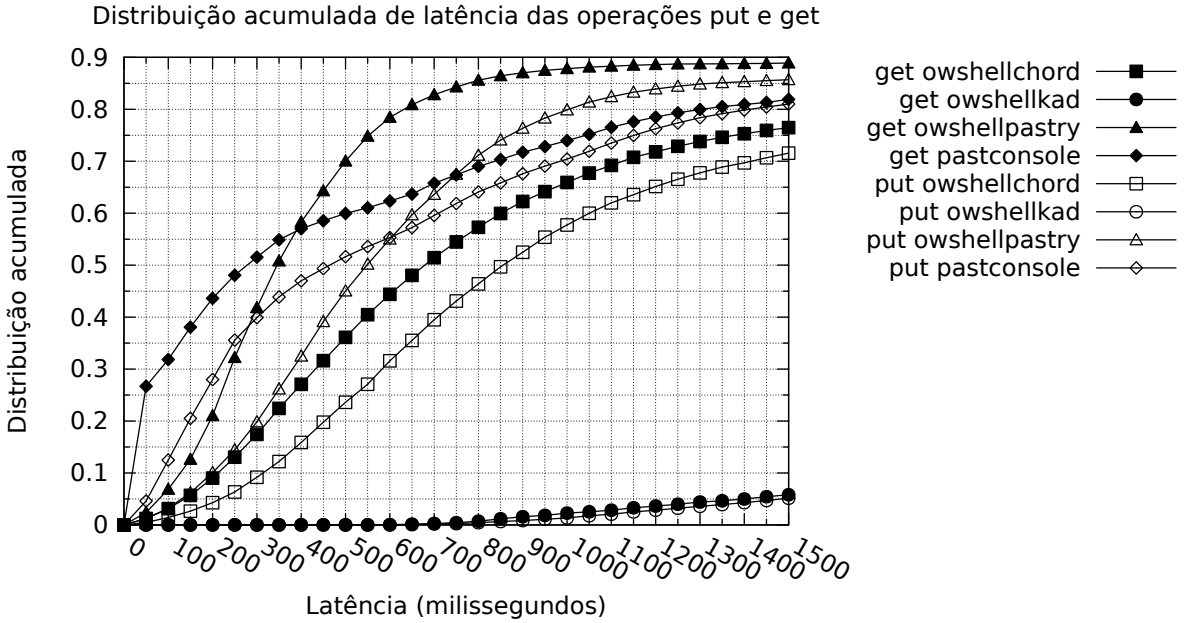

Figura A.18: Latência das operações `put` e `get` na carga de trabalho `high-churn`, ambiente PlanetLab

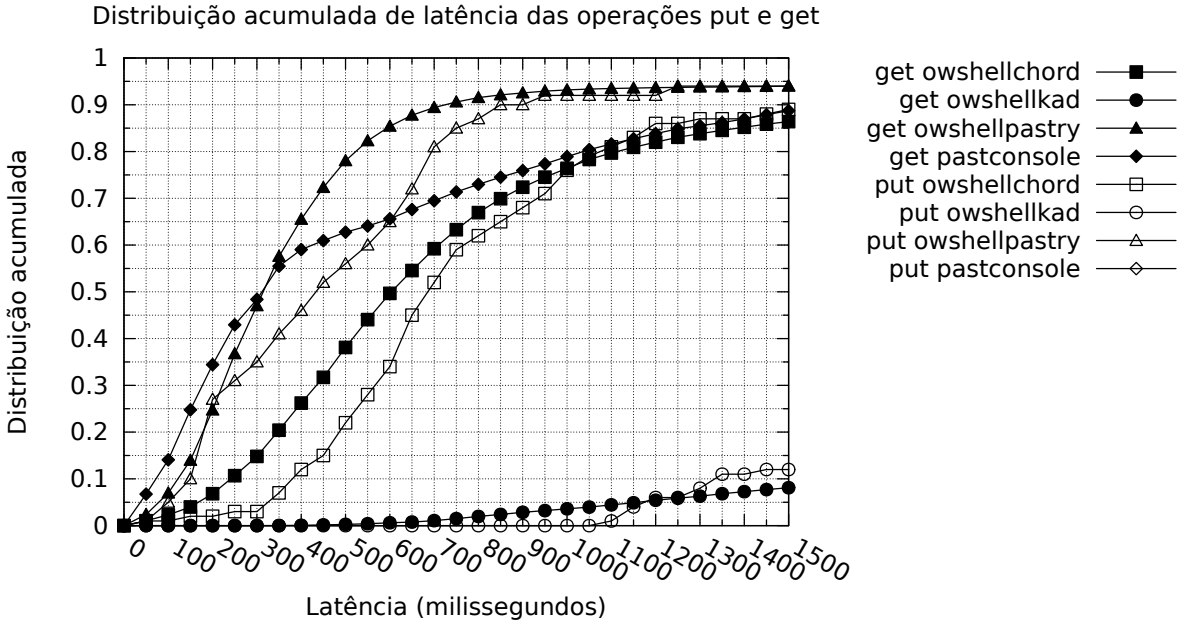

Figura A.19: Latência das operações `put` e `get` na carga de trabalho `successrate`, ambiente PlanetLab

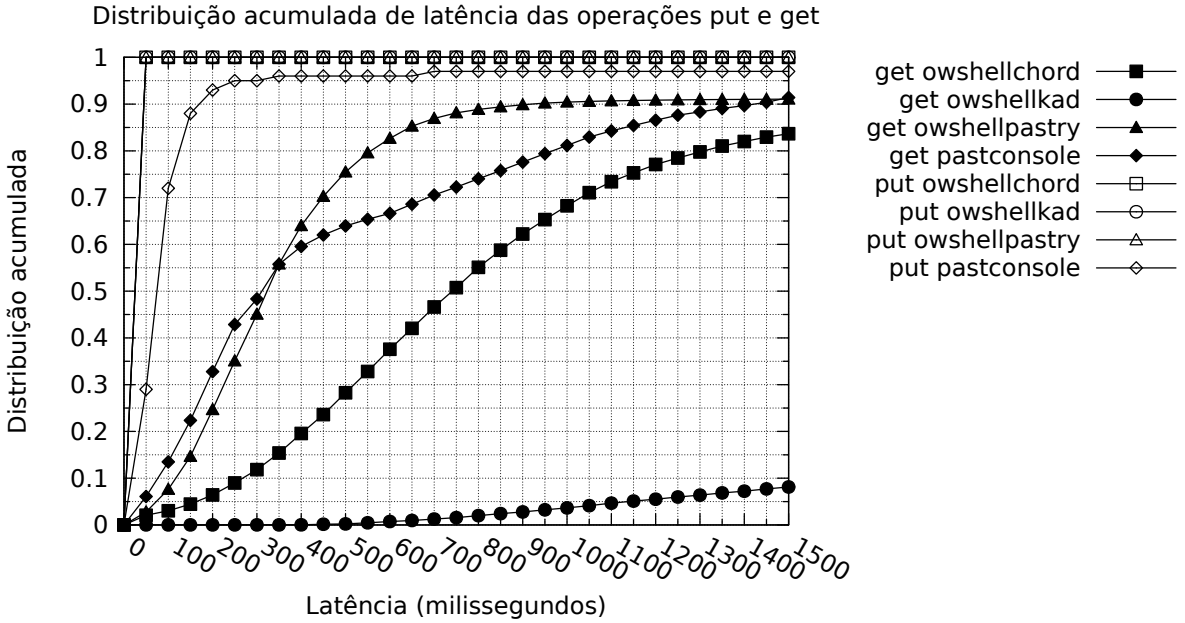

Figura A.20: Latência das operações `put` e `get` na carga de trabalho `key-spreading`, ambiente PlanetLab

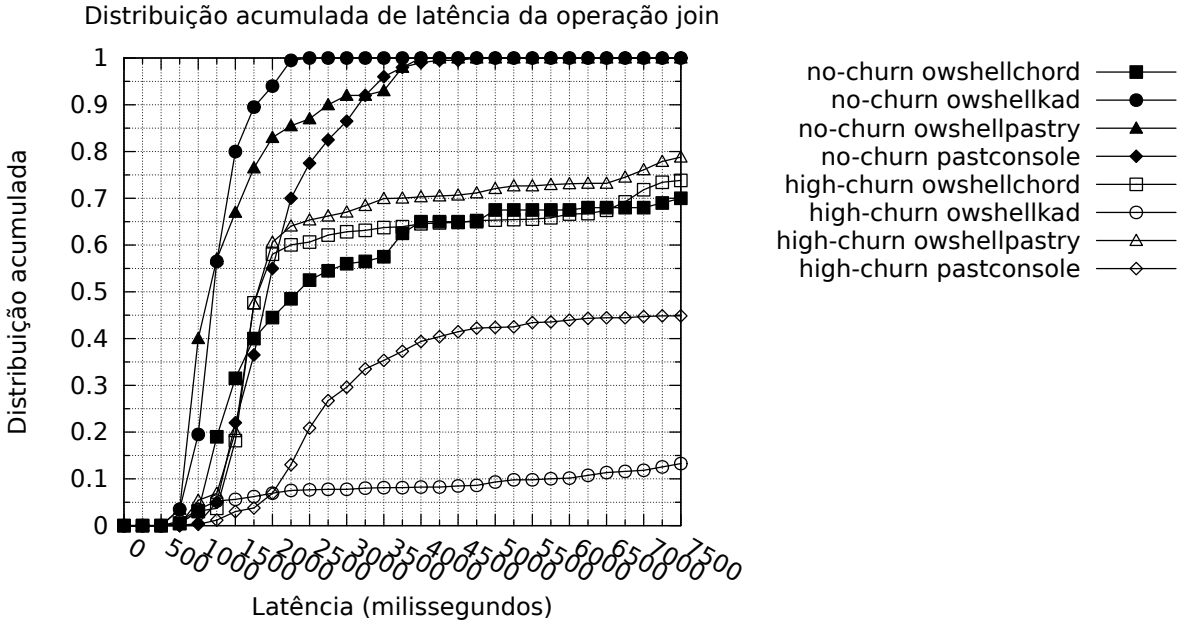

Figura A.21: Latência das operações `join` nas cargas de trabalho `no-churn` e `high-churn`, ambiente Dinf

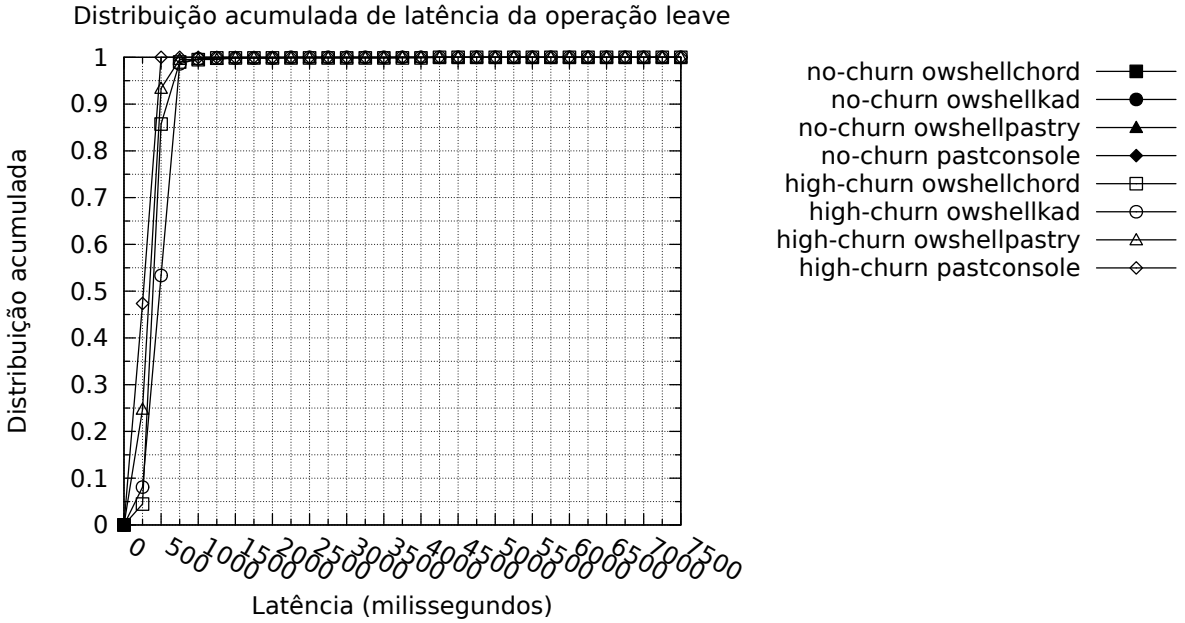

Figura A.22: Latência das operações `leave` nas cargas de trabalho `no-churn` e `high-churn`, ambiente Dinf

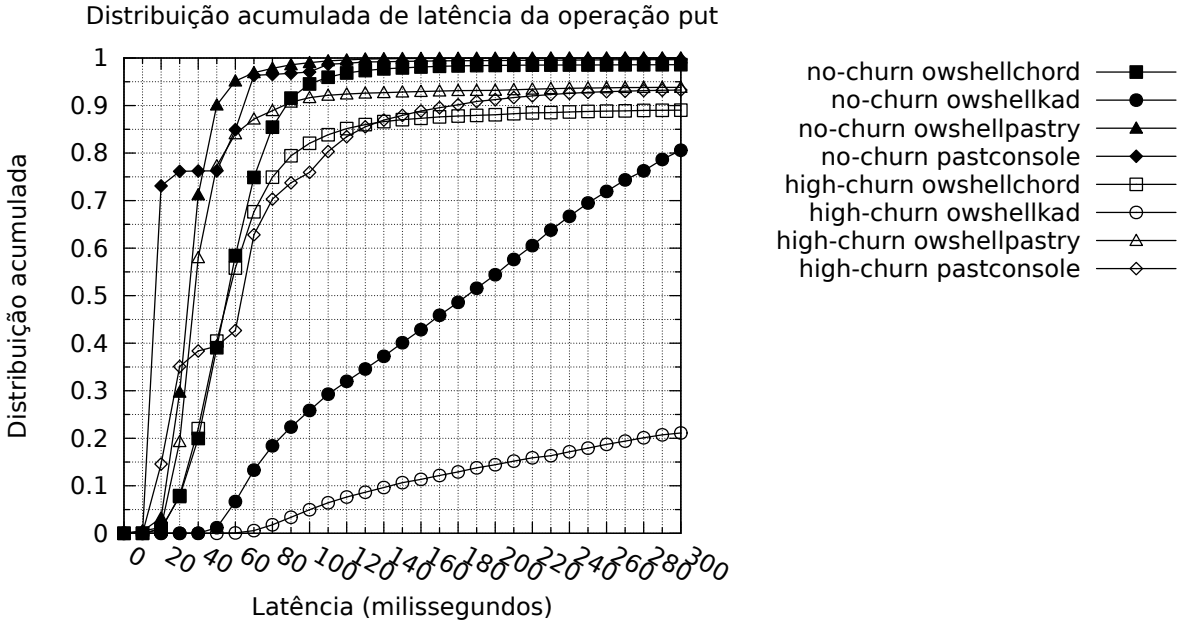

Figura A.23: Latência das operações `put` nas cargas de trabalho `no-churn` e `high-churn`, ambiente Dinf

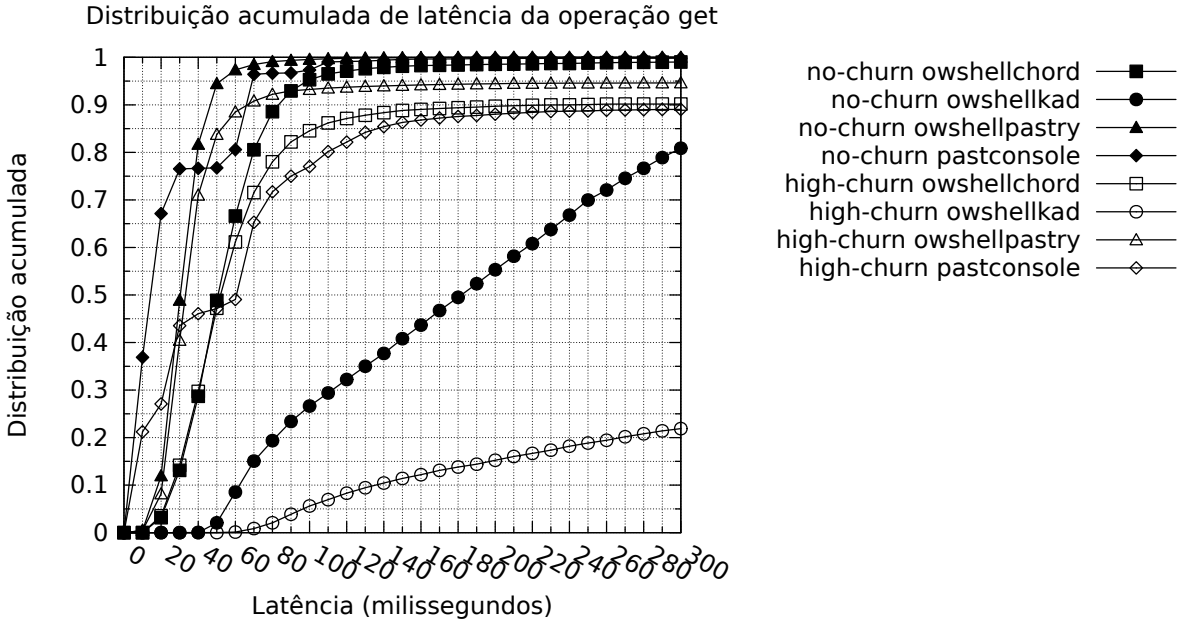

Figura A.24: Latência das operações `get` nas cargas de trabalho `no-churn` e `high-churn`, ambiente Dinf

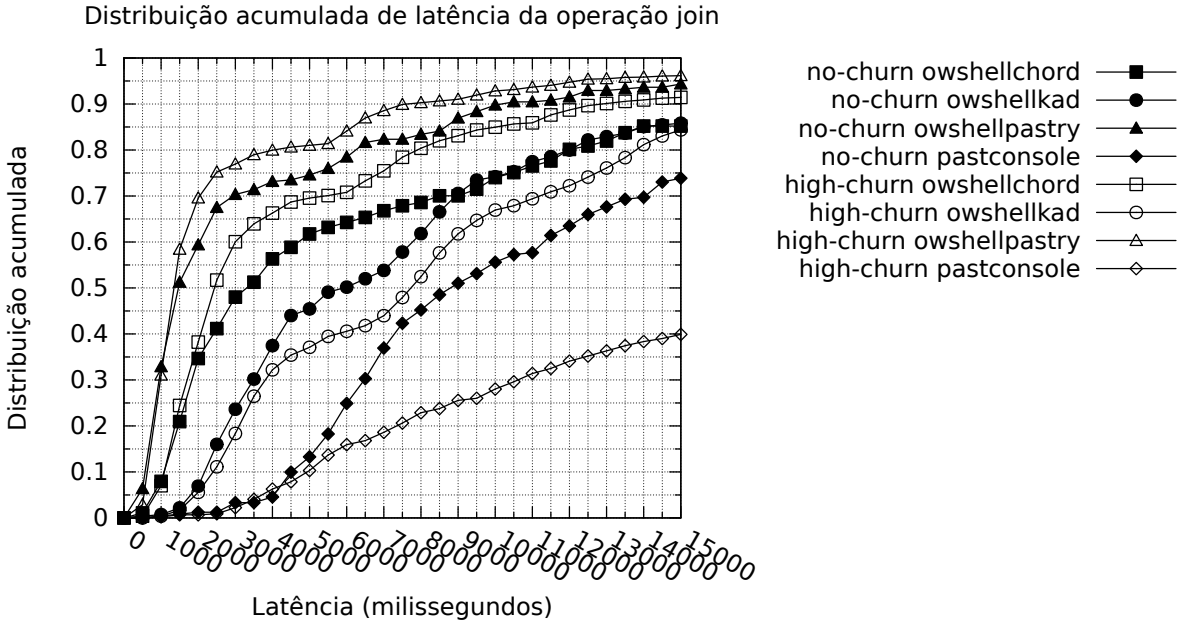

Figura A.25: Latência das operações `join` nas cargas de trabalho `no-churn` e `high-churn`, ambiente PlanetLab

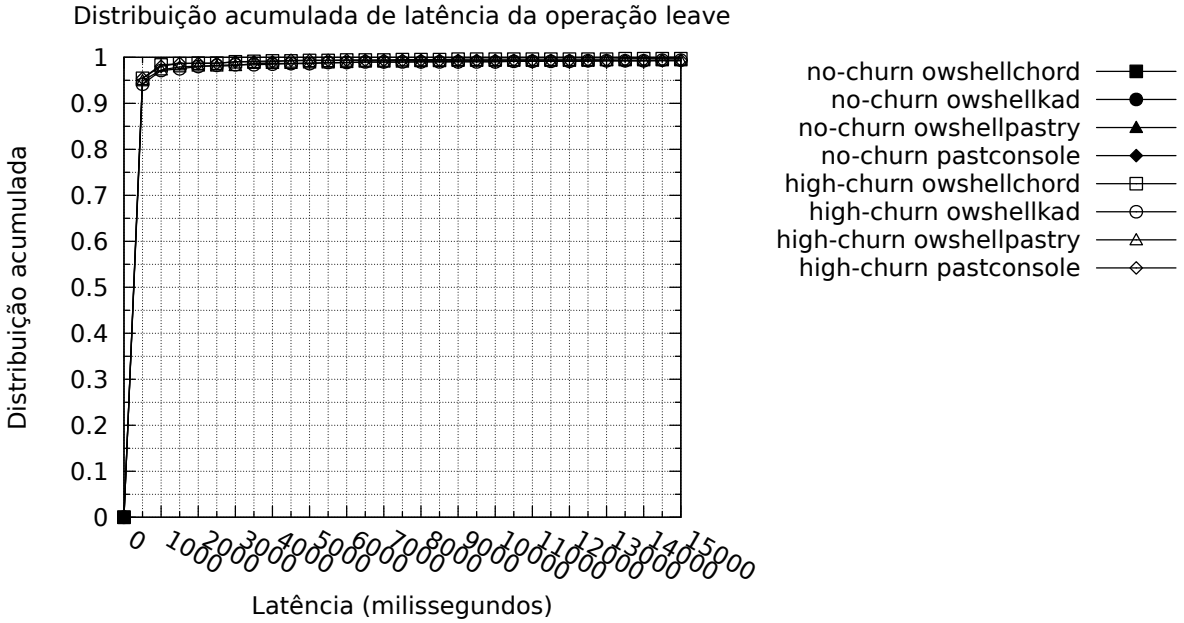

Figura A.26: Latência das operações `leave` nas cargas de trabalho `no-churn` e `high-churn`, ambiente PlanetLab

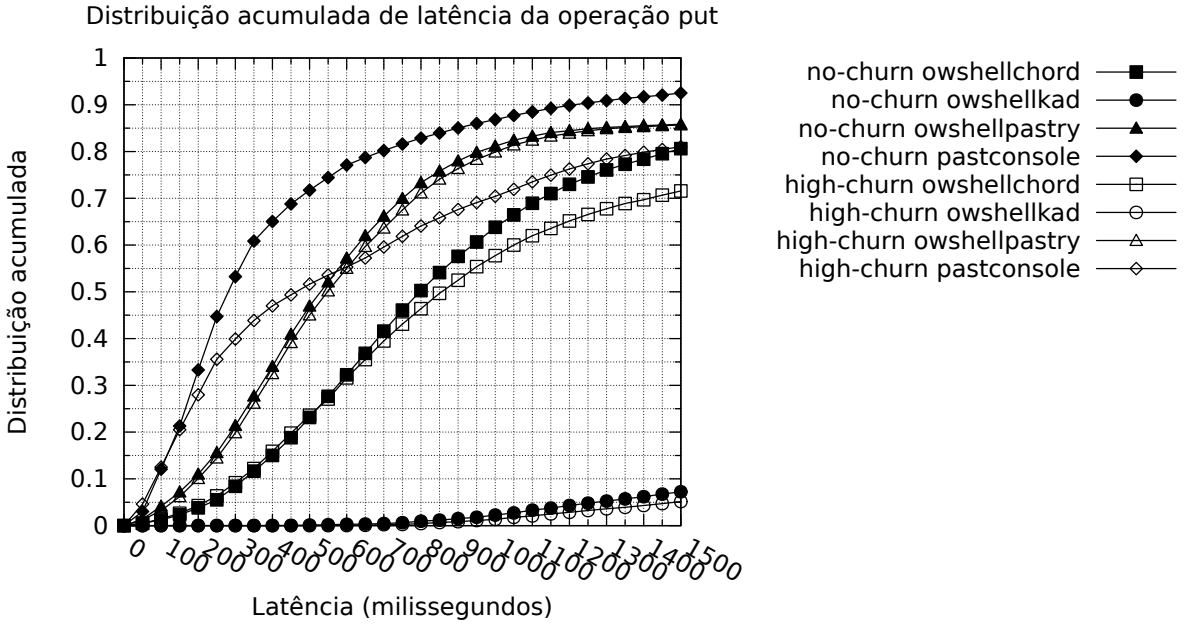

Figura A.27: Latência das operações `put` nas cargas de trabalho `no-churn` e `high-churn`, ambiente PlanetLab

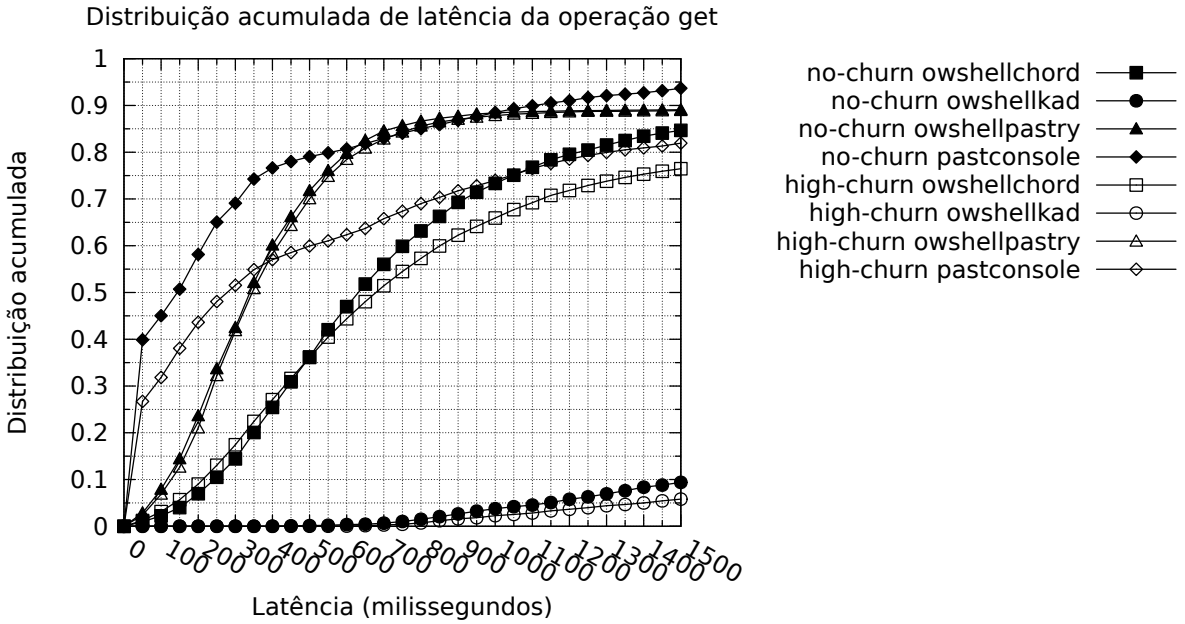

Figura A.28: Latência das operações `get` nas cargas de trabalho `no-churn` e `high-churn`, ambiente PlanetLab

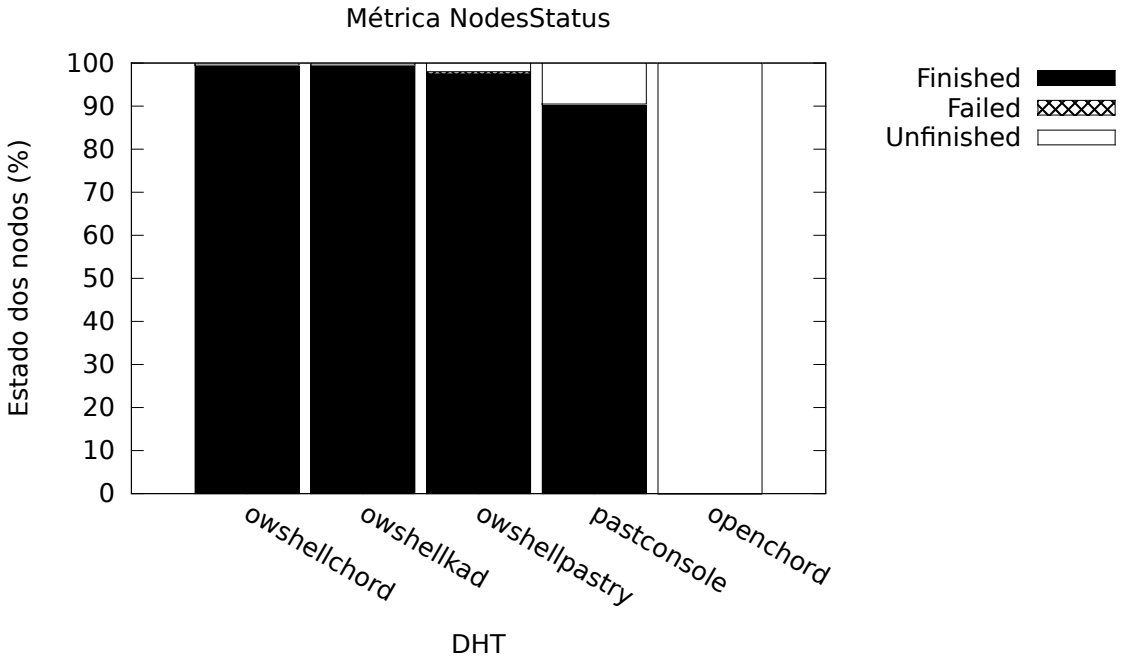

Figura A.29: Métrica `NodesStatus` para a carga de trabalho `filesharing`, ambiente Dinf

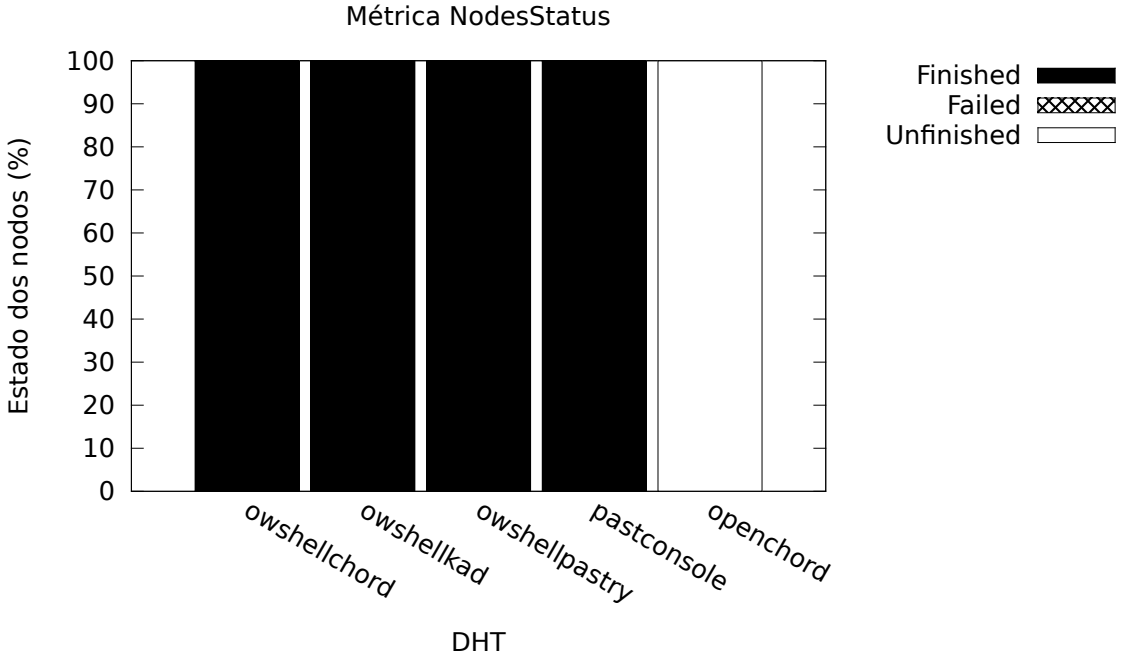

Figura A.30: Métrica `NodesStatus` para a carga de trabalho `no-churn`, ambiente Dinf

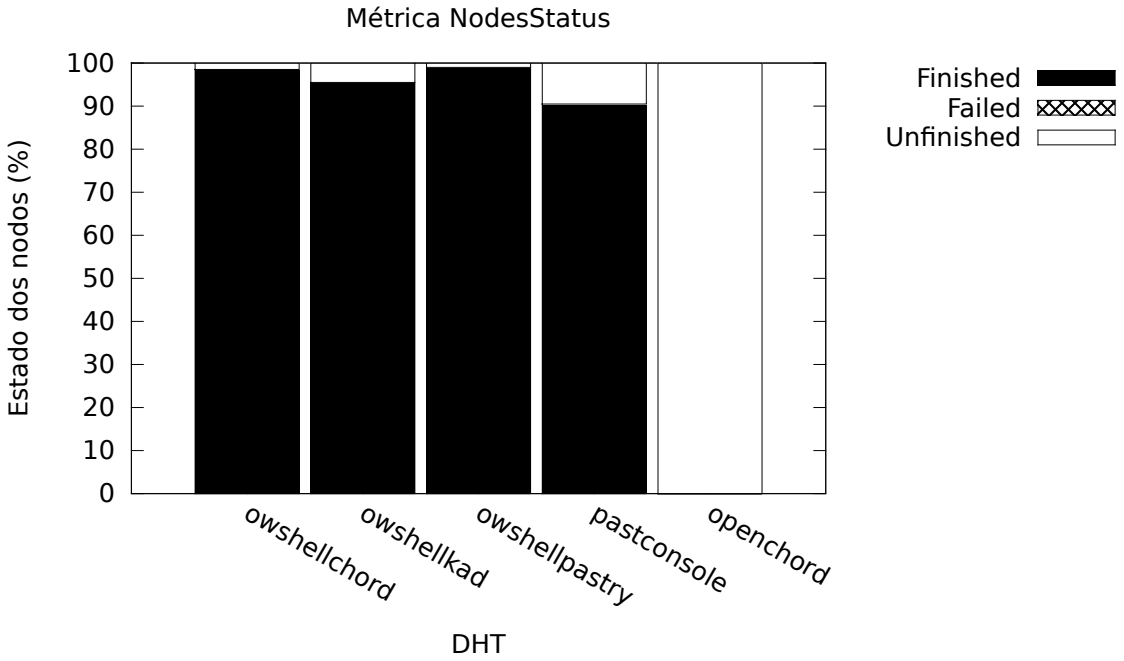

Figura A.31: Métrica `NodesStatus` para a carga de trabalho `high-churn`, ambiente Dinf

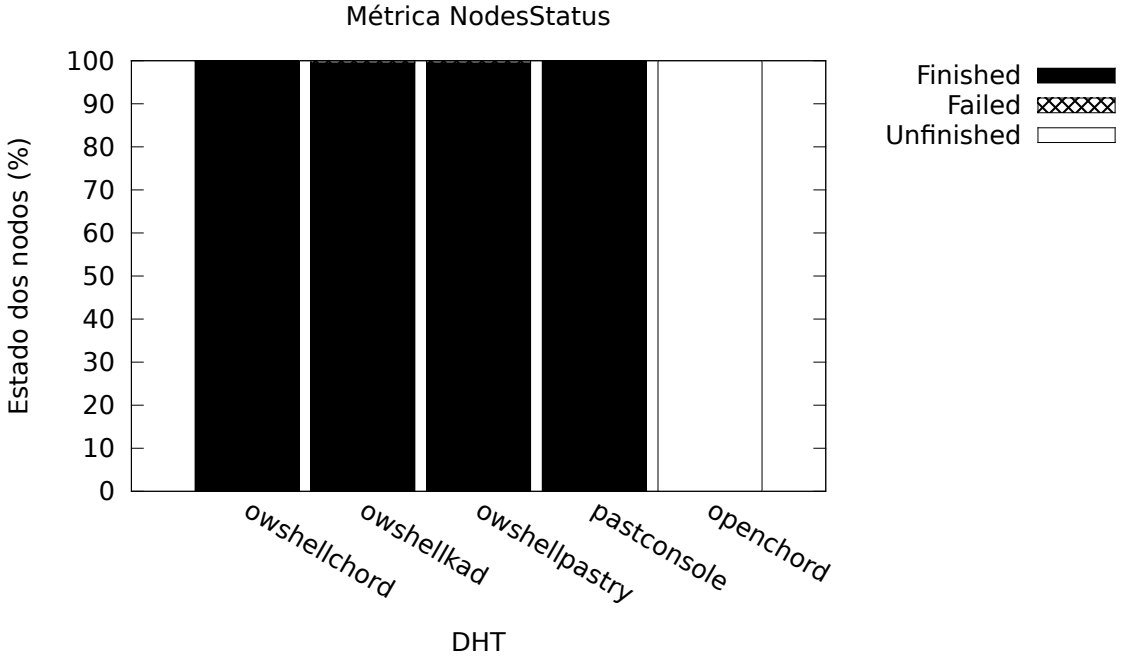

Figura A.32: Métrica `NodesStatus` para a carga de trabalho `successrate`, ambiente Dinf

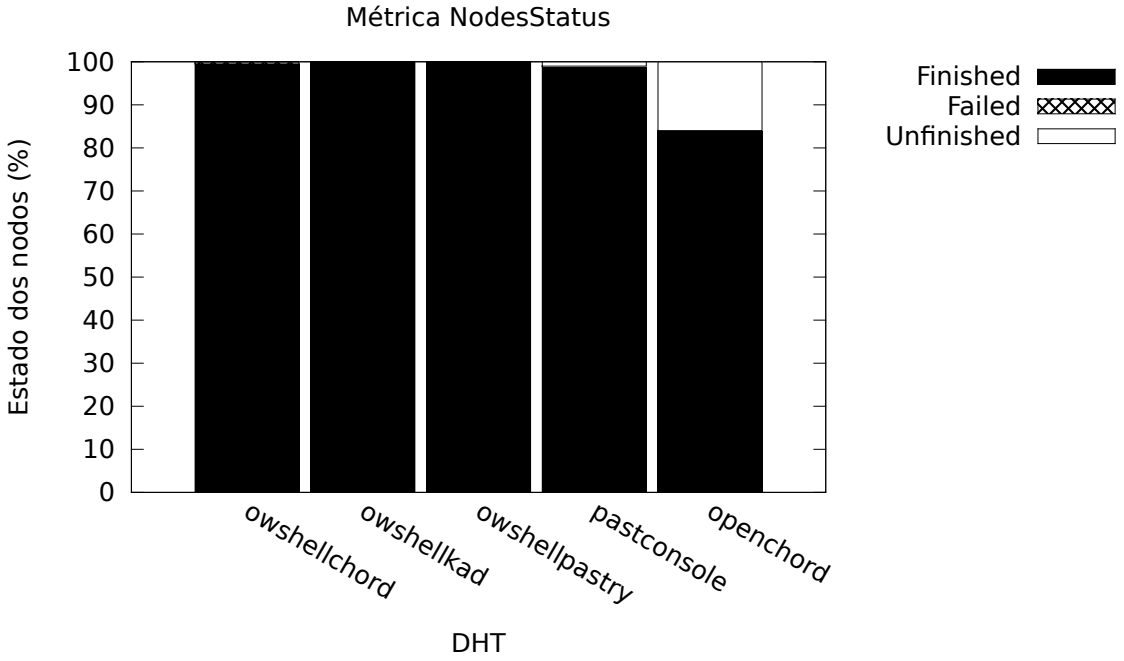

Figura A.33: Métrica `NodesStatus` para a carga de trabalho `key-spreading`, ambiente Dinf

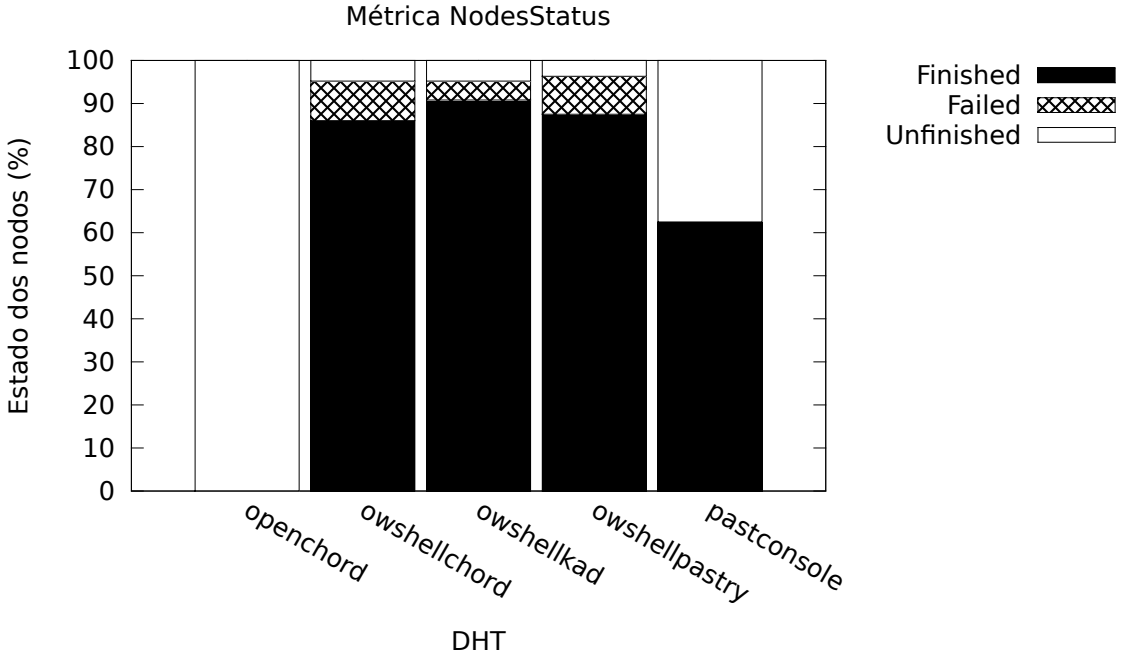

Figura A.34: Métrica `NodesStatus` para a carga de trabalho `filesharing`, ambiente PlanetLab

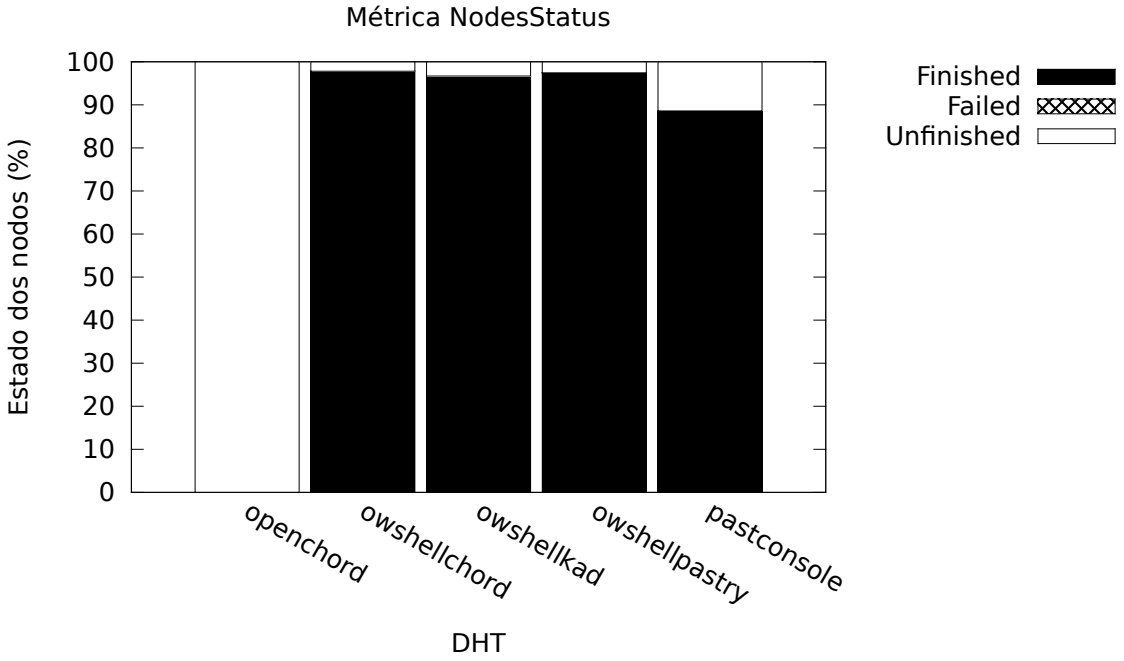

Figura A.35: Métrica `NodesStatus` para a carga de trabalho `no-churn`, ambiente PlanetLab

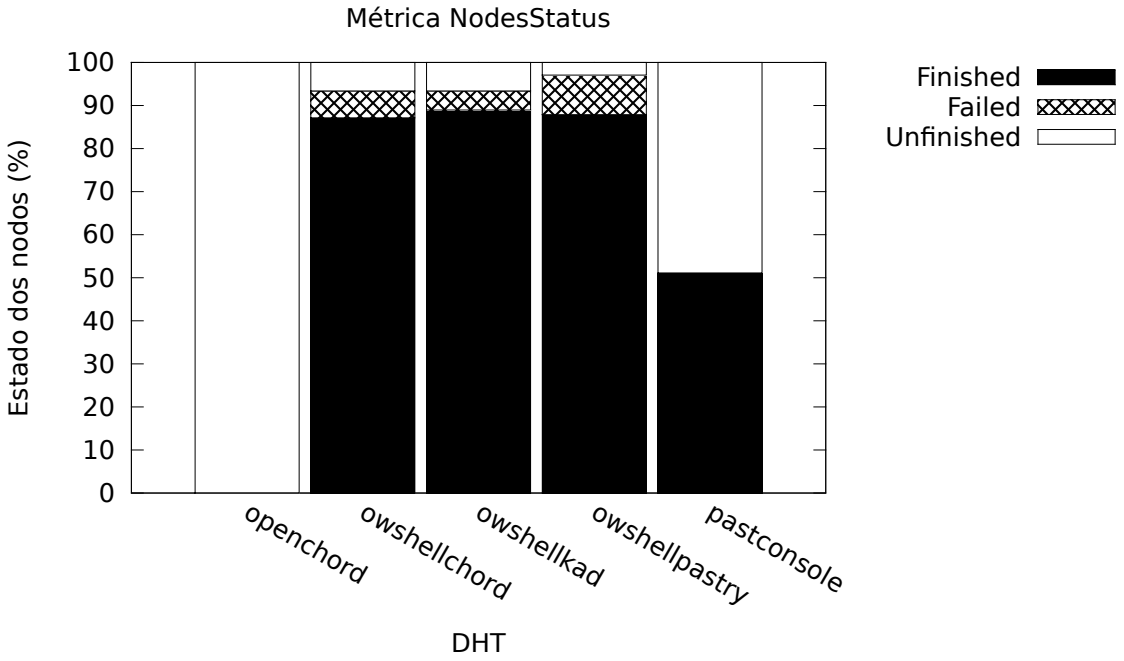

Figura A.36: Métrica `NodesStatus` para a carga de trabalho `high-churn`, ambiente PlanetLab

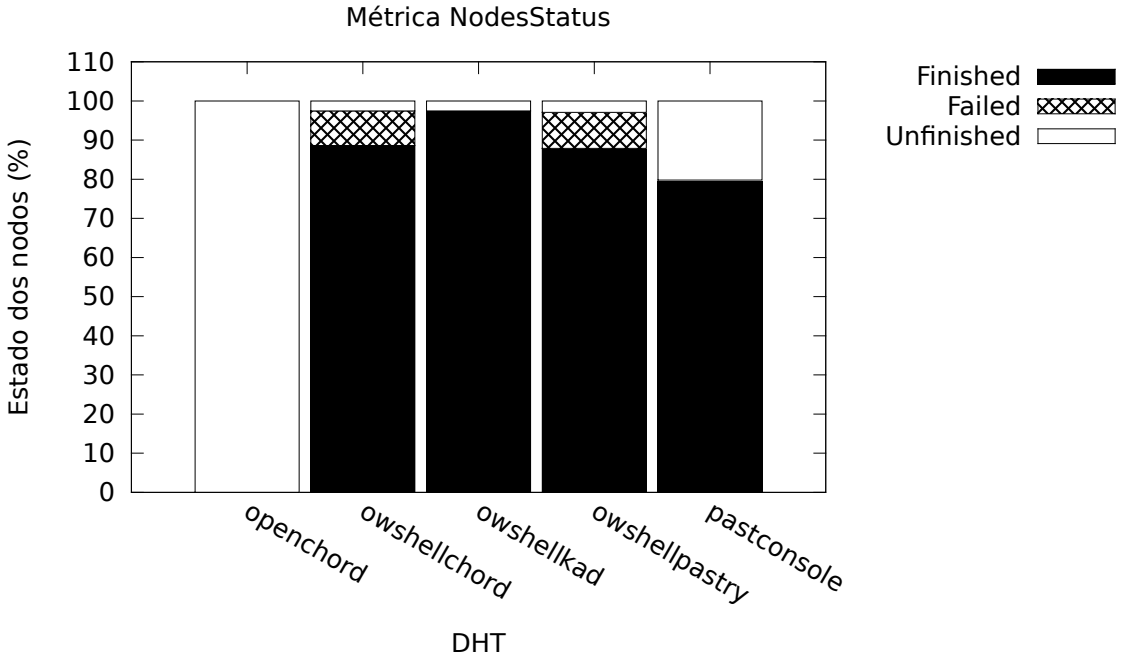

Figura A.37: Métrica `NodesStatus` para a carga de trabalho `successrate`, ambiente PlanetLab

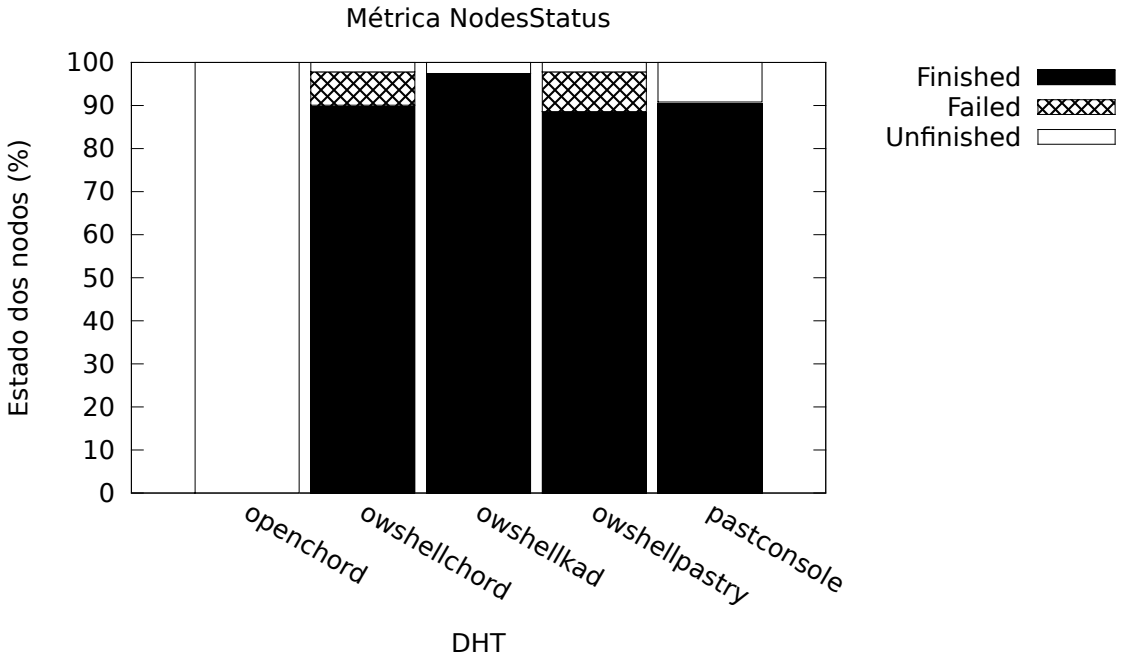

Figura A.38: Métrica `NodesStatus` para a carga de trabalho `key-spreading`, ambiente PlanetLab