Universidade Federal do Rio de Janeiro

TABELAS HASH DISTRIBUÍDAS COM CONSULTAS DE UM SALTO COM BAIXA CARGA DE MANUTENÇÃO

Luiz Rodolpho Rocha Monnerat

2010

TABELAS HASH DISTRIBUÍDAS COM CONSULTAS DE UM SALTO COM BAIXA CARGA DE MANUTENÇÃO

Luiz Rodolpho Rocha Monnerat

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação

Orientador: Claudio Luis de Amorim

Rio de Janeiro
Setembro de 2010

TABELAS HASH DISTRIBUÍDAS COM CONSULTAS DE UM SALTO COM BAIXA CARGA DE MANUTENÇÃO

Luiz Rodolpho Rocha Monnerat

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof. Claudio Luis de Amorim, Ph.D.

_____

Prof. Felipe Maia Galvão França, Ph.D.

_____

Prof. Luis Felipe Magalhães de Moraes, Ph.D.

_____

Prof. José Ferreira de Rezende, Dr.

_____

Profa. Noemi de La Rocque Rodriguez, D.Sc.

_____

Prof. Francisco Vilar Brasileiro, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2010

Monnerat, Luiz Rodolpho Rocha

Tabelas Hash Distribuídas com Consultas de um Salto com Baixa Carga de Manutenção/ Luiz Rodolpho Rocha Monnerat. - Rio de Janeiro: UFRJ/COPPE, 2010.

XIII, 136 p.: il.; 29,7 cm

Orientador: Claudio Luis de Amorim.

Tese (Doutorado) - UFRJ / COPPE / Programa de Engenharia de Sistemas e Computação, 2010

Referências Bibliográficas: p. 127-136.

1. Tabelas Hash Distribuídas. I. Amorim, Claudio Luis de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III Título

Em memória de Haroldo Monnerat

Aos meus amados Silvia, Rafael e Bruno

Agradecimentos

Inicialmente eu gostaria de agradecer à Petrobras por todo o apoio que recebi nestas duas últimas décadas, e pelo inestimável suporte ao desenvolvimento desta tese, incluindo a gentil e fundamental disponibilização do ambiente HPC para realização dos nossos experimentos. Este trabalho não teria sido possível sem a ajuda de inúmeros colegas petroleiros, e seria inviável listá-los todos. Dedico um agradecimento especial para Carlos Henrique Albrecht, Diana Gomes, Jeanice Menezes, José Emerson Lima, Monica Vargas, Paulo Fernando Santos, Paulo Roberto Ferreira, Ronaldo Pollak e Valério Dutra, que me *aturam* e ajudam há mais de 20 anos. Sou também profundamente grato a Fernando Faria e Washington Salles, que me proporcionaram a oportunidade deste doutorado, a Leandro Tavares Carneiro pelo suporte com os multicomputadores, a Guilherme Vilela pela ajuda com as configurações de rede, a Bernardo Fortunato pelo apoio com FUSE, e a Paulo Souza pela valiosíssima ajuda com as aplicações sísmicas paralelas.

A toda a UFRJ/COPPE por tudo que aprendi nestes últimos anos. Gostaria de agradecer especialmente à equipe do LCP, em particular a Diego Dutra, Lauro Whately, Leonardo Pinho, Luiz Maltar Castello Branco e Renato Dutra. Sou muito grato também às nossas secretárias Claudia Helena Prata, J. Solange Santos, Mercedes Barreto e Sonia Galliano, bem como ao suporte de Roberto Rodrigues. Diversos professores me ajudaram e me ensinaram muito, em particular Cristiana Bentes, Edil Pinheiro, Felipe França, Inês Dutra, Mário Benevides, Ricardo Farias, Valmir Barbosa, Vitor Costa e, muito especialmente, Ricardo Bianchini, ao qual serei sempre grato. Agradeço ainda a todos os ilustres membros da banca examinadora, por seus comentários e sugestões construtivas, bem como ao apoio provido pelo CNPq e pela FINEP.

Agradeço muito a Claudio Amorim, não só pela sua orientação visionária, como também pela paciência, dedicação e apoio. Trabalhar ao seu lado foi uma experiência altamente produtiva, prazerosa e gratificante.

Aos meus pais, Haroldo e Diva, que com muito amor sempre se esforçaram para me proporcionar a melhor educação.

Aos maravilhosos Bruno, Rafael e Silvia, por seu apoio, paciência, amor e carinho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

TABELAS HASH DISTRIBUÍDAS COM CONSULTAS DE UM SALTO COM BAIXA CARGA DE MANUTENÇÃO

Luiz Rodolpho Rocha Monnerat

Setembro/2010

Orientador: Claudio Luis de Amorim

Programa: Engenharia de Sistemas e Computação

Nesta tese estamos introduzindo uma inovadora Tabela *Hash* Distribuída (DHT), batizada de D1HT, que é capaz de resolver consultas com um salto e tem baixas demandas de rede, mesmo em ambientes com dinâmicas representativas de aplicações distribuídas populares. Para validar suas propriedades, introduzimos dois teoremas, implementamos o sistema proposto, e desenvolvemos a mais ampla avaliação experimental de DHTs já publicada, com até 4.000 pares e 2.000 nós físicos. Nosso conjunto de resultados nos permitiu provar teoricamente, experimentalmente e analiticamente as propriedades de correção, balanceamento de carga, custos e desempenho de D1HT, bem como sua superioridade frente a todas as outras DHTs de baixa latência, com reduções de custos de até uma ordem de magnitude para sistemas com dimensões e comportamentos representativos de aplicações reais e populares. De maneira a efetivamente validar a utilidade de nossa principal contribuição, introduzimos e desenvolvemos um sistema de arquivos distribuído fundamentado em D1HT, e conduzimos experimentos mostrando que este sistema pode prover desempenho até oito vezes superior ao alcançado por uma solução comercial de última geração. Deste modo, o nosso conjunto de resultados formais, analíticos e experimentais, nos permitiu concluir que, frente a contextos atuais e tendências futuras, D1HT já desponta como uma ferramenta útil e adequada para diversos ambientes, desde Centros de Computação de Alto Desempenho até aplicações de larga escala na Internet, devendo se tornar cada vez mais atraente para um espectro continuamente maior de aplicações. Disponibilizamos o código fonte de D1HT, facilitando seu uso e o desenvolvimento de novas pesquisas nesta área.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SINGLE HOP DISTRIBUTED HASH TABLES WITH LOW MAINTENANCE COSTS

Luiz Rodolpho Rocha Monnerat

September/2010

Advisor: Claudio Luis de Amorim

Department: Computing Systems Engineering

In this thesis we are introducing a new distributed hash table (DHT), called D1HT, which is able to solve lookups with one hop and low overhead even in systems with frequent node joins and leaves. In order to certify its properties, we introduced two theorems, implemented D1HT, and realized the largest DHT experimental comparison ever published, with up to 4,000 peers and 2,000 physical nodes in two radically distinct environments. Our set of results allowed us to prove formally, experimentally, and analytically the D1HT characteristics of correctness, load balancing, maintenance overheads and performance, as well as its superior properties in relation to other low latency DHTs, with D1HT's overhead reductions of up to one order of magnitude for systems which size and behavior are representative of popular applications. In order to demonstrate the utility of D1HT, we implemented a distributed file system on top of it, and carried out experiments showing that the resulting system could achieve performance up to eight times superior than that of a modern commercial solution. Our large set of formal, analytical, and experimental results allowed us to conclude that D1HT is already an useful tool for a wide range of computing environments, from High Performance Computing data centers to Internet large scale applications, and that future trends may make D1HT even more attractive for a larger range of applications. We have made the D1HT source code freely available, which may easy its use by other groups as well as to foster new research in this area.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A habilidade de localizar informação eficientemente em sistemas computacionais sempre foi uma questão fundamental da Ciência da Computação, sendo que este problema adquire uma nova dimensão no âmbito de sistemas distribuídos. Esta questão torna-se ainda mais crítica na medida em que estes ambientes distribuídos têm crescido aceleradamente, ao mesmo tempo em que os ganhos em termos de latência de rede não têm acompanhado o ritmo da evolução do desempenho dos processadores e da disponibilidade de banda passante [64]. Enquanto soluções com arquitetura Cliente/Servidor (p.ex., diretórios) têm sido capazes de atender satisfatoriamente a sistemas distribuídos com até alguns milhares de participantes, as suas limitações de escalabilidade as tornam onerosas ou mesmo inadequadas para uso em sistemas vastos, com dezenas ou centenas de milhares de participantes, o que tem motivado intensa atividade de pesquisa em torno de soluções alternativas com uso de arquitetura par-a-par (ou P2P, acrônimo do termo em inglês *peer-to-peer*) [79].

Esta tese considera os ambientes atuais e de um futuro próximo, que formam um horizonte onde as constantes e acentuadas evoluções da dimensão dos sistemas distribuídos, da velocidade dos processadores e da disponibilidade de banda passante de rede, ocorrem em um ritmo muito mais acelerado do que os ganhos em termos de latência de rede.

Motivado por esta realidade, esta tese avança o estado-da-arte do método de tabela hash distribuída (ou DHT, acrônimo do termo em inglês *Distributed Hash Tables*), introduzindo e desenvolvendo uma inovadora DHT que consegue prover baixa latência mesmo em ambientes sujeitos a altas taxas de adesões e partidas de participantes, e que se destaca por ser adequada para diversos ambientes, desde centros de computação de alto desempenho até aplicações de grande escala na Internet.

Nas próximas seções, discutiremos o contexto, a motivação, as principais contribuições desta tese, e então apresentaremos a organização dos seus demais capítulos.

1.1 Contexto

As primeiras soluções P2P desenvolvidas para o problema de localização de informação em sistemas distribuídos foram baseadas em técnicas de inundação (p.ex., Gnutella [77]), as quais têm limitações de escalabilidade e não garantem que a informação procurada será encontrada mesmo que ela exista no sistema, uma vez que cada par pode receber a mesma consulta várias vezes, sem a certeza de que cada consulta irá atingir todos os pares que contenham a informação procurada.

Alternativamente, DHTs provêem uma solução prática e escalável para localização de informação em ambientes amplamente distribuídos, e apresentam características superiores aos diretórios e às técnicas de inundação [4]. Entre as diversas vantagens em relação aos diretórios, deve-se destacar os fatos das DHTs serem mais escaláveis, baratas e fáceis de gerenciar, especialmente se considerarmos que podem ser implementadas segundo uma arquitetura puramente P2P e auto-reorganizável [83]. Em relação às técnicas de inundação, podemos afirmar que as DHTs evitam desperdício de recursos, são mais escaláveis e podem garantir que toda a consulta que tenha solução seja resolvida satisfatoriamente.

Por estas e outras razões, DHTs já foram propostas como substratos de diversas aplicações distribuídas, incluindo grades computacionais [76, 80, 87], resolução de nomes [16, 71], sistemas de armazenamento [40, 84], vídeo sob demanda (VoD) e *streaming media* [9, 32], soluções de *backup* [15], bibliotecas de referências bibliográficas [97], prevenção de ataques DDoS [38], busca na Internet [31], filtros de mensagens não solicitadas (*anti-spam filtering*) [106], controle de versões de programas (VCS) [103], telefonia sobre IP (VoIP) [8, 91], jogos na Internet [39], compartilhamento de arquivos [61] e bancos de dados [35], entre outras. Esta ampla variedade de aplicações atesta a aceitação das DHTs como uma ferramenta distribuída útil, confirmando-se assim a sua consolidação como uma alternativa robusta às demais soluções para localização de informação em sistemas distribuídos de larga escala.

Deve-se notar que apesar de tabelas *hash* distribuídas serem comumente propostas de maneira a suportar ambientes vastos e dinâmicos, como os formados por computadores domésticos conectados à Internet, sistemas DHTs são também usados em ambientes controlados e redes locais (p.ex., [15, 21]). Ressaltamos ainda que, enquanto DHTs são também empregadas em redes sem fio e móveis [17, 34, 101], incluindo redes de sensores [27, 89], nesta tese nos concentraremos apenas em ambientes que disponham de uma camada de rede conectando todos os seus participantes, e que o endereço de rede de cada participante seja fixo enquanto ele estiver conectado ao sistema DHT. Deste modo, as DHTs como discutidas nesta tese podem também ser usadas por dispositivos móveis

que se conectam à Internet ou redes locais com uso de protocolos IPv4 e IPv6.

Sistemas DHT disponibilizam uma função de consulta similar à encontrada em tabelas *hash* tradicionais, sendo que as chaves (informação) são distribuídas pelos diversos nós participantes do sistema, tipicamente de forma randômica segundo uma função criptográfica (p.ex., SHA1 [60]). De maneira a encaminhar uma consulta a partir do seu *nó origem* até aquele responsável pela chave procurada (*nó destino*), DHTs implementam uma rede sobreposta entre os nós participantes (*overlay network)*, com o uso de tabelas de roteamento armazenadas nos diversos nós. A menos que cada tabela de roteamento seja grande o suficiente para armazenar os endereços de todos os nós do sistema, o roteamento de cada consulta deverá requerer diversos saltos entre os nós, ou seja, cada consulta percorrerá diversos nós antes de alcançar o seu destino. De uma maneira geral, quanto maior for a quantidade de saltos necessária para resolver uma consulta, maior deverá ser sua *latência*, ou seja, maior deverá ser o intervalo de tempo para que o nó origem obtenha a sua resposta.

Apesar de tabelas de roteamento grandes permitirem consultas mais rápidas, deve-se também considerar que quanto maior for a quantidade de informação de roteamento armazenada em cada nó, maior será a demanda de comunicação para sua manutenção à medida que ocorrem adesões e partidas de nós no sistema. Deste modo, DHTs devem balancear a latência das consultas (número de saltos) com a demanda de comunicação para manutenção das tabelas de roteamento. Assim, de uma maneira geral, o tamanho das tabelas de roteamento de sistemas DHT representa um compromisso entre latência e uso de banda passante de rede, já que quanto maior for a tabela de roteamento, menor deverá ser a latência das consultas, mas maior será a demanda de banda de rede para sua manutenção [102].

Como consequência deste compromisso entre latência e banda passante, a vasta maioria das DHTs já propostas optou por soluções em que cada consulta é resolvida com múltiplos saltos (p.ex., [24, 36, 51, 52, 66, 72, 83, 96, 105]) de maneira a minimizar o tráfego de manutenção, com prejuízo para a latência das consultas. Esta opção foi feita porque à época que estas MHDHTs (acrônimo do termo em inglês *Multi-Hop DHTs*) foram propostas, o principal desafio era o suporte a sistemas vastos e altamente dinâmicos, que incluíam computadores domésticos com conexões tão restritas quanto ligações discadas, tipicamente curtas e lentas, implicando em um ambiente caracterizado por escassez de banda passante de rede e altíssimas taxas de adesão e partida de participantes.

Por outro lado, em uma sociedade onde agilidade e informação são aspectos fundamentais, acreditamos que devemos priorizar soluções de compromisso que favoreçam a minimização da latência, especialmente se considerarmos que o crescimento acentuado da

oferta de banda passante de rede observado nos últimos anos não tem sido acompanhado por uma redução de latências ponto-a-ponto no mesmo ritmo. Devemos ainda considerar que esta tendência deve se manter, uma vez que compromissos similares entre latência e banda passante ocorrem em diversas outras tecnologias, e já foi demonstrado que *com o passar do tempo, o aumento da largura de banda passante é tipicamente maior do que o quadrado da redução das latências* [64]. Desta maneira, é de se esperar que as restrições de latência de rede tendam a se tornar cada vez mais críticas, mesmo a médio e curto prazo, frente a um horizonte em que as expectativas e necessidades de desempenho dos usuários e aplicações deverão crescer de maneira consonante com os acelerados e constantes ganhos tanto em termos de disponibilidade de banda passante quanto em velocidade de processamento.

1.2 Motivação

Em virtude do contexto acima exposto, mais recentemente foram introduzidas algumas DHTs que são capazes de resolver a grande maioria das consultas (p.ex., 99%) com apenas um salto, procurando assim reduzir a sua latência ao mínimo possível. Para tanto, cada participante destas SHDHTs (acrônimo do termo em inglês *Single-Hop DHTs*) armazena uma tabela de roteamento completa, ou seja, uma tabela de roteamento que inclui o endereço de rede de todos os demais participantes do mesmo sistema, buscando assim um ponto extremo no compromisso latência/banda passante. Além da minimização da latência das consultas, a introdução destas SHDHTs foi também motivada por resultados mostrando que, para sistemas estáveis ou com muitas consultas, SHDHTs podem na verdade exigir níveis de tráfego menores do que as MHDHTs [43, 81, 99].

Deve-se ainda ressaltar que, naturalmente, a alta latência imposta pelas MHDHTs tem sido um forte limitador à sua adoção em grandes sistemas distribuídos com desempenho crítico, representados principalmente pelos ambientes de computação de alto desempenho (ou HPC, acrônimo do termo em inglês *High Performance Computing*) (e.g., [62]), mas que incluem também diversos outros ambientes corporativos como, por exemplo, Provedores de Serviços de Internet (ou ISPs, acrônimo do termo em inglês *Internet Service Providers*) [5]. Na medida em que estes ambientes corporativos distribuídos têm crescido continuamente, podendo alcançar dezenas de milhares de nós [100], diretórios com arquitetura Cliente/Servidor têm se tornado uma solução cara e limitante, de maneira que a alta escalabilidade das SHDHTs as tornam uma solução alternativa promissora [21].

Por outro lado, a menos do sistema DHT que apresentaremos nesta tese, todas as demais SHDHTs já introduzidas incorrem em altíssimos custos de manutenção [45, 78, 99],

têm intrínsecos problemas de balanceamento de carga [23], ou não são capazes de suportar ambientes dinâmicos [21, 82]. Desta maneira, mesmo que estas SHDHTs possam ser usadas para um conjunto específico de aplicações, elas não são propícias para uma variada gama de ambientes, em especial para aplicações P2P populares em uso na Internet (p.ex., KAD [93]). Deste modo, o emprego destas propostas requer o desenvolvimento de um sistema DHT para uso por uma classe limitada de aplicações, restringindo significativamente as suas atratividade e viabilidade.

Consideramos que uma SHDHT que tenha efetivo potencial de uso para diversas aplicações deva ser capaz de satisfazer aos seguintes requisitos essenciais:

(i) Garantir que uma grande parte (p.ex., 99%) das consultas seja resolvida com um único salto mesmo em ambientes dinâmicos.

(ii) Demandar níveis aceitáveis de tráfego de manutenção.

(iii) Prover bom balanceamento de carga entre os nós.

(iv) Adaptar-se a mudanças no comportamento do sistema.

(v) Ser auto-reorganizável, preferencialmente com uso de uma arquitetura puramente P2P.

Uma SHDHT capaz de atender aos requisitos acima não só será uma ferramenta potencialmente útil e atraente para uso em diversas aplicações largamente distribuídas (incluindo sistemas P2P populares, uma vez que permitirá que os seus usuários tenham acesso mais rápido às informações desejadas), como também irá habilitar o uso de DHTs em ambientes com desempenho crítico (incluindo HPC, onde baixa latência é um requisito comumente obrigatório que impede o uso de MHDHTs, e as soluções Cliente/Servidor têm limitações de escalabilidade).

1.3 Tese

Em virtude desta motivação, esta tese propõe uma nova SHDHT, denominada D1HT, que é capaz de atender a todas as cinco características essenciais enumeradas acima, com uso de uma arquitetura puramente P2P e auto-reorganizável. D1HT utiliza um algoritmo eficiente e escalável para detecção e propagação de eventos (nesta tese, iremos nos referir às adesões e saídas de pares ao sistema simplesmente como *eventos*). Este algoritmo, o qual batizamos de EDRA (acrônimo do termo em inglês *Event Detection and Propagation Algorithm*), é capaz de agrupar diferentes notificações de eventos em uma mesma

mensagem, e ainda assim garantir que uma grande fração das consultas (p.ex., 99%) será resolvida com apenas um salto, mesmo em ambientes dinâmicos, sem impor hierarquias ou diferenciação de funções entre os participantes do sistema. Estas propriedades são formalmente provadas através de um teorema que será enunciado e demonstrado nesta tese, que também garante bom balanceamento de carga entre os participantes do sistema, e ainda provê mecanismos que permitem que D1HT se ajuste a variações da dinâmica e do tamanho do sistema de maneira a minimizar as suas demandas de banda passante de rede, sem comprometer a latência de suas consultas.

Iremos ainda introduzir um segundo teorema, que também será formalmente enunciado e provado, o qual será a base para desenvolvimento da análise teórica que quantificará o número de mensagens transmitidas e a demanda de banda passante de manutenção das tabelas de roteamento em D1HT.

Nesta tese estamos também introduzindo um mecanismo de quarentena que é capaz de reduzir os custos de manutenção causados por pares voláteis, além de poder tornar sistemas P2P mais robustos em relação a ataques maliciosos e multidões repentinas, mas requer que pares que tenham se conectado recentemente ao sistema resolvam suas consultas com dois saltos.

Como parte do trabalho desta tese, implementamos D1HT, resultando em um código fonte com cerca de oito mil linhas em linguagem C++. Apesar da estratégia proposta por D1HT ser simples, a busca por implementação que seja estável, correta e eficiente, mesma quando usada em sistemas dinâmicos com milhares de pares, resultou em uma estrutura multifluxo para cada processo D1HT, de maneira a habilitar um alto nível de concorrência na execução de suas diversas tarefas. A interação entre este nível de concorrência que ocorre *dentro* de cada processo D1HT, que tem perfil heterogêneo e comunicação através de memória compartilhada, com o paralelismo que ocorre *entre* processos D1HT, que tem caráter homogêneo e distribuído, tornou ainda mais desafiadora as tarefas de depuração, teste e ajuste desta implementação, mais permitiu que executássemos D1HT corretamente e eficientemente em sistemas com milhares de pares. A partir do código fonte de D1HT, implementamos também o sistema 1h-Calot [99], a única outra SHDHT puramente P2P que é capaz de suportar sistemas vastos e dinâmicos.

Com uso das nossas implementações de D1HT e 1h-Calot, realizamos vários experimentos com o objetivo de validar as propriedades de D1HT e compará-lo com a sua principal alternativa. Estas avaliações foram conduzidas com até 4.000 pares em dois ambientes completamente distintos (um CPD HPC e uma rede mundialmente dispersa na Internet), resultando na mais completa comparação experimental entre sistemas DHTs já publicada. Estes experimentos não só validaram as análises quantitativas das demandas de

manutenção de tabelas tanto de D1HT (introduzidas nesta tese) quanto de 1h-Calot (que ainda não haviam sido validadas experimentalmente), como também mostraram que estes dois sistemas são capazes de resolver mais de 99% das consultas com apenas um salto. Nossos experimentos ainda mostraram que D1HT tem demandas desprezíveis de processamento e memória, podendo assim ser usado até mesmo em servidores corporativos com desempenho crítico que estejam sob alta carga de processamento.

Enquanto a escalabilidade tem sido um argumento fundamental em favor das DHTs em relação a soluções com arquitetura Cliente/Servidor, ao mesmo tempo em que advoga-se que SHDHTs proporcionam latências inferiores àquelas das MHDHTs, desconhecemos experimentos que tenham comprovado qualquer uma destas duas propriedades na prática. Preenchendo estas duas importantes lacunas, nesta tese foram conduzidos experimentos com até 4.000 pares comparando-se as latências proporcionadas por quatro soluções distintas para resolução de consultas, sendo duas SHDHTs (D1HT e 1h-Calot), uma MHDHT (uma implementação de Pastry [10, 83]), e um servidor de consultas. Estes experimentos inéditos quantificaram as latências das quatro soluções avaliadas para vários tamanhos de sistema e diferentes situações de carga de processamento, confirmando que as SHDHTs consistentemente proporcionam menores latências que a MHDHT, além de confirmar a sua superior escalabilidade em relação ao servidor de consultas.

Uma vez que nossos experimentos validaram as análises quantitativas de D1HT e 1h-Calot, nós também realizamos uma comparação analítica das demandas de rede para manutenção das tabelas de roteamento entre estes dois sistemas e a SHDHT OneHop, a qual já tinha tido a sua análise quantitativa comprovada [23]. Nossos resultados analíticos compararam os custos de manutenção destas três SHDHTs com até 10 milhões de pares e distintas dinâmicas de adesão e partidas de pares, e mostraram que D1HT consistentemente apresenta as menores demandas de banda passante de rede, com reduções de até uma ordem de magnitude em relação aos outros dois sistemas avaliados. Estes resultados também mostraram que as reduzidas demandas de manutenção de D1HT viabilizam o seu uso até mesmo em aplicações P2P largamente difundidas na Internet, como KAD [93] e BitTorrent [68]. Nossos resultados analíticos mostraram ainda que o mecanismo de Quarentena proposto nesta tese é efetivo, reduzindo em até 24% e 31% os custos de manutenção em sistemas D1HT com dinâmicas similares às de Gnutella e KAD, respectivamente.

Considerando que DHTs não são aplicações em si, mas substratos úteis e escaláveis para uso por aplicações distribuídas, julgamos importante a validação das propriedades de D1HT com um sistema distribuído real. Perseguindo este objetivo, nesta tese nós projetamos e implementamos um sistema de arquivos puramente P2P e aderente ao padrão

POSIX, o qual batizamos de D1HF, que procura armazenar cópias de arquivos lidos em áreas em disco e memória locais aos diversos nós participantes, buscando assim acelerar o acesso a arquivos que são lidos por diversos nós, padrão de acesso presente em aplicações HPC chaves [62]. Em D1HF, as cópias cache dos arquivos são divididas em blocos, os quais são distribuídos pelos vários pares do sistema. A distribuição e localização destes blocos é responsabilidade de D1HT, de modo a garantir rápido acesso a estes dados de maneira escalável, e viabilizar a arquitetura puramente P2P e auto-reorganizável do sistema de arquivos.

D1HF foi implementado e avaliado em um ambiente com 800 nós físicos, segundo padrões de acesso sequencial e salteado representativos de aplicações HPC reais. Estes experimentos compararam o desempenho de D1HF contra uma solução que representa o estado da arte em termos de sistemas de arquivos comerciais para ambientes de produção corporativa, e comprovaram a superior escalabilidade de D1HF. Por outro lado, nossos experimentos também mostraram que D1HF pode trazer perdas de desempenho para padrões de acesso que não são adequados a sua estratégia, mas que esta questão não deve trazer maiores problemas uma vez que o uso de D1HF pode ser facilmente evitado nas situações em que este sistema de arquivos se mostre contraproducente. A análise das latências dos acessos aos arquivos em D1HF demonstrou que D1HT permitiu a rápida localização dos blocos dos arquivos para todos os tamanhos de sistema estudados, tendo tido portando um papel importante na escalabilidade de D1HF.

1.4 Contribuições

Em resumo, as principais contribuições - parte delas apresentadas em [55, 56, 57] - presentes nesta tese podem ser melhor apreciadas dividindo-as em vários aspectos, a saber, aspectos formais, estudos analíticos, desenvolvimento e avaliação experimental de sistemas DHT e suas aplicações, e acesso a tecnologia DHT criada, como se segue:

(i) **Aspectos Formais**

(a) Introdução de um inovador sistema DHT puramente P2P e auto-reorganizável, denominado D1HT, que procura resolver uma grande parte das consultas com um único salto mesmo em ambientes dinâmicos, impor níveis aceitáveis de tráfego de manutenção, prover bom balanceamento de carga entre os nós, e adaptar-se a mudanças no comportamento do sistema;

(b) Descrição formal do algoritmo utilizado em D1HT para detecção e propagação de eventos, denominado EDRA;

(c) Enunciação e demonstração de teorema comprovando as características de correção, desempenho e balanceamento de carga do algoritmo EDRA;

(d) Desenvolvimento da análise teórica quantitativa das demandas de manutenção de EDRA, com base nas propriedades de um segundo teorema que também foi formalmente enunciado e provado;

(e) Introdução de um mecanismo de quarentena para redução dos custos de manutenção causados por pares voláteis em sistemas P2P.

(ii) **Resultados de Estudos Analíticos**

(a) D1HT tem custos de manutenção até uma ordem de magnitude inferiores aos das únicas duas outras SHDHTs que são capazes de suportar ambientes dinâmicos;

(b) D1HT é escalável até um milhão de pares com custos de manutenção aceitáveis mesmo em ambientes com dinâmica similar às de aplicações P2P populares;

(c) O mecanismo de quarentena é efetivo na redução dos custos de manutenção associados a pares voláteis.

(iii) **Desenvolvimento e Avaliação Experimental Comparativa de Sistemas DHTs**

(a) Desenvolvimento e implantação de D1HT e da única outra SHDHT capaz de suportar ambientes dinâmicos com arquitetura puramente P2P;

(b) Realização de experimentos comparativos entre D1HT e a outra SHDHT implementada nesta tese, com até 4.000 pares e 2.000 nós físicos em dois ambientes distribuídos completamente distintos (um CPD HPC e uma rede mundialmente dispersa na Internet). Estes representam a mais completa e vasta comparação experimental entre sistemas DHTs já publicada, e não só validaram a análise quantitativa teórica de D1HT, como também confirmaram que D1HT é capaz de resolver a vasta maioria das consultas com apenas um salto, entre outras conclusões relevantes;

(c) Realização de experimentos comparando-se as latências de consultas de D1HT contra aquelas de outros três sistemas, sendo uma SHDHT, uma MHDHT e um servidor de consultas. Estes foram os primeiros experimentos publicados comparando latências destes tipos de solução, além de terem sido também os primeiros a efetivamente confirmar que as SHDHTs têm latências de consulta menores que as alcançadas pelas MHDHTs, além de serem mais escaláveis do que servidores de consulta.

(iv) **Desenvolvimento e Avaliação de Aplicação Baseada em DHT**

(a) Introdução e desenvolvimento de um novo sistema de arquivos puramente P2P denominado D1HF, que é baseado em D1HT e integralmente compatível com o padrão POSIX.

(b) Realização de experimentos comparando o desempenho de D1HF contra uma solução que representa o estado da arte em termos de sistemas de arquivos comerciais para ambientes de produção corporativa, segundo dois padrões de acesso que são representativos de aplicações HPC reais. Estes experimentos comprovaram a maior escalabilidade de D1HF, que atingiu desempenho até oito vezes superior ao do sistema de arquivos comercial avaliado.

(v) **Acesso à Tecnologia D1HT**

(a) Disponibilização do código fonte de D1HT de maneira livre e gratuita [54].

É importante ressaltar que as contribuições listadas acima foram obtidas em uma área com intensa atividade de pesquisa nos últimos anos, o que acentua a sua relevância. Além disto, este conjunto de contribuições compõe o que acreditamos ser o ciclo ideal para esta tese, uma vez que nós introduzimos uma solução inovadora e em seguida a implementamos junto com outro proeminente sistema proposto na literatura. Nós então provamos através de métodos formais, analíticos e experimentais a suas propriedades de correção, desempenho, custos e balanceamento de carga, além de demonstrarmos sua superioridade frente às demais alternativas já introduzidas na literatura, fazendo uso, inclusive, da mais completa comparação experimental já publicada na sua área de pesquisa. Complementando este ciclo, implementamos um sistema real fundamentado na solução que introduzimos, e realizamos experimentos que mostraram que este sistema pode alcançar ganhos de desempenho até oito vezes superior em relação a um moderno sistema comercial, o que comprovou a utilidade e a adequação da solução que introduzimos. Por fim, nós disponibilizamos o código fonte da nossa implementação de maneira livre e gratuita [54].

1.5 Organização da Tese

Esta tese está organizada da seguinte forma. No próximo capítulo serão discutidos os trabalhos relacionados, e em seguida será apresentado o sistema D1HT, para no Capítulo 4 definirmos e discutirmos o algoritmo EDRA que permite a manutenção das suas tabelas

de roteamento. Nos três capítulos seguintes apresentaremos o mecanismo de Quarentena, a nossa implementação de D1HT, e o sistema de arquivos D1HF desenvolvido nesta tese. Nos Capítulos 8 e 9 apresentaremos objetivamente o nosso conjunto de resultados experimentais e analíticos, que serão discutidos no Capítulo 10, para em seguida concluirmos esta tese. Deve-se notar que ao longo desta tese foram identificadas e discutidas várias oportunidades de trabalhos futuros, que ao final foram agrupadas e apresentadas junto às nossas conclusões. Incluímos também um glossário com os significados de diversos termos e abreviaturas usadas ao longo desta tese, de modo a facilitar a sua leitura e entendimento.

Capítulo 2

Trabalhos Relacionados

Neste capítulo iremos discutir diversos trabalhos conduzidos nas mesmas áreas de pesquisas abrangidas por esta tese, sempre procurando salientar as características que são singulares às contribuições científicas aqui introduzidas.

2.1 SHDHTs que Suportam Ambientes Dinâmicos

Além de D1HT, as únicas outras duas SHDHTs que suportam ambientes dinâmicos são os sistemas OneHop [23] e 1h-Calot [99]. Nesta tese apresentaremos diversos resultados experimentais e analíticos mostrando que D1HT tem custos de manutenção consistentemente menores que os destes outros dois sistemas, com reduções de até uma ordem de magnitude em relação a ambos. Além disto, OneHop e 1h-Calot diferem de D1HT em outros importantes aspectos, como discutiremos a seguir.

O sistema OneHop foi a primeira DHT proposta a garantir que a maior parte das consultas são resolvidas com um único salto, mesmo em ambiente dinâmicos [29]. Em contraste com a arquitetura puramente P2P de D1HT, a propagação das informações sobre eventos em OneHop é baseada em uma hierarquia, onde os nós são agrupados em *unidades* (ou *slices*), que por sua vez são agrupadas em *blocos*. Uma vez que cada unidade e cada bloco têm um líder, a hierarquia imposta por OneHop divide os nós participantes em três níveis distintos: nós comuns, líderes de unidades e líderes de blocos. Esta topologia hierárquica é construída com o objetivo de viabilizar o agrupamento de várias notificações de eventos em uma mesma mensagem, procurando-se assim minimizar as demandas de rede para manutenção das tabelas de roteamento. Para tanto, cada líder de unidade é responsável por acumular as informações sobre todos os eventos em sua própria unidade e periodicamente propagá-las ao líder do seu bloco. Cada líder de bloco agrupa os eventos ocorridos em suas diversas unidades e periodicamente os encaminha para todos

os outros líderes de bloco. Os diversos líderes de bloco irão então tomar conhecimento de todos os eventos ocorridos no sistema, e vão encaminhar estas informações para todos os líderes de unidades do seu bloco. Cada líder de unidade será então responsável por iniciar a propagação destes eventos por todos os nós de sua respectiva unidade. Mais detalhes sobre o sistema OneHop podem ser obtidos em [23].

Enquanto a hierarquia imposta por OneHop facilita o agrupamento de eventos de maneira a minimizar os seus custos de manutenção das tabelas de roteamento, D1HT consegue atingir o mesmo objetivo usando uma arquitetura puramente P2P. Deste modo, D1HT não só tem custos de manutenção inferiores (como veremos nos resultados que serão apresentados na Seção 8.2), como também evita vários problemas intrínsecos ao sistema hierárquico usado por OneHop. Primeiro, por que em OneHop há um alto nível de desbalanceamento de carga de manutenção entre os nós de diferentes níveis da hierarquia, como mostrado nos resultados publicados em [23] (que foram confirmados pelos nossos resultados analíticos que serão mostrados na Seção 8.2), sendo que seus autores não apresentam nenhuma solução para este problema (e suas graves consequências). Deve-se notar que este desbalanceamento de carga traz limitações para a escalabilidade do sistema, e torna mais difícil a tarefa de encontrar voluntários para arcar com as funções (e respectivos custos) de líderes de unidade ou bloco, uma vez que estas implicam em demandas de rede que são até uma ordem de magnitude superiores àquelas dos nós ordinários. Segundo, a falha, e consequente substituição, dos líderes de unidade e de bloco são críticas, sendo que os autores também não apresentam nenhum meio para minimizar este problema (na verdade, os resultados apresentados pelos seus autores simplesmente ignoram a ocorrência de falhas em líderes de unidades e blocos, e suas consequências). Terceiro, não foram apresentados (e não são claros) quais seriam os critérios para criação (e remoção) de unidades e blocos à medida que o sistema cresça (ou encolha), e provavelmente seriam necessários mecanismos complexos em implementações reais. Quarto, para atingir seu melhor desempenho e minimizar seus custos de manutenção, os participantes de um sistema OneHop devem concordar em relação aos valores ótimos de alguns parâmetros de topologia (por exemplo, os tamanhos de unidades e blocos), o que é uma tarefa complexa e difícil, especialmente se considerarmos que a melhor parametrização depende de fatores que podem variar de maneira contínua e imprevisível (como, por exemplo, as taxas de adesão e partida de nós) [79, 99].

O sistema 1h-Calot [99], desenvolvido de maneira simultânea e independente ao nosso trabalho, guarda algumas similaridades com D1HT, já que ambos os sistemas procuram resolver as consultas com um único salto e fazem a propagação de eventos com uso de árvores logarítmicas. Há, no entanto, diferenças fundamentais entre estes dois sistemas.

Primeiro, por que D1HT constrói suas árvores de propagação de eventos a partir de TTLs, enquanto 1h-Calot baseia-se em intervalos de identificadores (mais detalhes sobre a maneira como 1h-Calot constrói suas árvores de disseminação são apresentadas na Seção 8.1.1, ou podem ser obtidas a partir de [99]). Além disto, 1h-Calot não tem suas propriedades de desempenho e correção formalmente provadas. Mas a principal diferença entre estas duas SHDHTs reside no fato de 1h-Calot não ser capaz de agrupar eventos para minimizar seus custos de manutenção, e assim as suas demandas de banda passante são muito superiores às de D1HT, tornando inviável o seu uso em muitas aplicações. De fato, os resultados que serão apresentados no Capítulo 8 mostrarão que D1HT consistentemente apresenta custos de manutenção menores do que os de 1h-Calot, com reduções que são tipicamente de uma ordem de magnitude. Além disto, enquanto 1h-Calot não se mostra viável para sistemas vastos ou dinâmicos, nossos resultados indicaram que D1HT é escalável até milhões de pares mesmo sob dinâmicas típicas de ambientes formados por pares domésticos.

Deve-se ressaltar que a habilidade de D1HT para agrupar eventos sem comprometer o objetivo de resolver a vasta maioria das consultas com apenas um salto, mesmo em ambientes dinâmicos e com comportamento variável, decorre das propriedades de um teorema que foi formalmente introduzido e provado neste trabalho. Isto por que, para que uma SHDHT seja capaz de agrupar eventos de maneira eficaz, estes não podem ser propagados à medida que ocorram ou sejam conhecidos. Para tanto, cada par deve guardar os eventos conhecidos durante um *período de agrupamento* para então propagá-los em uma única mensagem. Deste modo, esta estratégia de agrupamento impõe atrasos à propagação dos eventos, o que força que as tabelas de roteamento dos diversos pares fiquem desatualizadas por mais tempo. Como consequência, o uso de períodos de agrupamento muito longos irá comprometer o objetivo de resolução das consultas com apenas um salto. Por outro lado, períodos de agrupamento muito curtos não permitirão reduções significativas dos custos de manutenção, já que as oportunidades de agrupamento de eventos serão menores. Assim, o cálculo eficiente da duração do período de agrupamento é uma tarefa fundamental para o sucesso deste mecanismo. Deste modo, uma vez que 1h-Calot não dispõe de nenhum mecanismo para cálculo da duração do período de agrupamento, ele não seria capaz de empregar esta estratégia de maneira eficaz sem comprometer a sua habilidade de resolução de consultas com um único salto, nem mesmo para hipotéticos sistemas que não sofressem variações de tamanho ou de comportamento dos seus participantes. Deve-se ainda notar que este cálculo não é simples, já que a duração ideal depende de fatores que podem variar de maneira imprevisível, como o tamanho do sistema e as taxas de adesão e partida de pares.

2.2 Outras SHDHTs

Algumas outras DHTs foram introduzidas com o objetivo de resolver consultas com um único salto, mas, com exceção de D1HT, 1h-Calot e OneHop, todas as demais SHDHTs já propostas suportam apenas ambientes corporativos e controlados, não sendo portanto adequadas para sistemas P2P vastos e dinâmicos. Para duas destas outras SHDHTs, 1HS [78] e SFDHT [45], todos os resultados analíticos de 1h-Calot que apresentaremos no Capítulo 8 deverão ser válidos, já que 1HS usa o mesmo mecanismo de propagação de eventos de 1h-Calot, e SFDHT usa um algoritmo quase idêntico, que tem a mesma demanda média de banda passante de manutenção. Mais especificamente, a diferença na propagação de eventos entre 1h-Calot e SFDHT se deve basicamente ao fato de 1h-Calot balancear a carga de manutenção uniformemente entre os diversos pares, enquanto SFDHT procura uma distribuição que considere a heterogeneidade dos nós. Assim, as demandas de manutenção médias de 1h-Calot, 1HS e SFDHT devem ser idênticas (como representadas pelos resultados de 1h-Calot mostrados no Capítulo 8), enquanto a distribuição desta carga média entre os seus diversos pares deve diferir entre SFDHT e os outros dois sistemas. Deste modo, assim como 1h-Calot, 1HS e SFDHT devem requerer demandas médias de manutenção de tabelas de roteamento que são até uma ordem de magnitude superiores às de D1HT.

A SHDHT proposta em [82] é absolutamente distinta de D1HT, uma vez que este sistema baseia-se em uma hierarquia de dois níveis usando somente servidores dedicados, com o objetivo principal de prover proteção contra ataques maliciosos. Dynamo [21], também difere substancialmente de D1HT, já que usa um mecanismo de fofoca (*gossip*) para propagação de eventos, além de não ser capaz de suportar ambientes dinâmicos.

2.3 $O(1)$ DHTs

D1HT resolve as consultas com um único salto enquanto, em contraste, alguns outros sistemas (p.ex., [1, 29, 30, 46, 53, 99]) resolvem as consultas com um número constante de múltiplos saltos (i.e., $O(1)$ saltos). Além de serem incapazes de garantir latências ótimas, estes sistemas também diferem de D1HT em outros importantes aspectos.

Tulip [1] e Kelips [30] usam mecanismos de fofoca para manter tabelas de roteamento com tamanho $O(\sqrt{n})$ e assim resolver as consultas com dois saltos. Z-Ring [46] usa o protocolo Pastry [83] com uma base logarítmica grande, além de algumas extensões, para resolver consultas com dois saltos em sistemas com até 16 milhões de pares. Structured Superpeers [53] implementa uma topologia hierárquica, com intrínsecos problemas de

balanceamento de carga, para resolver consultas com até três saltos.

TwoHop [29] e 2h-Calot [99] são duas DHTs desenvolvidas a partir de, respectivamente, OneHop e 1h-Calot, em que cada par armazena e mantém uma tabela de roteamento com tamanho $O(\sqrt{n})$ de modo a resolver as consultas com dois saltos, o que pode ser um ponto de compromisso entre latência de consultas e custos de manutenção que seja interessante para sistemas com taxas de eventos altas o suficiente para tornar as SHDHTs inadequadas ou mesmo inviáveis. Pretendemos futuramente introduzir, desenvolver e avaliar D2HT, construída a partir de D1HT no mesmo sentido. Uma vez que, ao menos para o primeiro salto de cada consulta, cada par em D2HT terá um leque de opções de roteamento, pretendemos fazer uso de informações de localidade [14] de maneira que cada par possa escolher a alternativa de menor latência dentre os diversos destinos disponíveis para cada salto. Na verdade, para aplicações que requisitem a replicação das chaves por diversos pares, a latência do segundo salto de cada consulta poderá também ser minimizada com uso destas informações de localidade. Uma vez que D2HT será construída com uso de D1HT, o seu desenvolvimento poderá, de fato, tornar D1HT mais popular, lhe permitindo alcançar ambientes com dinâmicas que tornam inviável a manutenção de tabelas de roteamento completas.

2.4 Avaliações Experimentais de DHTs

Ao longo do trabalho desta tese, realizamos diversos experimentos em dois ambientes radicalmente diferentes (um CPD corporativo HPC e uma rede de computadores mundialmente dispersa) com até 4.000 pares e 2.000 nós físicos distintos. Afora estes experimentos, que foram parcialmente apresentados em [56], todas as avaliações já publicadas de sistemas DHT utilizaram, no máximo, duas centenas de nós físicos e foram restritas a um único ambiente computacional (p.ex., [18, 23, 35, 74, 75, 105]). Desta maneira, os nossos resultados representam o maior e mais completo conjunto de experimentos com DHTs, além de ser o primeiro a comparar a latência de três DHTs distintas com as alcançadas por um servidor de diretório.

2.5 Sistemas de Arquivos P2P

Mesmo que o sistema D1HF não esteja no centro das contribuições desta tese, e que exista uma contínua atividade de pesquisa e desenvolvimento em torno de sistemas de arquivos, acreditamos que D1HF tenha singularidades importantes e inovadoras.

Os trabalhos mais próximos de D1HF são os sistemas propostos em [58] e [49], que

também têm arquitetura P2P e são construídos sobre DHTs, mas ambos têm estrutura e propósitos completamente distintos dos nossos. Ivy [58] armazena os seus arquivos segundo sequencias de *logs* e procura criar um sistema de arquivos a partir de pares não confiáveis, mesmo que esta estratégia acarrete perdas de desempenho, ao ponto de Ivy ter se mostrado pelo menos duas vezes mais lento do que NFS. O sistema proposto em [49] procura aproveitar espaços ociosos em discos de computadores dispersos em uma rede, de maneira a construir um sistema de arquivos tolerante a falhas com uso de arquivos transparentes [12], e também não tem preocupação com desempenho de acessos e tão pouco procura suportar aplicações paralelas. Em contraste, o objetivo principal de D1HF é justamente o de prover ganhos de desempenho para aplicações paralelas, e os experimentos que apresentaremos no Capítulo 9 vão mostrar que D1HF conseguiu desempenho até oito vezes superior ao de um robusto sistema comercial de arquivos de última geração.

CFS [18] implementa um sistema de arquivos sobre Chord [96], mas não é aderente ao padrão POSIX, não tem desempenho como objetivo, e não permite que as aplicações modifiquem arquivos já criados. Em contraste, além de ter desempenho como seu objetivo central, D1HF disponibiliza para as aplicações o conjunto completo de rotinas e operações definidas no padrão POSIX, permitindo que a as aplicações modifiquem ou removam os arquivos acessados através de D1HF da maneira usual.

Assim como D1HF, os sistemas de arquivos Google (GFS) [26] e BeeFS [90, 92] também constroem sistemas de arquivos a partir de discos locais de computadores conectados em redes locais, mas ambos têm estratégias e arquiteturas distintas das de D1HF, já que utilizam servidores para o armazenamento dos metadados, além de não fazerem uso de DHTs.

Existem ainda diversos outros sistemas P2P, como KAD [93], PAST [84], FARSITE [2], FreeNet [13] e OceanStore [40, 74], que procuram implementar soluções de armazenamento ou compartilhamento de dados, mas não podem ser vistos como sistemas de arquivos, e tem propósitos e estratégias distintas das empregadas em D1HF.

2.6 Outros Trabalhos Relacionados

Beehive [70] não é um sistema DHT em si, mas um mecanismo de replicação para ser aplicado a sistemas DHT de maneira a reduzir o número de saltos necessários para resolver consultas a chaves populares.

Accordion [44] também procura acelerar as consultas e minimizar a demanda de banda passante, mas suas estratégias são completamente distintas daquelas usadas em D1HT. Accordion implementa algumas técnicas interessantes de adaptação que buscam

acelerar as consultas dentro de limites pré-estabelecidos de consumo de banda passante, mas não é capaz de assegurar uma fração máxima de falhas de roteamento, além de não garantir que qualquer consulta seja resolvida com um único salto. Em contraste, D1HT busca sempre resolver as consultas com a menor latência, minimiza as demandas de banda passante de manutenção, e se adapta a mudanças na dinâmica do sistema de maneira a constantemente assegurar uma fração máxima de falhas de roteamento. Adicionalmente, as propriedades de correção, balanceamento de carga e desempenho de D1HT foram formalmente enunciadas e provadas.

Mecanismos de quarentena têm sido usados há centenas de anos [25] com diversos fins, mas fomos os primeiros a propor, avaliar e confirmar a eficácia deste tipo de estratégia para sistemas P2P.

Capítulo 3

Uma Visão Geral do Sistema D1HT

Um sistema D1HT é composto de um conjunto $\mathbb{D}$ de n pares e associa itens (ou chaves) aos pares com uso das técnicas de *consistent hashing* [37], onde tanto as chaves quanto os pares são convertidos para identificadores inteiros (IDs) no mesmo intervalo $[0 : N]$, com $N \gg n$. Tipicamente, o ID de uma chave é obtido aplicando-se a função criptográfica SHA-1 [60] ao valor da chave, o ID de um par é obtido aplicando-se a função SHA-1 ao seu endereço IP, e $N = 2^{160} - 1$. Para simplificar a apresentação, referiremos aos IDs dos pares e das chaves como se fossem respectivamente os próprios pares e chaves.

Os diversos IDs dos pares e chaves de um sistema D1HT são dispostos em um mesmo anel, onde o identificador 0 sucede o identificador N, e o *sucessor* e o *predecessor* de um identificador i são respectivamente os seus vizinhos no sentido horário e anti-horário. Cada chave é associada ao seu sucessor e pode, opcionalmente, ser replicada nos k pares seguintes no anel, sendo que o valor do parâmetro k pode ser ajustado pela aplicação.

Cada par em um sistema D1HT mantém uma tabela de roteamento com os endereços IP de todos os participantes do sistema. Desta maneira, qualquer consulta é trivialmente resolvida com apenas um salto, desde que esta tabela local de roteamento esteja atualizada. Caso um par p não tome conhecimento de um evento causado pela adesão ou partida de algum outro par do sistema, a sua tabela de roteamento ficará desatualizada, de maneira que p poderá encaminhar uma consulta a um par errado ou a um par que já tenha saído do sistema. Desde que cada par tenha conhecimento do seu verdadeiro sucessor [96], em ambos os casos a consulta deverá ser resolvida após algumas tentativas, mas irá demorar mais do que inicialmente esperado. Uma vez que esta consulta será resolvida com perda de desempenho mas não irá falhar, chamamos este tipo de ocorrência de *falha de roteamento* ao invés de um *erro de consulta*.

Uma vez que um dos principais objetivos de uma SHDHT é desempenho, deve-se procurar minimizar a ocorrência destas falhas de roteamento com o uso de um algoritmo

que consiga disseminar a ocorrência dos eventos sem exigir grandes demandas de rede, nem causar desbalanceamento de carga entre os pares. Este algoritmo será apresentado e discutido no Capítulo 4.

O armazenamento da tabela de roteamento requer memória local dos diversos pares do sistema. Em D1HT estes requerimentos são minimizados com o uso de uma tabela *hash* local para armazenamento dos endereços IP, já que o espaço de identificadores (IDs) é esparsamente ocupado (uma vez que $N \gg n$). O índice desta tabela *hash* é formado pelos próprios identificadores dos pares, evitando assim a necessidade de armazenamento destes IDs. Desta maneira, cada tabela de roteamento irá requerer $6 \cdot n$ octetos para armazenar os endereços IPv4 (incluindo porta) dos n pares participantes do sistema, além de uma área adicional para tratamento das colisões. Assim, para aplicações em ambientes corporativos, como CPDs HPC ou ISP, a tabela de roteamento de cada par irá requerer no máximo algumas centenas de milhares de octetos. Para um vasto sistema com um milhão de pares, cada tabela de roteamento irá consumir cerca de seis milhões de octetos, o que é simplesmente desprezível para computadores domésticos, e plenamente aceitável mesmo para pequenos dispositivos móveis individuais modernos, como telefones celulares, tocadores de música e máquinas fotográficas.

Para aderir a um sistema D1HT, o novo par deverá conhecer ao menos um outro par que já faça parte do sistema, e então seguir um protocolo de adesão (*joining protocol*). Deixaremos os detalhes do protocolo de adesão para serem definidos pela implementação, desde que sejam atendidos os seguintes requisitos: (i) A adesão de um par só poderá ser considerada como completa, e então iniciada a sua disseminação, quando o novo par tiver recebido a tabela de roteamento completa e todas as suas chaves; (ii) A disseminação da adesão a um par deverá ser feita segundo o algoritmo EDRA, que será apresentado no Capítulo 4; (iii) O sucessor (ou predecessor) do novo par deverá garantir que este receba todas as notificações de eventos que ocorram enquanto a sua adesão está sendo disseminada; (iv) O protocolo de adesão deverá ser capaz de receber extensões como, por exemplo, o mecanismo de Quarentena que será apresentado no Capítulo 5. No Capítulo 6 serão dados mais detalhes do protocolo de adesão implementado nesta tese.

Mesmo que cada consulta seja resolvida com apenas um salto, D1HT pode fazer uso de informações de localidade (p.ex., com uso de coordenadas como as providas por PIC [14]), de modo a minimizar a latência das consultas em algumas situações, como, por exemplo, quando as chaves são replicadas em um número de pares sucessores de seu ID de maneira a prover redundância. Neste caso, D1HT terá algumas opções para o único salto necessário para resolver cada consulta, e poderá então escolher aquele que potencialmente tenha a menor latência. Um outro exemplo de uso de localidade seria o sistema D2HT

que pretendemos futuramente desenvolver, para resolver consultas com dois saltos com uso de D1HT e tabelas de roteamento com tamanho $O(\sqrt{n})$, como discutido na Seção 2.3. D2HT deverá fazer uso de D1HT de maneira a obter uma lista de opções de pares para o primeiro salto de cada consulta, e a disponibilidade de informação sobre a localidade destas opções seria útil para minimizar a latência de suas consultas. Assim, para viabilizar as otimizações acima, cada par deverá obter as suas coordenadas quando da adesão ao sistema, que seriam propagadas por EDRA junto com seu endereço IP. Para endereçar esta situação em que a informação de localidade é necessária *antes* da execução de cada consulta, as tabelas de roteamento de D1HT conteriam não só os endereços IP de todos os pares do sistema como também suas coordenadas. Deste modo, aumentaríamos o tráfego de manutenção e a demanda de memória para armazenamento das tabelas de roteamento, de maneira que este tipo de uso de informações de localidade só deverá ser ativado quando ele puder realmente ser útil.

Além das situações descritas acima, algumas aplicações e sistemas poderão também usar D1HT não só para resolver consultas, como também para, em seguida, obter o endereço do par responsável pela chave buscada e sua localidade. Esta situação difere da anterior, porque aqui a informação de localidade não é necessária antes da realização da respectiva consulta. Os custos deste tipo de uso de localidade seriam menores, já que, neste caso, não seria necessária a propagação das coordenadas dos pares por EDRA nem seu armazenamento nas tabelas de roteamento, bastando que cada par ao responder uma consulta informe também a sua localidade. Além disto, neste caso os pares poderiam atualizar as suas coordenadas periodicamente, já que a localidade de pares em ambientes dinâmicos como a Internet não é uma propriedade estática.

D1HT usa uma arquitetura puramente P2P, mas a sua topologia *plana* não impede que seja usado como um componente de soluções hierárquicas que procuraram explorar a heterogeneidade de seus participantes. Por exemplo, a popular rede FastTrack [47], que tem milhões de usuários e é usada por diversas aplicações P2P (incluindo KaZaA [41]), tem duas classes de nós, Super Nós (SN) e Nós Comuns (ON). Basicamente, os SNs são mais bem provisionados de recursos (especialmente em termos de banda passante de rede), e cada SN age como um diretório central para um grupo de ONs, enquanto técnicas de inundação são usadas entre os SNs. Uma vez que foi observado que a rede FastTrack deve ter menos que 40 mil SNs com duração média de sessão de 2,5 horas [47], a análise quantitativa que apresentaremos no Capítulo 4 indica que poderíamos usar um sistema D1HT para conectar os diversos SNs com custos de manutenção inferiores a 1 kbps por SN, sem alterar o relacionamento hierárquico entre SNs e ONs. Estes custos de manutenção são desprezíveis, especialmente se considerarmos que os SNs são bem providos de

recursos de rede e que seriam evitados todos os custos associados às inundações, e ainda teríamos sensíveis ganhos no desempenho das consultas.

Neste trabalho não serão estudadas questões relativas a nós maliciosos e ataques de rede, apesar de ser claro que, devido ao seu maior grau de saída (proporcionado pelo uso de tabelas de roteamento completas), SHDHTs são menos vulneráveis a estes tipos de problemas do que as MHDHTs. Além disto, o mecanismo de Quarentena, que será apresentado no Capítulo 5, poderá ser usado de maneira a aumentar a robustez do sistema contra ataques maliciosos.

Capítulo 4

Manutenção das Tabelas de Roteamento

Como cada par em um sistema D1HT deve saber o endereço IP de todos os outros pares do sistema, qualquer evento deverá ser conhecido por todos os pares de uma maneira rápida, de modo a minimizar a existência de endereços desatualizados nas tabelas de roteamento. Por outro lado, uma vez que temos como objetivo o suporte a sistemas grandes e dinâmicos, devemos evitar mecanismos rápidos porém ingênuos (p.ex. *broadcast*) de propagação das informações sobre os eventos, de maneira a evitar sobrecargas na rede. Deste modo, a detecção e propagação dos eventos impõem três importantes desafios para D1HT: minimizar as demandas de rede, prover bom balanceamento de carga, e garantir um limite superior para a fração de entradas desatualizadas nas tabelas de roteamento (de modo a permitir que a grande maioria das consultas seja resolvida com um único salto). Para atender estes desafios, estamos introduzindo o algoritmo EDRA (acrônimo do termo em inglês *Event Detection and Propagation Algorithm*) que é capaz de notificar qualquer evento a todos os pares do sistema em tempo logarítmico, tem ótimas características de balanceamento de carga, e baixa demanda de rede se comparado com outras SHDHTs. Adicionalmente, EDRA é capaz de se adaptar a mudanças no comportamento do sistema de maneira a continuamente satisfazer a um limite pré-definido de falhas de roteamento, usando uma arquitetura puramente P2P e auto-reorganizável.

Neste capítulo iremos apresentar EDRA formalmente através de regras bem definidas, para então enunciar e provar teoremas que atestam a sua correção e garantem que EDRA tem características ótimas em relação às demandas de rede e balanceamento do tráfego de chegada, além de permitir o desenvolvimento da análise quantitativa das suas demandas de rede. Concluiremos também que EDRA tem bom balanceamento do tráfego de saída para sistemas dinâmicos. Adicionalmente mostraremos como EDRA se comporta na presença de atrasos nas mensagens (*message delays*) e intervalos Θ assíncronos, e como podemos ajustá-lo de maneira a adaptar-se a mudanças na dinâmica do sistema.

Serão agora definidas algumas funções que facilitarão a apresentação dos próximos tópicos deste trabalho. Em um sistema D1HT composto por um conjunto $\mathbb{D}$ de pares (i.e., $n = |\mathbb{D}|$), para qualquer $i \in \mathbb{N}$ e $p \in \mathbb{D}$, o i_{esimo} sucessor de p é dado pela função $succ(p,i)$, onde $succ(p,0) = p$ e $succ(p,i)$ é o sucessor de $succ(p,i-1)$ para $i > 0$. Note que para $i \geq n$, $succ(p,i) = succ(p,i-n)$. De maneira similar, o i_{esimo} predecessor de um par p é dado pela função $pred(p,i)$, onde $pred(p,0) = p$ e $pred(p,i)$ é o predecessor de $pred(p,i-1)$, para $i > 0$. Para qualquer $p \in \mathbb{D}$ e $k \in \mathbb{N}$, $stretch(p,k) = \{\forall p_i \in \mathbb{D} \mid p_i = succ(p,i) \ \wedge \ 0 \leq i \leq k\}$. Deve-se notar que $stretch(p,n-1) = \mathbb{D}$ para qualquer $p \in \mathbb{D}$ [51].

4.1 Propagação de Eventos

Iniciaremos esta seção com uma breve descrição de EDRA para então apresentar a sua definição formal.

De modo a permitir o agrupamento de eventos e assim minimizar as demandas de banda passante para manutenção das tabelas de roteamento, cada par p agrupa os eventos recebidos durante cada intervalo de Θ segundos (intervalos Θ), e os propaga através de até ρ mensagens de manutenção, onde o valor de Θ é dinamicamente ajustado segundo o comportamento do sistema (como será visto na Seção 4.6), e ρ é definido pela equação:

$$\rho = \lceil \log_2(n) \rceil \tag{4.1}$$

Cada mensagem de manutenção $M(l)$, $l \in [0 : \rho)$, tem um contador $TTL = l$ (TTL é acrônimo do termo em inglês *Time To Live*) e é sempre endereçada ao par $succ(p,2^l)$. Adicionalmente, p irá incluir em cada mensagem $M(l)$ todos os eventos recebidos através de qualquer mensagem $M(j)$, $j > l$, durante os últimos Θ segundos. Para iniciar a disseminação de um evento, o sucessor do par que sofreu o evento irá reportá-lo em todas as ρ mensagens que enviará ao final do intervalo Θ corrente. A Figura 4.1, que será melhor descrita na Seção 4.2, ilustra como EDRA dissemina a informação sobre um evento em um sistema D1HT com onze pares.

As regras a seguir definem formalmente o algoritmo EDRA que foi brevemente descrito acima:

Regra 1: Todo par irá enviar ao menos uma e no máximo ρ mensagens de manutenção ao final de cada intervalo de tempo de Θ segundos (Intervalo Θ), onde $\rho = \lceil \log_2(n) \rceil$.

Regra 2: Cada mensagem de manutenção $M(l)$ terá um contador $TTL = l$ distinto no intervalo $[0 : \rho)$, e irá incluir informação sobre um conjunto de eventos. Todos os eventos recebidos através de uma mensagem $M(l)$ serão *conhecidos* com $TTL = l$ pelo par que a receber.

Regra 3: Cada mensagem somente irá conter informações sobre eventos conhecidos durante o intervalo Θ recém finalizado. Cada par irá incluir em cada mensagem $M(l)$ todos os eventos que ele tenha conhecido com $TTL > l$ durante o intervalo Θ que está sendo finalizado. Eventos que tenham sido conhecidos com $TTL = 0$ não serão incluídos em nenhuma mensagem.

Regra 4: As mensagens com $TTL = 0$ serão enviadas mesmo que não haja nenhum evento a ser reportado. Mensagens com $TTL > 0$ só serão enviadas quando houver eventos a reportar.

Regra 5: Se um par P não receber nenhuma mensagem do seu predecessor P_p por T_{detect} segundos, P irá verificar se P_p continua presente no sistema e, caso contrário, P irá assumir que P_p abandonou o sistema.

Regra 6: Quando um par P detecta um evento com seu predecessor (o predecessor aderiu ou saiu do sistema), P considera este evento como tendo sido conhecido com $TTL = \rho$ (e então, de acordo com a Regra 3, este evento será propagado através de ρ mensagens a serem enviadas por P ao final do intervalo Θ corrente).

Regra 7: Cada par P enviará as mensagens com $TTL = l$ para $succ(P, 2^l)$.

Regra 8: Antes de enviar qualquer mensagem para $succ(P,k)$, P irá remover da mensagem os eventos sofridos por qualquer par em $stretch(P,k)$.

Deve-se notar que as regras acima definem que cada par em um sistema D1HT não irá imediatamente propagar os eventos que venha a receber, o que viabiliza o agrupamento de todos os eventos recebidos em um intervalo Θ em no máximo ρ mensagens. Este mecanismo permite a redução do número de mensagens enviadas, mas o valor de Θ deve ser cuidadosamente escolhido, como será visto na Seção 4.6. Deve-se notar também que EDRA usa os contadores TTL de uma maneira diferente da usual, já que um evento que tenha sido conhecido com $TTL = l$, $l > 0$, não será propagado através de apenas uma

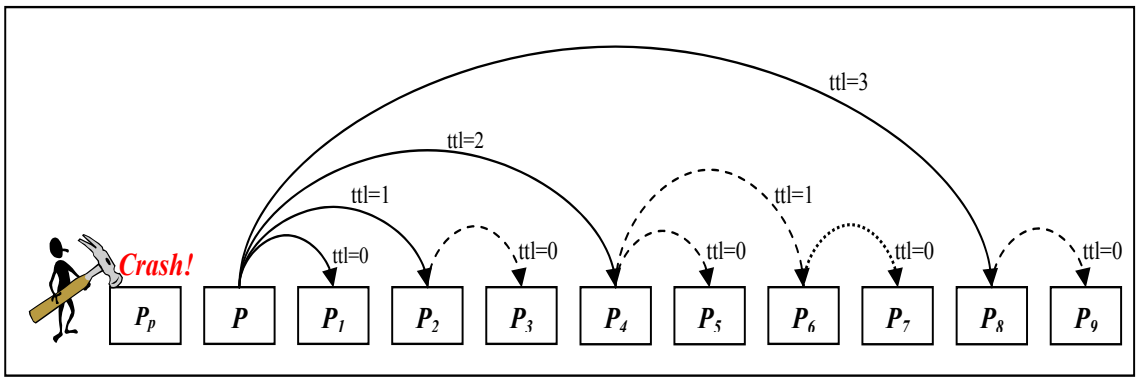


Figura 4.1: Esta figura mostra um sistema D1HT com 11 pares ($\rho = 4$), onde o par P_p falha e este evento é detectado pelo seu sucessor P, e propagado para todo o sistema segundo as regras de EDRA. Os pares no sistema estão representados em linha (ao invés de em anel) para facilitar a apresentação. Os pares P_i são tais que $P_i = succ(P,i), 1 \leq i \leq 9$. A figura também mostra o TTL de cada mensagem enviada.

mensagem com $TTL = l - 1$, e sim através de l mensagens, cada uma com um valor de TTL no intervalo $[0 : l)$.

As Regras 4 e 5 permitem que cada par mantenha ponteiros para os seus verdadeiros predecessor e sucessor, mesmo em casa de falhas. Adicionalmente, com o objetivo de aumentar a robustez do sistema, cada par P deverá executar uma rotina de estabilização de ponteiros periodicamente, ou quando receber uma mensagem com $TTL = 0$ de um par diferente daquele que P acredita que seja seu predecessor.

A Figura 4.1 mostra como EDRA dissemina as informações sobre os eventos, e ilustra algumas propriedades que enunciaremos e provaremos formalmente na próxima seção. A figura apresenta um sistema D1HT com 11 pares ($\rho = 4$), sendo que, para facilitar a apresentação, os pares na figura são mostrados em linha ao invés de em anel. Em um determinado instante o par P_p falha e este evento ε é detectado por seu sucessor P com $TTL = \rho$ (Regra 6) depois de T_{detect} segundos (Regra 5). De acordo com as Regras 3 e 7, P irá encaminhar ε através de $\rho = 4$ mensagens endereçadas a $P_1 = succ(P, 2^0)$, $P_2 = succ(P, 2^1)$, $P_4 = succ(P, 2^2)$ e $P_8 = succ(P, 2^3)$, como representado pelas setas contínuas na figura. Os pares P_2, P_4 e P_8 irão tomar conhecimento de ε com $TTL > 0$ (Regra 2) e irão então encaminhar ε com mensagens endereçadas a $P_3 = succ(P_2, 2^0)$, $P_5 = succ(P_4, 2^0)$, $P_6 = succ(P_4, 2^1)$ e $P_9 = succ(P_8, 2^0)$, como representado pelas setas tracejadas na figura. Como P_6 irá tomar conhecimento de ε com $TTL = 1$, ele ainda irá encaminhar ε para $P_7 = succ(P_6, 2^0)$ (seta pontilhada). Deve-se ainda notar que a Regra 8 irá impedir que P_8 encaminhe ε para $succ(P_8, 2^1)$ e $succ(P_8, 2^2)$, que são na verdade P e P_3, evitando que estes dois pares recebam ε duas vezes.

4.2 Correção

As regras apresentadas garantem que EDRA irá notificar qualquer evento a todos os pares de um sistema D1HT em tempo logarítmico, como formalmente demonstraremos a seguir. Para o Teorema 1 não será considerada a existência de retardos na transmissão de mensagens (*message delays*) e iremos assumir que todos os pares são capazes de manter intervalos Θ síncronos, i.e., que os intervalos Θ de todos os pares sempre iniciam simultaneamente. A ausência de retardo nas mensagens significa que as mensagens chegarão imediatamente ao seu destino, e, uma vez que estamos também considerando que todos os intervalos Θ são síncronos, temos que qualquer mensagem enviada ao final de um intervalo Θ irá chegar ao seu destino exatamente no início do intervalo Θ subsequente (como representado na Figura 4.2, que será melhor descrita na Seção 4.3). Além disto, iremos também considerar que nenhum novo evento ocorrerá até que o evento anterior tenha sido informado a todos os pares do sistema. Na Seção 4.3 iremos considerar a influência destes efeitos.

Teorema 1. *Seja qualquer par p, $p \in \mathbb{D}$, onde $\mathbb{D}$ é o conjunto de pares de um sistema D1HT. Um evento ε que tenha sido conhecido por p com $TTL = l$, e por nenhum outro par em $\mathbb{D}$, será encaminhado por p através de l mensagens, de maneira que ε será conhecido exatamente uma vez por todos os pares em $stretch(p, 2^l - 1)$ e por nenhum outro par em $\mathbb{D}$. O tempo médio para que os pares em $stretch(p, 2^l - 1)$ tomem conhecimento de ε será no máximo $l \cdot \Theta/2$ segundos após p tê-lo conhecido.*

Prova: Por indução forte em l. Para $l = 1$ as regras estabelecem que p somente irá encaminhar ε através de uma única mensagem com $TTL = 0$ endereçada ao par $succ(p, 1)$. Como esta mensagem será enviada ao final do intervalo Θ corrente, $succ(p, 1)$ irá tomar conhecimento de ε no máximo Θ segundos depois que p o tenha conhecido. Desta maneira, o tempo médio para que todos os pares em $stretch(p, 2^l - 1) = stretch(p, 1) = \{p, succ(p, 1)\}$ tomem conhecimento de ε será (no máximo) $(0 + \Theta)/2 = \Theta/2$.

Para $l > 1$, as regras de EDRA estabelecem que p irá encaminhar ε através de l mensagens ao final do intervalo Θ corrente, sendo que cada mensagem terá um contador TTL distinto no intervalo $[0 : l)$. Desta maneira, após Θ segundos (no máximo) cada par $p_k = succ(p, 2^k)$, $0 \leq k < l$, terá tomado conhecimento de ε exatamente uma vez através de uma mensagem com $TTL = k$. Aplicando a hipótese de indução a cada uma destas l mensagens recebidas pelos diversos pares p_k, e considerando que a Regra 8 evita que ε seja recebido mais de uma vez por qualquer par em $stretch(p, 2^\rho - n)$, teremos então que ε será recebido exatamente uma vez por todos os pares em $stretch(p, 2^l - 1)$. Como nenhum

destes pares irá retransmitir ε para nenhum outro par fora de $stretch(p, 2^l - 1)$, nenhum outro par do sistema irá receber ε. A hipótese de indução também assegura que o tempo médio para que os pares em cada conjunto $stretch(p_k, 2^k - 1)$ tomem conhecimento de ε será (no máximo) $k \cdot \Theta/2$ segundos após o respectivo par p_k ter tomado conhecimento de ε, e assim teremos que o tempo médio para os pares em $stretch(p, 2^l - 1)$ tomarem conhecimento de ε será no máximo $l \cdot \Theta/2$ segundos. ■

Com os resultados do Teorema 1, iremos a seguir não só provar a correção de EDRA (i.e., que EDRA é capaz de notificar qualquer evento a todos os pares de um sistema D1HT), como também que cada par só será notificado uma única vez sobre cada evento, e ainda definir o tempo médio para que todos os pares do sistema sejam notificados.

Corolário 1. *Seja um sistema D1HT composto por um conjunto $\mathbb{D}$ de pares. Qualquer evento ε que venha a ocorrer com qualquer p, $p \in \mathbb{D}$, será notificado uma única vez a todos os pares em $\mathbb{D}$, e o limite superior do tempo médio para esta notificação será $T_{sync} = \rho \cdot \Theta/2$.*

***Prova**:* Segundo a Regra 6, qualquer evento com qualquer par p será conhecido com $TTL = \rho$ por seu sucessor $succ(p, 1)$.

Aplicando-se as propriedades do Teorema 1 teremos então que este evento será conhecido uma única vez por todos os pares em $stretch(succ(p, 1), 1 + 2^\rho - 1) = \mathbb{D}$, e o tempo médio para que esta notificação seja recebida pelos pares em $\mathbb{D}$ será no máximo $T_{sync} = \rho \cdot \Theta/2$. ■

O Corolário 1 nos garante três resultados muito importantes sobre EDRA. Primeiro, que qualquer evento será notificado a todos os pares de um sistema D1HT, o que atesta a sua correção, assegurando que os todos pares de um sistema D1HT receberão as informações necessárias para a manutenção de suas tabelas de roteamento. Segundo, que cada evento será notificado uma única vez a cada par, o que mostra que D1HT evita o desperdício de banda passante e provê bom balanceamento de carga do tráfego de entrada. Terceiro, determina um teto para o tempo médio de notificação de qualquer evento, resultado este que será usado na Seção 4.6 para determinação do valor ótimo de Θ.

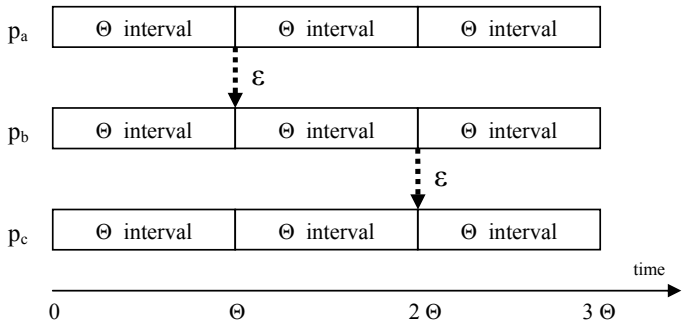


Figura 4.2: Propagação de um evento ε com intervalos Θ síncronos e na ausência de retardos nas mensagens.

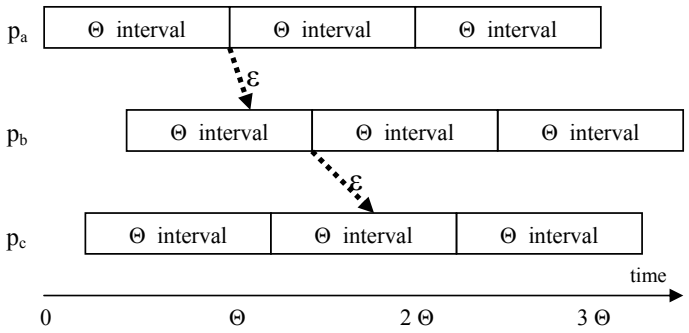


Figura 4.3: Propagação de um evento ε com intervalos Θ assíncronos e retardos nas mensagens.

4.3 Aspectos Práticos

No Teorema 1, não foram considerados os efeitos de atrasos em mensagens e intervalos Θ assíncronos, bem como a ocorrência de eventos concorrentes, os quais serão abordados nesta seção.

As Figuras 4.2 e 4.3 ilustram como EDRA faz a propagação de um evento ε em circunstâncias ideais e na presença de atrasos em mensagens e intervalos Θ assíncronos, respectivamente. Cada uma destas duas figuras representa três intervalos Θ para cada par. A Figura 4.2 mostra a situação ideal onde não ocorrem atrasos nas mensagens e os intervalos Θ de todos os pares iniciam simultaneamente, sendo que as setas pontilhadas mostram as mensagens propagando ε. Nesta situação hipotética, cada par adiciona exatamente Θ segundos ao tempo de propagação de ε, resultando no tempo de propagação médio de $T_{sync} = \rho \cdot \Theta/2$ segundos, como demonstrado no Corolário 1.

A Figura 4.3 ilustra a situação real típica aonde os vários intervalos Θ não são síncronos e as mensagens sofrem atrasos. Iremos considerar que o atraso médio de mensagens

para o sistema será δ_{avg} (o qual já inclui o tempo médio despendido com retransmissões). Assim, em média cada mensagem levará δ_{avg} para alcançar o par destino, aonde tipicamente chegará no meio de um intervalo Θ. Desta maneira, cada par no caminho de disseminação de um evento irá em média adicionar $\delta_{avg} + \Theta/2$ ao tempo de propagação de ε, resultando no tempo médio de propagação ajustado

$$T_{async} = \frac{\rho \cdot (2 \cdot \delta_{avg} + \Theta)}{4} \tag{4.2}$$

Deve-se notar que este valor de T_{async} ainda não considera o tempo gasto para detecção do evento. Como um par levará até T_{detect} para detectar um evento com seu predecessor, o tempo médio de propagação de um evento para todos os pares de um sistema D1HT será no máximo $T_{detect} + T_{async}$ segundos após o evento ter acontecido.

De agora em diante vamos assumir que $T_{detect} = 2 \cdot \Theta$, o que reflete o pior caso onde após uma mensagem com $TTL = 0$ perdida, um par p sonda o seu predecessor p_p e, uma vez tendo confirmado a partida de p_p, p irá propagar este evento ao final do intervalo Θ subsequente. Desta maneira, podemos calcular o limite superior do tempo médio para propagação de qualquer evento para todos os pares de um sistema D1HT:

$$T_{avg} = 2 \cdot \Theta + \frac{\rho \cdot (2 \cdot \delta_{avg} + \Theta)}{4} \tag{4.3}$$

A Equação 4.3 é conservadora uma vez que só considera o pior caso de falhas de pares, enquanto $T_{detect} = 0$ para adesões e saídas voluntárias do sistema.

Para o Teorema 1 foi também considerado que nenhum novo evento ocorreria até que todos os pares do sistema tivessem sido notificados a respeito do evento anterior, o que não é uma hipótese razoável para sistemas dinâmicos reais. Enquanto a admissão de novos pares concorrentemente a outros eventos pode ser devidamente tratada pelo protocolo de adesão, a partida de pares traz alguns problemas, especialmente as causadas por falhas. Por exemplo, quando um par falha antes de propagar eventos recebidos, a árvore de disseminação destes eventos ficará incompleta, e por consequência alguns pares não receberão todas as devidas notificações. Devemos, no entanto, ressaltar que este efeito ocorre em outras DHTs sem causar muita preocupação. Por exemplo, falhas de líderes de unidades e grupos em OneHop podem levar à perda de vários eventos, mas o efeito destas falhas não é computado em seus resultados analíticos [23]. Por outro lado, podemos dizer, de fato, que ao longo desta tese realizamos vários experimentos em diferentes ambientes que mostraram que D1HT é capaz de resolver a vasta maioria das consultas com um único

salto, mesmo na presença de altas taxas de adesão e partidas concorrentes. Desta maneira, os nossos resultados experimentais, que serão apresentados no Capítulo 8, são uma forte evidência de que a fração de falhas de roteamento causadas por estas questões práticas não é significativa em implementações reais.

Afora os problemas acima discutidos, na prática podem ocorrer diversas outras situações que causem problemas na propagação de eventos em D1HT (bem como em outras DHTs), e não seríamos capazes de remediá-las todas. Deste modo, como em vários outras DHTs (por exemplo, [23, 30, 44, 52]), qualquer implementação de D1HT deve permitir que os seus pares aprendam a partir das mensagens de manutenção e consultas recebidas e efetuadas, de maneira a fazer sem custos correções adicionais nas suas tabelas de roteamento. Por exemplo, uma consulta ou mensagem de manutenção recebida de um par localmente desconhecido deve implicar na sua inserção na tabela de roteamento. Adicionalmente, uma falha de roteamento indica que a adesão ou partida de um par não foi refletida na tabela de roteamento local, e permite que este desvio seja corrigido.

4.4 Balanceamento de Carga e Desempenho

O Corolário 1 não só prova que todos os pares de um sistema D1HT irão receber as informações necessárias para manutenção de suas tabelas de roteamento em tempo logarítmico, como também que nenhum par irá receber informações redundantes. Estes resultados mostram que cada par em D1HT irá receber cada evento uma (e somente uma) vez, confirmando que EDRA faz bom uso da rede disponível e provê balanceamento de carga ótimo em termos de tráfego de entrada.

Como nenhum par irá trocar mensagens de manutenção com pares que não pertençam a $\mathbb{D}$, podemos assegurar que os tráfegos médios de entrada e saída serão idênticos, como também as quantidades de mensagens enviadas e recebidas. Por outro lado, em um primeiro momento tem-se a impressão que EDRA não é capaz de garantir uma boa distribuição do tráfego de saída. Por exemplo, na propagação do evento ε ilustrada na Figura 4.1, P enviou quatro mensagens, enquanto P_1 não enviou uma sequer.

Podemos mostrar que em relação ao tráfego de manutenção para reportar um evento solitário, a carga máxima ocorrerá no sucessor do par que sofreu o evento, e será $O(\log(n))$ maior que a carga média. Entretanto, este desbalanceamento de carga pontual não é preocupante uma vez que o nosso objetivo é o de suportar sistemas grandes e dinâmicos, aonde podem ocorrer vários eventos por segundo. Desta maneira, não devemos nos preocupar com a carga gerada por eventos solitários, e sim otimizar o balanceamento da carga agregada necessária para disseminação de todos os eventos no sistema.

Em um sistema D1HT o balanceamento de carga em termos de número de mensagens e tráfego de saída é obtido em razão das propriedades randômicas da função utilizada para geração dos identificadores dos pares e chaves. Esta função deverá ser capaz de distribuir aleatoriamente os identificadores dos pares pelo espaço de identificação (anel), o que pode ser conseguido com o uso de uma função criptográfica como a função SHA-1 [60]. Assim, como em vários outros trabalhos (p.ex. [18, 29, 39, 42, 43, 48, 51, 75, 96]), iremos assumir que os eventos são randomicamente distribuídos ao longo do anel e ocorrem segundo uma taxa de eventos r, de modo que os tráfegos médios de entrada e saída por par serão dados por (incluindo as confirmações de recebimento de mensagens):

$$\frac{2 \cdot N_{msgs} \cdot (v_m + v_a) + r \cdot b \cdot \Theta}{\Theta} \text{ bps} \tag{4.4}$$

aonde Θ é expresso em segundos, N_{msgs} é o número médio de mensagens que cada par envia (e recebe) por intervalo Θ, b é o número médio de *bits* necessários para descrever um evento, e v_m e v_a são respectivamente o número de *bits* por cabeçalho de mensagem de manutenção e de confirmação de recebimento (*ack*).

Devemos salientar que a Equação 4.4 não requer que r se mantenha constante ao longo do tempo. Na verdade, r poderá variar mesmo em nossa mais simples hipótese, uma vez que iremos assumir que a dinâmica de um sistema D1HT pode ser representada por seu tamanho médio de sessão S_{avg}, como em [29]. Aqui nos referimos por *tamanho de sessão* (ou *duração de sessão*) como o intervalo de tempo que um par esteja continuamente conectado ao sistema D1HT, ou seja, o intervalo de tempo entre a entrada de um par no sistema e a sua subsequente saída. Como cada par gera dois eventos por sessão (uma entrada e uma saída), a taxa de eventos pode ser calculada por:

$$r = \frac{2 \cdot n}{S_{avg}} \tag{4.5}$$

Uma vez que existem medições da duração média das sessões de diferentes sistemas P2P [3, 7, 85, 93], a equação acima nos permite calcular taxas de eventos que são representativas de aplicações P2P largamente utilizadas, como Gnutella, BitTorrent e KAD.

De acordo com a Equação 4.5, a taxa de eventos r é diretamente proporcional ao tamanho do sistema. Hipóteses realistas devem também assumir que o tamanho médio de sessão S_{avg} (e portanto r) pode variar ao longo do tempo, de maneira a considerar sistemas em que, por exemplo, a duração média das sessões durante o dia difere daquela observada à noite. Na Seção 4.6 mostraremos que EDRA é capaz de se adaptar a variações em r de

modo a continuamente garantir que uma fração mínima das consultas será resolvida com um único salto, mesmo quando o comportamento do sistema varia ao longo do tempo. No Capítulo 8 serão apresentados resultados experimentais e analíticos com diferentes valores de r, que nos vão permitir avaliar o seu efeito nos custos de manutenção de D1HT.

4.5 Número de Mensagens

A Equação 4.4 requer que saibamos a quantidade média de mensagens que cada par envia e recebe, que é o objetivo do teorema a seguir.

Teorema 2. *O conjunto de pares S para o qual um par genérico $p \in \mathbb{D}$ toma conhecimento de eventos com $TTL \geq l$ é tal que $|S| = 2^{\rho-l}$.*

Prova: Por indução em j, onde $j = \rho - l$. Para $j = 0$, a Regra 2 garante que não há mensagens com $TTL \geq l = \rho$. Assim, os únicos eventos que p pode tomar conhecimento com $TTL \geq l$ são aqueles relacionados com seu predecessor (Regra 6), de modo que $S = \{pred(p,1)\}$ e portanto $|S| = 1 = 2^0 = 2^{\rho-l}$.

Para $j > 0$, temos que $l = \rho - j < \rho$. Como S é o conjunto de pares para os quais p toma conhecimento de eventos com $TTL \geq l$, podemos dizer que $S = S1 \cup S2$, onde $S1$ e $S2$ são respectivamente os conjuntos de pares para os quais p toma conhecimento de eventos com $TTL = l$ e $TTL > l$. Como o Teorema 1 assegura que p toma conhecimento de cada evento uma única vez, p irá tomar conhecimento de cada evento com um único TTL, o que nos garante que $S1$ e $S2$ são disjuntos. Da hipótese de indução podemos afirmar que $|S2| = 2^{\rho-(l+1)}$. Como $l < \rho$, $S1$ não inclui o predecessor de p (Regra 6), e a Regra 7 assegura que p somente recebe mensagens $TTL = l$ de um par k, $k = pred(p, 2^l)$. Assim, da Regra 3 temos que $S1$ será o conjunto de pares para os quais k toma conhecimento de eventos com $TTL > l$, e então, como a hipótese de indução também se aplica a k, temos que $|S1| = 2^{\rho-(l+1)}$. Como $S1$ e $S2$ são disjuntos, podemos então afirmar que $|S| = |S1| + |S2| = 2^{\rho-(l+1)} + 2^{\rho-(l+1)} = 2^{\rho-l}$. ■

As Regras 3 e 4 asseguram que um par p somente irá enviar uma mensagem com $TTL = l > 0$ caso tome conhecimento de pelo menos um evento com $TTL \geq l + 1$. Dos resultados do Teorema 2 podemos então dizer que p só enviará uma mensagem com $TTL = l > 0$ se ao menos um par em um conjunto de $2^{\rho-l-1}$ pares sofrer um evento. Como a probabilidade de um par genérico sofrer um evento em um intervalo Θ é $\Theta \cdot r/n$,

teremos, com ajuda da Equação 4.5, que a probabilidade $P(l)$ de um par genérico enviar uma mensagem com $TTL = l > 0$ ao final de cada intervalo Θ é:

$$P(l) = 1 - (1 - 2 \cdot \Theta / S_{avg})^k, \text{ onde } k = 2^{\rho - l - 1} \tag{4.6}$$

Como a mensagem com $TTL = 0$ será enviada ao final de todos os intervalos Θ (Regra 4), teremos então que o número médio de mensagens enviadas (e recebidas) por cada par por intervalo Θ será:

$$N_{msgs} = 1 + \sum_{l=2}^{\rho-1} P(l) \tag{4.7}$$

As Equações 4.1, 4.4, 4.5, 4.6, e 4.7 nos permitem calcular o tráfego médio de manutenção por par em um sistema D1HT, a partir do tamanho médio de sessão S_{avg}, do tamanho do sistema n, e da duração dos intervalos Θ.

4.6 Ajustando EDRA

Nesta seção iremos mostrar como o algoritmo de detecção e propagação de eventos usado por D1HT (EDRA) pode, automática e dinamicamente, se ajustar à variações no comportamento dos pares e do tamanho do sistema, de maneira a continuamente assegurar que a grande maioria das consultas (p.ex. 99%) será resolvida com um único salto. Em outras palavras, o objetivo será o de, independente do comportamento do sistema, garantir que a fração das consultas que não sejam resolvidas com um único salto seja inferior a um valor máximo aceitável f definido pelo usuário (ou projetista da aplicação), como por exemplo $f = 1\%$.

Como as consultas são resolvidas com apenas um salto, para satisfazer f é suficiente garantir que a probabilidade de cada salto falhar será no máximo f. Assumindo que os alvos das consultas são randomicamente espalhados ao longo do anel de pares (como em vários outros trabalhos, p.ex. [18, 23, 29, 42, 43, 53, 75, 96]), a fração de consultas resolvidas com mais de um salto será um resultado direto da fração de entradas inválidas nas tabelas de roteamento. Desta maneira, para satisfazer f basta assegurarmos que a fração média de entradas inválidas nas tabelas de roteamento dos diversos pares em um sistema D1HT seja mantida abaixo de f [29].

Como o tempo médio para um par tomar conhecimento de um evento é T_{avg}, o número médio de entradas inválidas nas tabelas de roteamento será dado pelo número de eventos

ocorridos nos últimos T_{avg} segundos, ou seja, $T_{avg} \cdot r$, onde r é a taxa de eventos no sistema (eventos por segundo). Isto implica que para satisfazer um valor de f pré-definido temos que satisfazer a inequação

$$\frac{T_{avg} \cdot r}{n} \leq f \tag{4.8}$$

Como, de acordo com a Equação 4.4, para minimizar os custos de manutenção devemos maximizar a duração dos intervalos Θ, a inequação acima e as Equações 4.5 e 4.3 nos garantem que o máximo valor de Θ para satisfazer um valor pré-definido de f será:

$$\Theta = \frac{2 \cdot f \cdot S_{avg} - 2 \cdot \rho \cdot \delta_{avg}}{8 + \rho} \tag{4.9}$$

Iremos agora assumir que $\delta_{avg} = 0,250$ segundos, o que é conservador em relação aos resultados apresentados em [85], aonde 80% das latências unidirecionais medidas para Gnutella foram abaixo de 140 ms (latências *round-trip* de 280 ms), sendo também conservador em relação às latências medidas para o conjunto King (77 ms) e para o PlanetLab (90 ms) [19]. Uma vez que os resultados publicados em [55] mostraram que Θ deverá ser sempre superior a 1 segundo, podemos então conservadoradamente assumir que $\delta_{avg} = \Theta/4$, o que nos permitirá simplificar a Equação 4.10 da seguinte maneira:

$$\Theta = \frac{4 \cdot f \cdot S_{avg}}{16 + 3 \cdot \rho} \tag{4.10}$$

É importante notar que a Equação 4.10 é conservadora mas só é válida para $\Theta \geq 1$ segundo, o que é verdade de acordo com os resultados apresentados em [55]. Como foi ressaltado na Seção 4.4, não é razoável assumir que a taxa de eventos r será constante, e a equação acima provê meios para que EDRA se adapte a mudanças no comportamento dos pares do sistema, uma vez que permite que cada par calcule dinamicamente a duração dos intervalos Θ a partir da taxa de eventos observada localmente (deve-se notar que S_{avg} pode ser facilmente calculada a partir de r com uso da Equação 4.5). Assim, a Equação 4.10 permite que um sistema D1HT se adapte dinâmica e automaticamente a mudanças no tamanho e comportamento do sistema, de maneira a continuamente garantir os menores custos de manutenção de tabelas de roteamento sem prejudicar o desempenho das consultas. Vale a pena frisar que esta equação não seria possível sem as propriedades formalmente provadas do Teorema 1.

Deve-se ressaltar que, com exceção de OneHop (que depende de um esquema hi-

erárquico para atingir este objetivo), todas as outras SHDHTs já propostas, incluindo 1h-Calot, não são capazes de determinar durações adequadas para os intervalos de agrupamento de eventos, nem mesmo de maneira estática ou para sistemas que não tenham variação de comportamento ou de tamanho, de modo que estes sistemas simplesmente não são capazes de agrupar eventos de maneira eficaz e ao mesmo tempo garantir suas propriedades de resolução de consultas com um único salto.

Como é assegurado pelo Corolário 1 que todos os pares em um sistema D1HT irão tomar conhecimento de qualquer evento em tempo proporcional ao logaritmo do tamanho do sistema, podemos considerar que as taxas de eventos observadas pelos diversos pares são similares e podem ser calculadas por cada par através da equação $r = E/\Theta$, onde E é o número de eventos conhecidos pelo par nos últimos Θ segundos. Assim, a partir das Equações 4.5 e 4.10 podemos calcular a quantidade máxima de eventos que um par pode tomar conhecimento durante um intervalo Θ:

$$E = \frac{8 \cdot f \cdot n}{16 + 3 \cdot \rho} \ eventos \tag{4.11}$$

Desta maneira, para satisfazer um valor de f previamente definido, cada par poderá simplesmente encerrar um intervalo Θ assim que tenha tomado conhecimento de E eventos (onde o valor de E é calculado segundo a equação acima), o que é um modo conveniente para implementação da adaptação de EDRA às mudanças de comportamento no sistema. É interessante notar que, segundo a Equação 4.11, a quantidade de eventos por intervalo Θ depende unicamente do tamanho do sistema e da fração máxima aceitável de falhas de roteamento.

Capítulo 5

Quarentena

Em qualquer sistema DHT a adesão de pares é custosa porque cada novo par deverá coletar informação a respeito das suas chaves, além dos endereços IP necessários para formação da sua tabela de roteamento, e estes custos podem ser inúteis se o novo par partir rapidamente do sistema. Este problema é agravado em SHDHTs, uma vez que a adesão (e a subsequente partida) do par volátil deverá ser notificada a todos os pares do sistema. Por outro lado, medições feitas em sistemas P2P reais [11, 85, 93] mostram que seus tamanhos de sessão usualmente têm distribuições estatísticas do tipo "cauda-pesada" (ou *heavy tailed*), o que significa que quanto maior for o tempo que um par estiver conectado ao sistema, maior será a probabilidade deste par continuar conectado.

De maneira a tomar proveito destas características observadas em sistemas P2P existentes e tratar o problema de pares voláteis, estamos nesta tese propondo um novo mecanismo de adesão gradativa, batizado de Quarentena. Segundo este mecanismo, novos pares não irão ser imediatamente inseridos na rede sobreposta, mas poderão realizar consultas a qualquer momento.

No mecanismo básico de adesão de D1HT (e outras DHTs), cada novo par p deverá, a partir de um conjunto $\mathbb{P}$ de outros pares do sistema (que poderá ser formado por um único par), obter as suas chaves bem como os endereços IP necessários para construir a sua tabela de roteamento. Com Quarentena, os pares em $\mathbb{P}$ vão simplesmente aguardar por um período de quarentena T_q (que pode ser fixo ou ajustado dinamicamente) antes de enviar as chaves e os endereços IP para p, adiando assim a sua inserção na rede sobreposta. Desta maneira, até que p receba as suas chaves e tabela de roteamento, a sua adesão não será divulgada e p não será responsável por nenhuma chave, mas, por outro lado, p já será capaz de realizar consultas. Para tanto, durante a sua quarentena, p deverá encaminhar suas consultas para um par próximo em termos de latência bi-direcional, escolhido entre os pares em $\mathbb{P}$, o qual será responsável por resolvê-las.

Deve-se notar que o mecanismo de Quarentena apresentado acima irá eliminar os custos de propagação de adesões e saídas de pares com durações de sessão menores que T_q. Por outro lado, as consultas feitas por pares novos (ou seja, pares que ainda estejam em quarentena) serão resolvidas através de dois saltos (o primeiro até um par próximo escolhido em $\mathbb{P}$, e o segundo a partir deste par próximo até o destino da consulta). Acreditamos que este custo adicional será aceitável por diversos motivos. Primeiro, por que o custo do salto adicional deverá ser pequeno, uma vez que este será encaminhado para um par próximo em termos de latência bi-direcional. Segundo, por que este custo adicional só será necessário durante uma parte inicial da vida de cada par (por exemplo, 5% da duração média de sessão). Terceiro, por que mesmo os pares voláteis terão benefícios, uma vez que será evitada a sobrecarga causada pela transferência inútil das chaves e endereços IP. Quarto, por que as reduções dos custos de manutenção deverão ser significativas e serão usufruídas por todo o sistema (como será comprovado pelos resultados que serão apresentados no Capítulo 8).

As reduções dos custos de manutenção de sistemas D1HT trazidas por Quarentena podem ser analiticamente quantificadas a partir da distribuição estatística das durações de sessão do sistema em questão e do período de quarentena escolhido. Assim, para um sistema D1HT com n pares e período de quarentena T_q, somente os q pares com duração de sessão maior que T_q serão efetivamente inseridos na rede sobreposta e terão seus eventos (adesão e partida) propagados para o sistema.

Assim, o efeito de Quarentena pode ser refletido na análise já feita através da substituição de n por q nas Equações 4.1 e 4.5, resultando em:

$$\rho_q = \lceil \log_2(q) \rceil \tag{5.1}$$

$$r_q = \frac{2 \cdot q}{S_{avg}} \tag{5.2}$$

Como os resultados das demais equações apresentadas no Capítulo 4 para análise quantitativa dos custos de manutenção de D1HT não dependem diretamente de n, estas equações continuam válidas com Quarentena. Deve-se ressaltar que o mecanismo de Quarentena que estamos introduzindo pode também ser usado por outros sistemas DHT (incluindo MHDHTs), mas os estudos analíticos apresentados nesta tese só são válidos para D1HT.

Adicionalmente às reduções dos custos associados a pares voláteis, o mecanismo de

Quarentena pode ser empregado para outras finalidades. Uma utilidade adicional deste mecanismo seria, por exemplo, aumentar a robustez do sistema em relação a ataques maliciosos. Para tanto, o período de quarentena seria dinamicamente ajustado de maneira que pares suspeitos teriam que aguardar por mais tempo para serem completamente aceitos no sistema, e o seu comportamento e identidade seriam observados durante o período de quarentena.

Um outro exemplo seria o uso de Quarentena para atenuar sobrecargas devidas a multidões repentinas (*flash crowds*), já que o período de quarentena poderia ser trivialmente elevado sempre que a taxa de adesões chegasse próxima de um limite que possa ser confortavelmente tratado pelo sistema.

Capítulo 6

Implementação de D1HT

Neste capítulo será apresentada e discutida a implementação de D1HT realizada como parte do trabalho desta tese. Iremos ressaltar os principais aspectos desta implementação, incluindo a nossa preocupação tanto com a sua correção, que nos levou a incluir extensos auxílios para depuração em seu código fonte, quanto com o seu desempenho e custos de manutenção, que nos levaram a implementar as consultas e mensagens para propagação de eventos diretamente sobre protocolo UDP. Ao final, vamos descrever detalhadamente os formatos usados para as mensagens de manutenção nesta implementação, que foram cuidadosamente desenhados de maneira a minimizar as demandas de rede, bem como a complexa estrutura de um processo D1HT, que é composto por diversos fluxos de execução assimétricos, de modo a lhe permitir o desempenho de diversas funcionalidades concorrentes de maneira eficiente.

6.1 Código Fonte

A implementação de D1HT foi completamente feita em C++, evitando-se o uso de Java ou outros pré-requisitos que dificultassem a sua instalação e uso em servidores de ambientes de computação de alto desempenho. A sua versão atual conta com cerca de oito mil linhas de código, apesar de ainda não termos implementado nem o mecanismo de Quarentena (apresentado no Capítulo 5) nem o uso de localidade (discutido na página 20 do Capítulo 3). Muitas desta linhas de código foram inseridas como medida de auxílio à depuração, só se tornando ativas quando este auxílio é explicitamente acionado durante a compilação de D1HT (o que é feito através da simples ativação da chave `-DDEBUG`). Desta maneira, a versão executável do processo D1HT para produção tem cerca de 500 KB de tamanho, enquanto a versão com auxílio de depuração é mais de três vezes maior, com 1,7 MB. Esta diferença ilustra bem o nosso cuidado e preocupação com a correção do código.

Deste modo, apesar do seu código fonte ter milhares de linhas, a versão de D1HT para produção é leve e eficiente, enquanto, ao mesmo tempo, dispomos de uma versão instrumentada que facilita o processo de depuração. Na verdade, a implementação produzida como uma das contribuições desta tese mostrou-se correta, robusta, leve e eficiente, uma vez que foi testada repetidamente em dois ambientes completamente diferentes com até 4.000 pares e 2.000 nós físicos distintos, resolvendo as consultas corretamente sem em nenhum momento interferir na produção normal destes ambientes. Ainda assim, como será mostrado no Capítulo 8, nossa implementação de D1HT foi superior a todas as outras soluções para resolução de consultas avaliadas (incluindo um servidor de diretórios), seja em termos de desempenho ou de escalabilidade ou de custos de manutenção. Além disto, D1HT foi capaz de resolver mais do que 99% das consultas com um único salto em todos os nossos experimentos.

De modo a dar mais robustez ao sistema, cada par em nossa implementação encerra o intervalo Θ corrente quando a sua duração Θ (calculada segundo a Equação 4.10 da página 35) expira, ou quando a quantidade de eventos acumulados E (calculada segundo a Equação 4.11 da página 36) é atingida, o que ocorrer primeiro.

O código fonte de D1HT está integralmente disponível de maneira livre e gratuita segundo licenciamento GPL a partir de [54], tendo sido testado em milhares de nós executando Linux. Não existem ainda versões para outros sistemas operacionais.

6.2 Protocolo para Execução de Consultas

De maneira a trabalharmos em direção à padronização das interfaces de chamadas de DHTs e outras soluções P2P, nossa implementação de D1HT procura seguir a API proposta em [20]. Foram inicialmente estudadas duas estratégias distintas para implementação desta API, sendo que em uma destas opções D1HT é implementado como um processo separado, como discutiremos a seguir.

Um Processo Único para D1HT e Aplicação: Nesta abordagem, todo o código objeto de D1HT deve ser ligado (*linked*) ao código da aplicação, seja de maneira estática ou dinâmica. Assim, todos os fluxos de execução de D1HT (que serão vistos em detalhe na Seção 6.6) compartilharão o mesmo espaço de endereçamento da aplicação. Esta é a opção mais simples de ser implementada, facilitando também a execução da aplicação, mas requer compatibilidade integral entre os códigos e ambientes da aplicação e de D1HT. Por exemplo, poderíamos ter problemas para adaptar aplicações HPC escritas em versões legadas de FORTRAN para que façam

consultas através da nossa implementação de D1HT escrita em C++ com múltiplos fluxos POSIX. Além disto, não seria possível que aplicações executando em outros sistemas operacionais (como, por exemplo, Windows, AIX, Solaris, etc.) fizessem consultas à nossa implementação atual de D1HT, que é compatível apenas com Linux, requerendo então o desenvolvimento de uma versão completa de D1HT para cada sistema operacional.

Processos Separados para D1HT e Aplicação: Neste caso, existe um processo separado para execução de D1HT, sendo que as consultas são realizadas enviando-se mensagens de rede para um endereço IP pré-definido deste processo. Como ambos os processos estarão, tipicamente, sendo executados no mesmo computador, estas mensagens deverão ter latência insignificante em relação ao salto que será necessário para resolução de cada consulta. A criação, envio e tratamento destas mensagens são transparentes para a aplicação, uma vez que estas tarefas são feitas pelas rotinas da API de D1HT. Como uma primeira vantagem desta abordagem, temos que um mesmo processo D1HT pode ser compartilhado por diversos processos da aplicação. Mais importante, esta estratégia evita que tenhamos que ligar (*link*) todo o código objeto de D1HT com a aplicação, de modo que eventuais questões de interoperabilidade e compatibilidade entre diferentes linguagens e sistemas operacionais possam ser mais facilmente contornadas, bastando criar versões das rotinas da API de D1HT (que são poucas e relativamente simples) para as diversas linguagens e sistemas operacionais que executam as aplicações. Por exemplo, poderíamos ter uma aplicação escrita em FORTRAN ou .NET executando em uma máquina virtual Windows Vista, realizando consultas através de um processo D1HT que está sendo executado em uma máquina virtual Linux residente no mesmo (ou em outro) nó físico. Deve-se ressaltar, no entanto, que no caso de nós fisicamente distintos, deveremos ter latências relevantes para as trocas de mensagens de consulta entre os nós físicos que hospedam os processos da aplicação e de D1HT.

Obviamente, o ideal seria desenvolver D1HT de maneira a suportar as duas estratégias discutidas acima, e este continua sendo nosso objetivo, mas julgamos também que seria muito custoso desenvolvê-las conjuntamente. Assim, de maneira a minimizar a possibilidade de eventuais problemas de compatibilidade e interoperabilidade com os ambientes que teríamos disponíveis para realização de experimentos, optamos por inicialmente implementar D1HT como um processo estanque, o que já foi feito, ficando como trabalho futuro o desenvolvimento da outra opção discutida acima.

Deve-se ainda notar que, como as duas opções discutidas acima implementam exatamente a mesma API, a migração de uma aplicação de uma estratégia para outra é simples, sendo desnecessária qualquer alteração no seu código fonte. Na verdade, quando as duas estratégias estiverem implementadas, pretendemos disponibilizá-las através de ligação dinâmica de módulos, de maneira que caso a aplicação esteja sendo executada em um ambiente Linux a estratégia de um único processo seja selecionada, usando-se dois processos nos outros casos.

6.3 Protocolos de Comunicação

De maneira a minimizar as necessidades em termos de pré-requisitos de *software* e buscar o melhor desempenho para as consultas e propagação de eventos, todas as tarefas de comunicação foram implementadas diretamente com uso de soquetes IP.

Deve-se salientar que em D1HT todas as mensagens devem ter suas entregas confirmadas, o que pode ser feito implicitamente com uso de um protocolo de transporte como TCP, ou explicitamente com uso de UDP. O uso do protocolo TCP é muito mais confortável, além de ser mais robusto em relação a perdas de pacotes e retransmissões, mas a instauração de uma conexão TCP entre dois pares requer a troca de três mensagens (procedimento conhecido como *three-way handshake* [95]), de maneira que este custo inicial só será compensado caso a conexão seja usada para a troca de diversas mensagens.

Assim, escolhemos usar o protocolo UDP, com controle de retransmissões explicitamente implementado em D1HT, para dois tipos de mensagens, consultas e propagação de eventos, já que para estas, em geral, cada conexão TCP seria usada para transmissão de uma única mensagem (além de sua resposta ou confirmação de entrega). Os demais tipos de mensagens foram implementados sobre TCP, ou por que cada conexão é usada para transmitir várias mensagens (como, por exemplo, a transferência de uma tabela de roteamento), ou por que são usadas em procedimentos de controle que precisam de maior robustez (como, por exemplo, a rotina de estabilização de ponteiros para o sucessor e o predecessor de um par).

6.4 Protocolo de Adesão de Novos Pares

Para a nossa implementação optamos por um protocolo de adesão inspirado no empregado em Chord [96], com algumas importantes modificações para que sejam atendidos os requisitos discutidos no Capítulo 3. Mais especificamente, a adesão de um par é sempre tratada por seu sucessor, que lhe transfere toda a tabela de roteamento e então inicia a pro-

pagação da sua adesão segundo EDRA. Para assegurar que o novo par irá receber todas as notificações de eventos enquanto a sua adesão não for conhecida por todo o sistema, o seu sucessor irá lhe enviar todos os eventos que receba até que o novo par receba ao menos uma mensagem de manutenção com cada TTL.

6.5 Formato das Mensagens de Manutenção

Uma vez que é nosso objetivo a redução das demandas de banda passante de manutenção, torna-se essencial a otimização do formato das mensagens de modo a minimizar o seu tamanho. Para tanto, devemos nos preocupar não só com os tamanhos dos cabeçalhos das mensagens, que estão associados aos parâmetros v_a e v_m da Equação 4.4 (apresentada na página 32), como também com quantidade de informação necessária para descrever cada evento, que forma a parte variável das mensagens e é quantificada segundo o parâmetro b da mesma equação.

Uma vez que a nossa implementação atual não faz uso de localidade, e como em D1HT o identificador de cada par pode ser obtido a partir de seu endereço IP, para cada evento é suficiente transmitir apenas o endereço do respectivo par, além da indicação do tipo de evento ocorrido (adesão ou partida). Para tanto, basta incluirmos contadores com as quantidades de adesões e partidas no cabeçalho da mensagem, o qual deverá ser seguido pelos endereços IPv4 dos diversos pares que sofreram estes eventos. Uma vez que não é razoável assumir um número de porta fixo para todos os pares, devemos nos preocupar também em propagar o número da porta usada por cada par, o que nos impede de assumir que cada evento possa sempre ser descrito por apenas quatro octetos. Por outro lado, consideramos que, assim como em diversas aplicações P2P populares (como, por exemplo, KAD e Gnutella), podemos definir uma porta padrão, que provavelmente será adotada pela maioria dos pares, sem impedir que qualquer par possa escolher uma porta alternativa caso isto seja de sua necessidade ou conveniência. Assim, poderemos nos aproximar da média de quatro octetos por descrição de evento para o parâmetro b da Equação 4.4, mas precisaremos de quatro contadores no cabeçalho das mensagens de manutenção, sendo dois contadores para as quantidades de adesões e partidas de pares que usam a porta padrão (e que serão descritas por quatro octetos), e os outros dois contadores para as eventuais adesões e partidas de pares que usam outras portas (e que portanto necessitam de seis octetos para serem descritas).

Assim, como pode ser visto da Figura 6.1, em nossa implementação de D1HT cada mensagem contém um cabeçalho com os seguintes campos:

Tipo da Mensagem (*Type*): Tamanho de um octeto. Para mensagens de manutenção o valor deste campo deverá ser menor que 128 e corresponderá ao seu TTL. Para cada outro tipo de mensagem este campo terá um valor pré-definido, maior que 128. O valor 128 é reservado.

Número Sequencial (*SeqNo*): Tamanho de um octeto. Contador usado para permitir o controle das confirmações de recebimento e retransmissões das mensagens sobre protocolo UDP (ou seja, para mensagens de consulta e propagação de eventos). Para mensagens sobre protocolo TCP este campo não é usado.

Número de Porta IP (*PortNo*): Tamanho de dois octetos. Uma vez que as diversas mensagens são transmitidas com uso de soquetes temporários com números de porta efêmeros, neste campo o transmissor da mensagem deve armazenar o número da porta do endereço IP pelo qual é conhecido.

Identificador do Sistema D1HT Corrente (*SystemID*): Tamanho de quatro octetos. Uma vez que podemos ter diversos sistemas D1HT disjuntos executando simultaneamente, o valor deste campo é usado para impedir que pares de diferentes sistemas se mesclem inadvertidamente. O valor deste campo é definido pelo primeiro par de um sistema D1HT, sendo tipicamente o seu ID, mas pode também ter o seu valor arbitrariamente definido por este par.

Contadores de Eventos: Quatro contadores, cada um com tamanho de um octeto. Os valores destes campos refletem as quantidades de eventos notificados através da mensagem, sendo:

Join-def: Quantidade de adesões de pares que usam a porta padrão.

Join-other: Quantidade de adesões de pares que não usam a porta padrão.

Leave-def: Quantidade de partidas de pares que usam a porta padrão.

Leave-other: Quantidade de partidas de pares que não usam a porta padrão.

O cabeçalho de mensagem acima descrito (e ilustrado na Figura 6.1), é seguido pelos endereços IP dos pares que aderiram e partiram do sistema, sendo que só são incluídos os números de porta daqueles pares que não usam a porta padrão. Já os cabeçalhos das mensagens de confirmação de entrega (*acks*) e das respostas a consultas só incluem os quatro primeiros campos listados acima (i.e., *Type*, *SeqNo*, *PortNo* e *SystemID*).

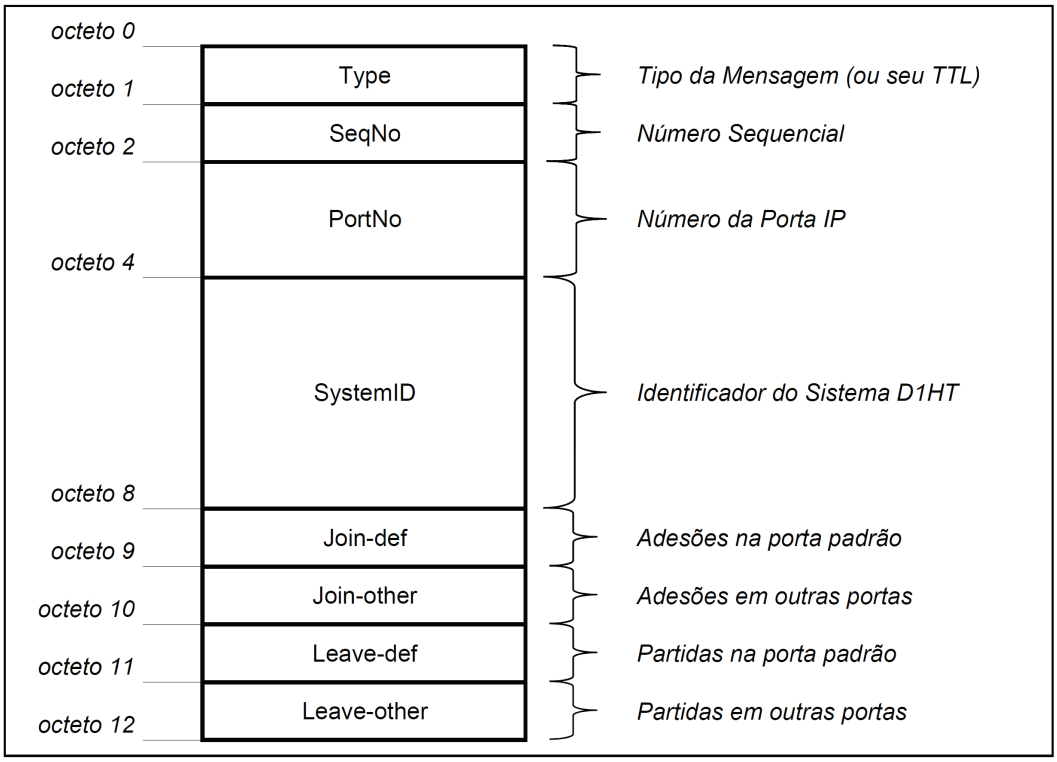


Figura 6.1: Cabeçalho das mensagens em nossa implementação de D1HT, que têm tamanho fixo de 12 octetos. Além deste, cada mensagem incluirá os necessários cabeçalhos dos protocolos de rede utilizados. No caso das mensagens de manutenção na versão atual da implementação, estes protocolos são IPv4 (cabeçalho com tamanho de 20 octetos) e UDP (cabeçalho com tamanho de 8 octetos).

Devemos ainda ter em conta que cada mensagem deverá incluir os cabeçalhos dos protocolos de rede e transporte utilizados. Uma vez que para as mensagens de manutenção usamos o protocolo UDP, que tem cabeçalhos com 28 octetos (oito octetos para o próprio protocolo UDP, além de 20 octetos para o protocolo IPv4), para a nossa implementação do sistema D1HT os fatores v_m e v_a da Equação 4.4 terão valores de 40 e 36 octetos, respectivamente, de maneira a contabilizar os cabeçalhos D1HT, UDP e IPv4.

É importante ressaltar que a maioria das simulações e avaliações analíticas de sistemas DHT publicadas não computam adequadamente os custos dos cabeçalhos das mensagens (por exemplo, [23, 29, 42, 43, 44]), uma vez que consideram um custo fixo máximo de 20 octetos por mensagem para contabilizar todos os cabeçalhos de protocolos de rede e da própria DHT, o que não reflete de maneira precisa a realidade, já que só os protocolos de rede e transporte requerem ao menos 28 octetos por mensagem. Devemos então frisar que os resultados que serão apresentados no Capítulo 8 são mais realistas do que os que são normalmente encontrados na literatura, uma vez que os nossos resultados incluem os verdadeiros custos dos cabeçalhos das mensagens de uma implementação real (incluindo

os tamanhos dos necessários cabeçalhos dos protocolos de rede). Por outro lado, deve-se também notar que os dimensionamentos feitos nesta seção são dependentes dos protocolos de rede e transporte utilizados (no caso, IPv4 e UDP), e portando devem ser revistos caso outros protocolos sejam escolhidos (por exemplo, IPv6 e/ou TCP).

6.6 Nível de Concorrência

Uma vez que qualquer par D1HT deve ser capaz de desempenhar diversas funcionalidades simultaneamente, é importante que a sua implementação permita que suas funções possam ser executadas concorrentemente de maneira a evitar a serialização de tarefas, e consequentes problemas de desempenho. Por exemplo, um par poderá ao mesmo tempo ser solicitado a receber uma mensagem de manutenção, realizar duas consultas e tratar um pedido de adesão, e a execução destas tarefas não poderá afetar a propagação das mensagens de manutenção. Caso estas ações sejam serializadas o seu desempenho poderá ser comprometido.

Como as funções inerentes aos pares D1HT são, em geral, rotineiras e de baixa complexidade, julgamos que seria muito custosa a criação dinâmica de um processo para a execução de cada uma destas ações, sendo portanto mais atrativo o uso de fluxos leves de execução (*threads*), sempre seguindo o padrão POSIX [67]. Desta maneira, D1HT foi implementado na forma de um único processo a ser executado por cada par, sendo este processo composto por alguns fluxos perenes, além de outros fluxos que são criados e finalizados dinamicamente para o desempenho de algumas funções. Em virtude da nossa proposta ser puramente P2P, todos os pares executam o mesmo código e portanto têm exatamente as mesmas funcionalidades.

Na Figura 6.2 ilustramos a estrutura de um processo D1HT, mostrando o conjunto básico de fluxos necessários para implementação das funções de consulta, manutenção das tabelas de roteamento e tratamento de adesões e partidas. A seguir descreveremos estes diversos fluxos, indicando através de letras aqueles que recebem (`R`) e transmitem (`T`) mensagens (os que apenas respondem a mensagens recebidas estão indicados com `t`), bem como os que atualizam a tabela de roteamento (`A`) e criam fluxos de execução adicionais (`F`). Os fluxos que transmitem mensagens são também responsáveis pelo recebimento e tratamento das respectivas confirmações de entrega (i.e., *acks*) ou respostas. Deve-se notar que qualquer acesso à tabela de roteamento deve ser protegido por seções críticas de maneira a garantir a sua integridade.

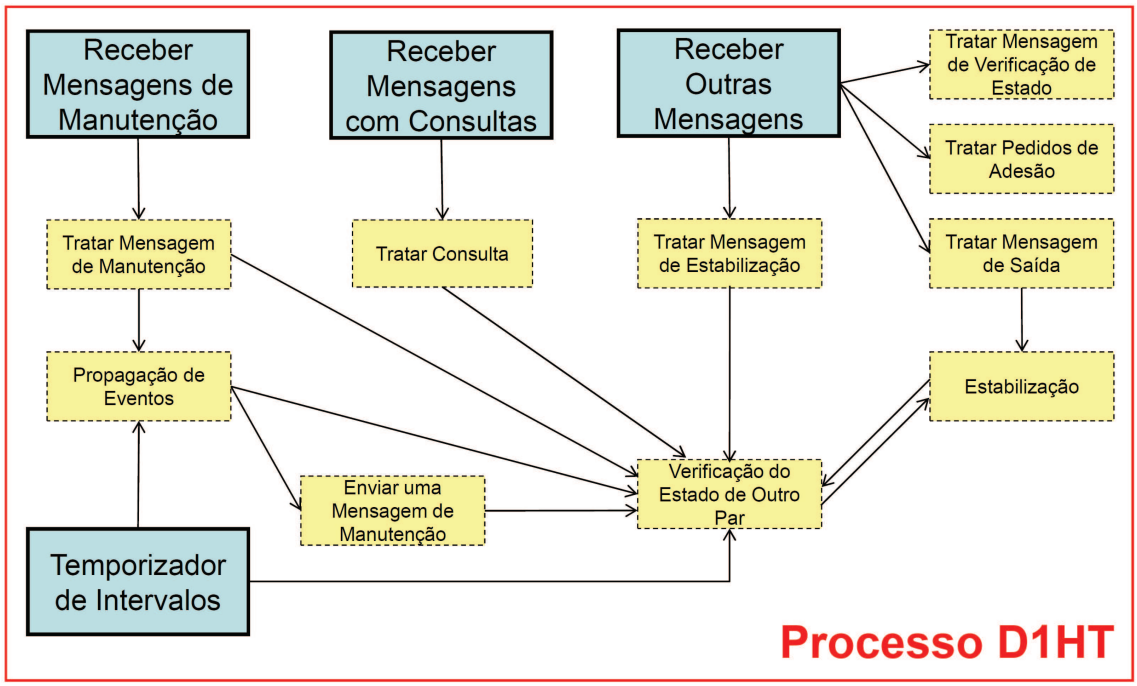


Figura 6.2: Diagrama de um processo D1HT, onde cada caixa com bordas sólidas representa um fluxo (i.e., *thread*) perene (por exemplo, *Receber Mensagens de Manutenção*), e as caixas com bordas tracejadas representam fluxos de execução que são criados e removidos dinamicamente. As setas indicam a relação entre os diversos fluxos, de maneira que uma seta partindo de um fluxo até outro indica que o segundo fluxo é (ou pode ser) criado pelo primeiro.

Receber Mensagens (`RF`): Estes três fluxos são responsáveis por receber todas as mensagens enviadas ao par (a menos das mensagens de confirmação de entrega e respostas), sendo que são descartadas todas as mensagens com *SystemID* diferente daquele do sistema D1HT corrente. De maneira a evitar contenções nestes fluxos, para cada mensagem recebida é criado um novo fluxo para o seu tratamento. Pelo mesmo motivo, estes três fluxos não entram em qualquer seção crítica e portanto não acessam a tabela de roteamento. Desta maneira, a menos das mensagens de saída, o fluxo que for criado para o tratamento de uma mensagem recebida deverá também adicionar o par que a enviou na tabela de roteamento local, em conformidade com o discutido na Seção 4.3.

Tratar Mensagem de Manutenção (`AF`): Atualiza a tabela de roteamento com os eventos recebidos e, para mensagens com $TTL > 0$, os inclui nas listas de eventos a transmitir ao final do intervalo Θ corrente. Além disto, este fluxo é responsável por atualizar o número de eventos recebidos no intervalo Θ corrente (E) e, caso este valor exceda o definido pela Equação 4.11, cria um novo fluxo para Propagação de Eventos, mesmo que a duração do intervalo Θ corrente ainda não tenha expi-

rado. Caso seja recebida uma mensagem com $TTL = 0$ de outro par que não seja o predecessor, inicia um fluxo para verificação do estado deste.

Tratar Consulta (`tA`): Verifica se a consulta pode ser satisfeita localmente (i.e., se o par que recebe a consulta é o sucessor do ID buscado) e, caso positivo, responde esta consulta afirmativamente. Caso contrário, a mensagem é respondida com os endereços IP dos dois sucessores imediatos do ID consultado, segundo a tabela de roteamento local. Caso a consulta devesse ter sido encaminhada ao predecessor, inicia um fluxo para verificação do estado deste.

Tratar Pedido de Adesão (`tA`): Caso não seja o sucessor deste novo par, encaminha o pedido de adesão para o seu sucessor, segundo a tabela de roteamento local. Caso contrário, providencia a inserção do novo par no anel e lhe transmite a tabela de roteamento e as chaves que serão de sua responsabilidade. Finalmente, inicia a propagação deste evento.

Tratar Mensagem de Saída (`AF`): Atualiza a tabela de roteamento, e, caso se trate da saída do seu predecessor, a inclui na lista de eventos a transmitir com $TTL = \rho$ e, em seguida, inicia um fluxo de Estabilização.

Temporizador de Intervalos Θ (`F`): A cada Θ segundos cria um novo fluxo Propagação de Eventos.

Propagação de Eventos (`F`): Reinicializa o relógio temporizador de intervalos Θ, e cria um fluxo separado para o envio de cada mensagem de manutenção. Deve-se notar que, segundo a Regra 4 de EDRA, a mensagem com $TTL = 0$ será enviada mesmo que não haja nenhum evento a propagar. Este processo é também o responsável por verificar se foi recebida alguma mensagem do predecessor nos últimos Θ segundos e, caso contrário, inicia um fluxo para verificação do seu estado.

Enviar Uma Mensagem de Manutenção (`TF`): Envia uma mensagem de manutenção (com um determinado TTL) e aguarda a sua confirmação de recebimento. Caso esta confirmação não seja recebida dentro de um determinado intervalo de tempo (cuja duração é dinamicamente ajustada), retransmite a mensagem de manutenção para o mesmo destino. Depois de três tentativas de transmissão sem sucesso, inicia um fluxo de execução separado para verificação do estado deste destino, e escolhe como novo destino o seu sucessor. Repete este procedimento sucessivamente até conseguir confirmar a entrega da mensagem de manutenção.

Verificação do Estado de Outro Par (*Probe*) (`TA`): Envia uma mensagem de verificação de estado para o par e aguarda a sua confirmação de recebimento, até o máximo de três tentativas. Remove ou insere o par na tabela de roteamento local no caso de, respectivamente, insucesso ou sucesso. Caso seja detectada a ausência do predecessor, inicia um fluxo de Estabilização.

Tratar Mensagem de Verificação de Estado (`tA`): Envia mensagem resposta de confirmação de recebimento, atestando a sua presença no sistema.

Estabilização (`TAF`): No caso de adesão ou saída do predecessor, inicia procedimento de estabilização com os dois mais próximos sucessores e predecessores. Caso seja detectada alguma incoerência entre as informações da tabela de roteamento local e as de seus vizinhos, inicia um ou mais fluxos de Verificação do Estado de Outro Par.

Tratar Mensagens de Estabilização (`tAF`): Atualiza a tabela de roteamento com as informações recebidas, e envia mensagem resposta com os endereços dos seus dois mais próximos sucessores e predecessores. Caso seja detectada alguma incoerência entre as informações recebidas e a tabela de roteamento local, inicia um ou mais fluxos de Verificação do Estado de Outro Par.

O conjunto básico de fluxos brevemente descrito acima visa simplesmente a manutenção das tabelas de roteamento (incluindo adesões e partidas) e a resolução de consultas, sendo que para algumas aplicações poderão ser necessárias funções adicionais.

Apesar da estratégia proposta por D1HT ser simples, especialmente se o compararmos com sistemas hierárquicos como OneHop, a Figura 6.2 ilustra o nível de complexidade que uma implementação robusta e eficiente pode atingir, não só para ser desenvolvida como também para ser depurada. Esta figura mostra que cada processo D1HT poder ter até quatorze fluxos com funções distintas sendo executados simultaneamente. Além disto, vários destes fluxos (como, por exemplo, Verificação do Estado de Outro Par, Tratar Consultas, Tratar Mensagem de Manutenção e Enviar Uma Mensagem de Manutenção) podem ter diversas instâncias sendo executadas concorrentemente. Considerando-se ainda que poderemos ter milhares (ou, talvez, milhões) de processos D1HT sendo executados simultaneamente, a cuidadosa depuração deste sistema torna-se por um lado altamente trabalhosa, e por outro absolutamente necessária, já que pequenos erros que ocorram ocasionalmente podem ter graves consequencias para o desempenho, correção e/ou robustez do sistema, especialmente quando usado em ambientes de larga escala.

Deve-se ainda notar que as tarefas de desenvolvimento e depuração se tornam ainda mais complexas pelo fato de serem simplesmente opostas as características do paralelismo

encontrado *intra* e *inter* processos D1HT. Isto por que o paralelismo existente *dentro* de cada processo D1HT é heterogêneo, com comunicação através de memória compartilhada e sincronização com POSIX `locks`. Em franco contraste, o paralelismo *entre* processos D1HT é homogêneo, de caráter distribuído, e tem comunicação e sincronização feitas através de trocas mensagens implementadas em soquetes IP com uso de dois protocolos de transporte diferentes (TCP e UDP), sendo que as mensagens transmitidas sobre UDP têm controle de perda de pacotes e retransmissões implementado explicitamente em D1HT.

Capítulo 7

D1HF: Um sistema de arquivos baseado em D1HT

Neste capítulo apresentaremos D1HF, um novo sistema de arquivos puramente P2P baseado em D1HT, que é completamente aderente ao padrão POSIX [67] e foi introduzido e desenvolvido no trabalho desta tese. Mesmo que o objetivo principal de D1HF no contexto desta tese seja o de validar D1HT como uma ferramenta útil para construção de sistemas distribuídos, acreditamos que este sistema de arquivos possa também ser considerado como uma das contribuições desta tese.

Iniciaremos com discussões sobre aspectos presentes no contexto atual que nos levaram a projetar e desenvolver D1HF, que incluem aplicações chaves que apresentam padrões de acesso de leituras repetitivas a arquivos, e os multicomputadores Beowulf, que atualmente são o padrão de fato em termos de plataforma de computação para HPC. Em seguida, traçaremos os objetivos de D1HF frente a este contexto, e apresentaremos uma visão geral deste inovador sistema de arquivos, que faz uso de D1HT para garantir seu desempenho, seu balanceamento de carga, e sua arquitetura puramente P2P e auto-reorganizável. Nas quatro seções seguintes iremos discutir a sua atuação nas primitivas POSIX de `open()`, `read()`, `lseek()` e `close()`, respectivamente. Discutiremos então como D1HF se auto-reorganiza na presença de adesões e partidas de pares, e as eventuais perdas de desempenho decorrentes destes eventos. Nas duas últimas seções deste capítulo iremos discutir o modelo de consistência de dados de D1HF e apresentaremos a sua implementação preliminar.

7.1 Acessos Repetitivos de Leitura a Arquivos de Dados

D1HF foi projetado a partir da observação de que, assim como para os acessos à memória principal, existem aplicações que apresentam uma forte localidade espacial no acesso a arquivos (ou fluxos) de dados, em especial para leitura, sendo que muitas vezes esta localidade se manifesta na forma de diversas instâncias (i.e., processos) de uma mesma aplicação distribuída acessando o mesmo conjunto de dados. Mais especificamente, ao longo do trabalho desta tese foram estudados os padrões de acesso de algumas aplicações paralelas largamente utilizadas para imageamento sísmico da sub-superfície terrestre [62], que utilizam a maior parte da capacidade dos mais de 10 mil processadores de um ambiente HPC corporativo de produção [63].

As nossas observações indicaram que o perfil de acesso aos dados demonstrado pelas aplicações estudadas é fortemente dominado por atividades de leitura, onde as mesmas porções dos seus arquivos de entrada são lidas por centenas, ou mesmo milhares, de processos paralelos distintos. Além disto, as aplicações estudadas têm carga de comunicação entre processos (IPC) reduzida e pequena quantidade de dados de saída (se comparada com a quantidade de dados de entrada), de maneira que a carga de comunicação é amplamente dominada pela leitura dos seus arquivos de entrada.

Abaixo descrevemos algumas características dos dois padrões de acesso predominantes das aplicações paralelas de imageamento sísmico estudadas.

Acesso de Leitura Sequencial: Nas aplicações que apresentam este padrão de acesso, todos os seus processos lêem dados de entrada de um mesmo arquivo, que pode ter tamanho com até dezenas de bilhões de octetos. Ao longo da sua execução, cada processo perfaz iterações em que o ponteiro de leitura é posicionado em um ponto do arquivo (através da execução de uma operação POSIX `lseek()`), a partir do qual é lida uma porção contígua deste de modo completamente sequencial. Na próxima iteração o ponteiro de leitura é reposicionado e uma outra porção do mesmo arquivo é lida, e assim repetidamente, sendo que cada processo pode ler a mesma porção do arquivo várias vezes. Como resultado da execução de diversas destas tarefas por cada processo da aplicação paralela, cada porção do arquivo de entrada é lida por diversos processos distintos. Deve-se notar que este padrão tem um perfil de acesso regular, que permite significativa otimização de desempenho com uso de pré-carga (*prefetch*) de dados. Em geral, o fator limitante do desempenho deste tipo de acesso é a capacidade de banda passante do sistema de arquivos (incluindo subsistemas de discos), ou da rede de interconexão.

Acesso de Leitura Salteado: Para este padrão de acesso, os dados são distribuídos por

dezenas de milhares de arquivos distintos, todos residindo no mesmo sistema de arquivos, com cada arquivo contendo algumas dezenas de milhões de octetos. Neste padrão, cada processo da aplicação paralela perfaz múltiplas iterações, sendo que em cada iteração são lidos alguns arquivos de entrada distintos. Para cada arquivo aberto a tarefa define um tamanho de salto (*stride*), que ficará fixo enquanto o arquivo estiver aberto. Assim, entre duas operações de leitura sempre haverá um salto, ou seja, um reposicionamento do ponteiro de leitura através da execução de uma operação POSIX `lseek()`. Tipicamente, cada arquivo de entrada será lido diversas vezes por diferentes processos, sendo que a cada vez que um arquivo seja aberto poderá ser usado um tamanho de salto distinto. Em geral este padrão de acesso apresenta demandas de banda passante inferiores ao anterior, uma vez que o fator limitante de seu desempenho normalmente é a taxa de operações de leitura por segundo suportada pelo sistema de arquivos. Este padrão salteado difere do padrão aleatório clássico, no qual o tamanho do salto varia constantemente, mas ambos devem apresentar cargas similares ao sistema de arquivos, a menos do fato do padrão salteado que estudamos ser suscetível de otimização com uso de pré-carga (*prefetch*) de dados.

Devemos ressaltar que as descrições apresentadas acima não procuram refletir padrões de acesso sequenciais e salteados genéricos, e sim aqueles que foram observados nas aplicações paralelas que foram monitoradas.

Para ambos os padrões de acesso, cada porção dos arquivos de entrada é tipicamente lida por até centenas (ou mesmo milhares) de processos distintos, implicando em altos *níveis de releitura* de dados (como será definido pela Equação 7.1 abaixo). Assim, as demandas agregadas de banda passante de leitura destas aplicações poderão atingir níveis próximos a 100 Gbps. Este nível de carga de leitura de dados é extremamente desafiador para soluções baseadas em sistemas de arquivos de rede que sejam abertos, padronizados e largamente aceitos (i.e., NFS), e mesmo sistemas de arquivos paralelos podem ter que ser levados a dimensões que são difíceis de gerenciar. Por exemplo, o ambiente HPC estudado faz hoje uso de uma complexa solução baseada em GPFS [88], com subsistemas de discos de alto desempenho e mais de 50 servidores multiprocessados dedicados. Além de onerosa, esta solução requer gerenciamento e ajuste cuidadosos e especiais.

Uma vez que as *releituras* a dados de entrada representam uma característica fundamental dos padrões de acesso que pretendemos endereçar, vamos abaixo introduzir uma equação de maneira a não só definir precisamente como também quantificar o que nesta tese chamamos de *nível de releitura* (ou simplesmente *R*):

$$R = \frac{L}{B} - 1 \tag{7.1}$$

onde L é o quantidade de octetos lidos pela aplicação, e B é a quantidade de octetos *distintos* lidos. Aqui chamamos de *distintos* aqueles octetos que estão em arquivos diferentes, ou em diferentes posições de um mesmo arquivo, mesmo que tenham o mesmo conteúdo (valor). Deve-se ressaltar que enquanto o nível de releitura é fortemente dependente do padrão de acesso característico de uma aplicação, R pode variar também segundo outros fatores, em especial a quantidade de instâncias simultâneas da própria aplicação. Por exemplo, o nível de releitura de uma aplicação distribuída tende a crescer com a quantidade de processos que são executados simultaneamente.

Apesar da definição de R, como apresentada na equação 7.1, quantificar a carga de releitura em termos de banda passante, como normalmente é feito para acessos sequenciais, ela é também representativa em termos de quantidade de operações de leitura quando feitas com tamanho fixo.

Na Seção 7.4 apresentaremos o sistema de arquivos D1HF, que procura otimizar padrões de acesso com altos níveis de releitura, como os discutidos acima. Devemos no entanto salientar que padrões similares estão presentes em outras aplicações largamente utilizadas, algumas radicalmente distintas das encontradas em ambientes HPC, como Vídeo sob Demanda, *media streaming*, ou mesmo acessos HTTP a páginas populares na Internet. Desta maneira, acreditamos que as estratégias utilizadas por D1HF podem endereçar uma fração significativa dos acessos a dados presentes tanto em ambientes de computação de alto desempenho quanto na Internet.

7.2 Multicomputadores em Ambientes HPC

A computação de alto desempenho tem sido recentemente dominada por multicomputadores da classe Beowulf [94], que podemos classificar como aglomerados de computadores idênticos, construídos com componentes que são padrão de mercado (i.e., *commodities*), que são utilizados para processamento científico paralelo. Muitos destes multicomputadores são construídos com nós computacionais que possuem discos locais dedicados e conexões de rede (tipicamente Ethernet) formando uma topologia em árvore gorda, de maneira similar à arquitetura dos multicomputadores usados por ISPs [5].

Os discos locais aos diversos nós têm, em geral, custo e desempenho reduzidos, sendo normalmente utilizados para alojar sistema operacional, arquivos de paginação e arquivos de trabalhos temporários. Tipicamente, estes discos ficam a maior parte do tempo ociosos,

e não comprometem o desempenho da solução como um todo.

A topologia em árvore gorda dos multicomputadores Beowulf pode ser implementada com diferentes tecnologias de rede, incluindo Infiniband, mas é normalmente construída com uso de uma ou duas interfaces Gigabit Ethernet 1000baseT integradas às placas mãe dos nós computacionais, que são conectadas a comutadores Ethernet não bloqueantes com 24 ou 48 interfaces 1000baseT. Estes comutadores, em alguns casos, podem dispor também de duas interfaces 10GbE (usualmente no padrão 10GBASE-SX).

Os nós do multicomputador são então distribuídos por diversos bastidores, instalando-se pelo menos um dos citados comutadores Ethernet em cada bastidor, que são portando comumente chamados de comutadores *TOR* (acrônimo do termo em inglês *Top Of Rack*). Cada nó computacional tem então uma ou duas conexões 1000baseT a um comutador TOR, que por sua vez tem uma conexão com capacidade entre 2 Gbps e 20 Gbps ao nível imediatamente superior da árvore gorda.

A relação entre a quantidade de conexões dos nós aos comutadores TOR e a capacidade agregada das conexões ao nível superior da árvore gorda representa o *nível de contenção* (ou *nível de bloqueio*) da árvore (ou do multicomputador). Por exemplo, um multicomputador com 40 nós conectados a cada comutador TOR (uma conexão 1000baseT por nó), que por sua vez tenham conexões de 20 Gbps (2 x 10GBASE-SX) ao nível superior da árvore, apresenta nível de contenção 2:1.

A solução em árvore gorda com padrão Ethernet tem construção simples, padronizada e de baixo custo, uma vez que utiliza apenas comutadores comuns e interfaces integradas. Além disto, esta topologia é adequada para aplicações com carga de comunicação dominada pela entrada e saída de dados, ou que têm forte localidade de comunicação entre processos (de maneira que a maior parte desta carga IPC fique interna aos comutadores TOR não bloqueantes), desde que as limitações de latência e banda passante inerentes a redes Ethernet sejam adequadas para a aplicação e não restrinjam o seu desempenho.

Em relação às aplicações com o perfil discutido na Seção 7.1, que são de nosso especial interesse, a topologia em árvore gorda é particularmente apropriada, uma vez que as demandas de comunicação de cada nó não são em geral desafiadoras, mas a soma das demandas dos diversos nós pode atingir níveis elevados. Desta maneira, o nível de contenção da árvore pode, e deve, ser dosado de acordo com o perfil de cada aplicação, procurando-se um ponto conservador para multicomputadores usados por aplicações com perfis distintos. Por exemplo, em um multicomputador Beowulf que é usado por uma única aplicação com demanda estável de leitura de dados de 200 Mbps para cada um de seus mil nós, as diversas conexões 1000baseT individuais ficam a maior parte do tempo ociosas, de modo a permitir o uso de altos níveis de contenção (por exemplo, 4:1). Por

outro lado, a demanda agregada de leitura de dados será 200 Gbps, que requer sistemas de arquivos e soluções de armazenamento especiais, em geral complexas e proprietárias. É importante observar que, mesmo nos casos de acesso puramente sequencial, a demanda agregada de 200 Gbps seria desafiadora mesmo para sistemas de discos complexos e dedicados, enquanto a demanda de 200 Mbps individual a cada nó poderia ser atendida por discos comuns.

Deve-se ainda notar que, uma vez que na maioria das vezes são utilizados comutadores TOR comuns, que representam uma fração do custo total da solução, os níveis de contenção comumente implantados nos multicomputadores são conservadores, procurando-se assim dar-lhes flexibilidade suficiente para o processamento de aplicações com perfis distintos, além de potencialmente preservar a sua vida útil frente a possíveis aumentos de cargas de rede que possam ocorrer ao longo dos anos. Além disto, para o perfil de acesso das aplicações estudadas, as diversas conexões de comutadores e nós de processamento têm que ser dimensionadas para a carga de leitura (i.e., de *descida*) de dados e, uma vez que as tecnologias utilizadas são invariavelmente *full-duplex*, sua capacidade de transmissão no sentido inverso (i.e., de *subida*) normalmente fica ociosa.

A disponibilização dos dados de entrada para os diversos nós em um multicomputador Beowulf é normalmente feita com uso de um sistema de arquivos em rede, de maneira que todos os nós computacionais tenham acesso de leitura e escrita a qualquer arquivo. Em geral, a rede utilizada para tráfego destes dados é a própria árvore gorda Ethernet, sendo NFS o sistema de arquivos comumente utilizado. Para os casos onde o servidor NFS é um limitador do desempenho, pode-se usar sistemas de arquivos paralelos, como GPFS [88] ou Lustre [86]. Há também opções de sistemas de arquivos compartilhados com acesso através de rede FC (i.e., SAN) ou Infiniband. De qualquer forma, o acesso aos arquivos a partir de qualquer nó computacional é um requisito comum, que facilita o desenvolvimento e uso das aplicações. Além disto, para multicomputadores que usam sistemas operacionais UNIX, Linux e FreeBSD (que representam a quase totalidade dos multicomputadores usados em HPC [100]), os sistemas de arquivos são invariavelmente aderentes ao padrão POSIX.

Todos os multicomputadores mostrados na Tabela 8.1 na página 79, que serão apresentados na Seção 8.1.2, são exemplos reais de multicomputadores Beowulf com a arquitetura discutida nesta seção. Mesmo que ambientes HPC sejam caracterizados por taxas de entradas e saídas de pares muito menores do que as encontradas em aplicações utilizadas na Internet, a dimensão que os multicomputadores podem atingir, com até dezenas ou mesmo centenas de milhares de processadores [100], nos leva a falhas rotineiras de componentes [5], de modo que soluções auto-reorganizáveis sejam interessantes e desejáveis.

Apesar de nesta seção termos focado nossa discussão em multicomputadores com tráfego de dados através de redes Ethernet, que é o caso mais comum, topologias em árvore gorda são também usadas com outras tecnologias, incluindo Infiniband e FC, sem invalidar os conceitos básicos discutidos aqui. Além disto, a discussão aqui conduzida em torno de multicomputadores Beowulf é válida também para multicomputadores híbridos que têm recentemente sido implantados para computação heterogênea com aceleradores de desempenho (p.ex., GPGPU [63]), uma vez que estas duas classes de multicomputadores têm várias similaridades.

Deve-se ainda notar que da mesma maneira que o padrão de acesso a dados de algumas aplicações paralelas, como discutido na Seção 7.1, encontra similares entre aplicações populares, a ociosidade da atividade dos discos e das conexões de rede que muitas vezes ocorre em multicomputadores Beowulf, pode também ser observada nos computadores individuais de empregados de uma corporação (ou mesmo em computadores domésticos) quando acessam aplicações como VoD e *media streaming* (entre outras) através de uma Intranet (ou Internet). Além disto, assim como a agregação das modestas demandas de leitura geradas por centenas ou milhares de nós computacionais pode gerar sérias contenções em sistemas de arquivos com arquitetura Cliente/Servidor, provedores de serviços VoD e *media streaming* populares têm que satisfazer demandas agregadas que podem ser altamente desafiadoras, mesmo que as demandas individuais dos seus diversos usuários sejam modestas. Desta maneira, acreditamos que o mesmo contexto existente em centros de computação de alto desempenho que nos levaram ao desenvolvimento de D1HF pode ocorrer, de maneira similar, com algumas aplicações populares (seja em Intranets corporativas ou na Internet), de maneira que a nossa proposta, que será apresentada nas próximas seções, pode trazer contribuições que sejam usadas para endereçar as necessidades destas aplicações.

7.3 Objetivos e Características Principais de D1HF

Em virtude do contexto atual acima apresentado e discutido, estamos nesta tese introduzindo, desenvolvendo e avaliando o sistema de arquivos D1HF. Em resumo, as principais características e objetivos de D1HF são:

(i) Habilitar a avaliação de D1HT como um substrato útil, robusto e eficiente para construção de aplicações, ferramentas e sistemas P2P auto-reorganizáveis com desempenho crítico;

(ii) Otimizar o desempenho de aplicações com acesso de leitura repetitivo, padrão de

acesso encontrado em ambientes HPC, mas que também é observado em aplicações distribuídas populares;

(iii) Minimizar a carga de leitura nos sistemas utilizados para armazenamento dos dados acessados segundo os padrões de acesso endereçados, de maneira que estes sistemas de arquivos possam ser implantados segundo padrões populares e com uso de componentes comuns e padronizados;

(iv) Ter arquitetura puramente P2P;

(v) Ser auto-reorganizável;

(vi) Ser implantado sem a necessidade de aquisição de quaisquer componentes de *software* ou *hardware*;

(vii) Explorar apenas os recursos de *hardware* que estejam ociosos, como discos locais, conexões de rede, memória principal e comutadores TOR de multicomputadores Beowulf;

(viii) Ser integralmente compatível com o padrão POSIX, de modo a facilitar o seu uso de maneira transparente pelas aplicações;

(ix) Ser de fácil implantação e integração com padrões e rotinas já em uso em ambientes corporativos;

Apesar de estarmos inicialmente endereçando somente multicomputadores usados em ambientes HPC, pretendemos no futuro avaliar a viabilidade de D1HT para aplicações distribuídas (como VoD ou *media streaming*) em uso por computadores Linux conectados através de Intranets corporativas. Além disto, acreditamos que algumas de nossas estratégias e resultados experimentais poderão ser úteis para embasar o projeto e implantação de sistemas distribuídos de larga escala na Internet.

Deve-se ressaltar que, em contrapartida às diversas características positivas listadas acima, D1HF endereça uma classe específica de padrões de acesso, podendo na verdade trazer prejuízos de desempenho quando usado em outras circunstâncias (como será exemplificado na Seção 9.2). Acreditamos que esta questão não causará maiores problemas uma vez que, aliado ao seu baixo custo e a sua compatibilidade com o padrão POSIX, o uso de D1HF para um (ou vários) arquivo(s) de uma aplicação pode ser facilmente ativado ou desativado pelo seu próprio usuário, como será visto na Seção 7.11. Além disto, mesmo que a classe de padrões de acesso endereçados por D1HF seja limitada, estes padrões não só representam uma fração significativa do tráfego de rede observado tanto em

ambientes HPC quanto em Intranets e Internet, como também estão presentes em algumas aplicações chaves com latência crítica (com em VoD e nas aplicações HPC discutidas na Seção 7.1).

Nas seções seguintes iremos apresentar e discutir as características e estratégias usadas por D1HF para atingir seus objetivos, e no Capítulo 9 iremos apresentar os resultados alcançados por D1HF em nossos experimentos.

7.4 Uma Visão Geral de D1HF

A conjunção de aplicações paralelas com demandas de comunicação dominadas por repetitivas leituras ao mesmo conjunto de dados com os multicomputadores Beowulf com topologia em árvore gorda, nos leva a um ambiente em que as conexões de rede, os comutadores TOR e os discos locais dos diversos nós de processamento do multicomputador ficam predominantemente ociosos, enquanto ocorre uma alta carga no sistema utilizado para armazenamento dos dados de entrada.

Ao invés de usar soluções complexas para este problema, nesta tese estamos propondo um novo sistema de arquivos puramente P2P, que foi batizado de D1HF, que procura fazer uso da capacidade em disco que esteja ociosa nos diversos nós de um multicomputador para armazenar cópias dos arquivos lidos pelas aplicações distribuídas. Além disto, as partes mais acessadas destas cópias serão automaticamente mantidas na memória principal dos seus nós, na medida de sua disponibilidade. Desta maneira, o conjunto formado pela memória principal e discos locais dos diversos nós atuará como um cache em dois níveis para os dados lidos pela aplicação, de maneira que leituras repetitivas possam ser satisfeitas com estas cópias cache dos arquivos de entrada, que serão então trafegadas através das conexões de rede e comutadores TOR sub-utilizados.

Do mesmo modo em que, por exemplo, o sistema de arquivos NFS provê acesso de entrada e saída via rede a dados originalmente armazenados em diversos tipos distintos de sistemas de arquivo [73], D1HF foi projetado de maneira a ser capaz de adicionar esta funcionalidade de cache P2P a qualquer sistema de arquivos de rede que seja aderente ao padrão POSIX [67]. Desta maneira, assim como diversos outros sistemas de arquivos (p.ex., CIFS, Google File System, Lustre e o próprio NFS), D1HF não dispõe de uma estrutura própria para armazenamento dos arquivos, fazendo uso de um sistema de arquivos subjacente para desempenhar esta função. Por exemplo, assim como um sistema de arquivos NFS repassa para o seu sistema subjacente (p.ex., um sistema de arquivos ext2) todas as chamadas de operações que lhe são feitas (e será o sistema ext2 que efetivamente realizará as operações nos dados armazenados), D1HF irá repassar para seu

sistema subjacente (p.ex., um sistema de arquivos RedHat GFS) várias das chamadas que lhe são endereçadas (e será o sistema RedHat GFS que efetivamente realizará as operações repassadas nos dados armazenados). Na verdade, os resultados que apresentaremos no Capítulo 9 mostrarão que em alguns casos D1HF é capaz resolver mais de 99% das operações de leitura sem acessar o seu sistema subjacente.

Para que um sistema de arquivos possa atuar como sistema subjacente para D1HF, ele deverá atender às duas seguintes exigências:

(i) Ser aderente ao padrão POSIX.

(ii) Disponibilizar os arquivos em todos os nós que usarão D1HF.

É importante notar que, como discutido na Seção 7.2, estes dois requisitos já são comumente atendidos pelos sistemas de arquivos usados em multicomputadores Beowulf. O que é mais interessante, em virtude destes dois únicos requisitos, o próprio sistema diretamente subjacente a D1HF poderá fazer uso de um outro sistema de arquivos como sistema subjacente, e assim por diante, podendo nos levar a uma pilha com três (ou mais) sistemas de arquivos [33, 104]. Assim, para evitar confusões, iremos chamar o sistema de arquivos imediatamente subjacente a D1HF de Sistema BASE, de maneira a facilitar a sua diferenciação em relação aos outros sistemas de arquivos subjacentes que podem estar sendo usados de maneira empilhada para prover acesso aos arquivos.

Deve-se ainda notar que D1HF não impõe restrições à arquitetura do Sistema BASE (ou seus sistemas subjacentes), sendo que o acesso aos arquivos a partir dos nós do multicomputador pode ser feito tanto através de redes FC (SAN), quanto Ethernet, IB ou mesmo Myrinet. Além disto, D1HF é um sistema puramente P2P de maneira transparente e independente da arquitetura do Sistema BASE, sendo que D1HF suporta sistemas subjacentes com arquiteturas variando entre Cliente/Servidor até P2P. Assim, existem vários tipos de sistemas de arquivos distintos que podem atuar como Sistema BASE, incluindo, mas não se limitando a, NFS, RedHat GFS, SGI CXFS, Quantum StorNext e IBM GPFS. Esta versatilidade facilita a integração de D1HF a ambientes que tenham padrões definidos para seus sistemas de arquivos em rede, devendo-se notar, entretanto, que a implementação que apresentaremos na Seção 7.11 somente foi testada e avaliada sobre Sistemas BASE NFS executando em Linux.

Arquivos que são lidos através de D1HF podem também ser lidos ou escritos através do Sistema BASE. Nos casos em que, como discutido acima, o Sistema BASE for por sua vez construído sobre outro sistema subjacente, poderemos ter uma pilha com três (ou

mais) sistemas de arquivos distintos acessando os mesmos dados. Por exemplo, D1HF pode ser implantado sobre um sistema de arquivos NFS, que por sua vez esteja sendo usado para disponibilizar em rede dados que estejam armazenados em um sistema de arquivos JFS local a um servidor AIX. Assim, estes arquivos poderiam ser acessados através de D1HF ou NFS a partir de qualquer nó do multicomputador, ou ainda através de JFS no servidor AIX onde os dados residem. No entanto, como veremos na Seção 7.10, esta pilha de sistemas de arquivos não traria problemas de consistência de dados para as aplicações que já os acessavam através do Sistema BASE.

Em D1HF, à medida que são lidos, os arquivos de entrada são particionados em *blocos* que são distribuídos pelos diversos nós do multicomputador para serem armazenados em seus discos locais. Desta maneira, porções de arquivos que sejam lidas por mais de um processo serão trazidas do Sistema BASE para os discos locais apenas uma vez, de modo que os demais acessos serão satisfeitos a partir destas cópias cache, sendo que a distribuição destas cópias, e sua subsequente localização pelos diversos pares do sistema, são orquestradas por D1HT. Deste modo, D1HF alivia a carga de leitura no Sistema BASE em até R vezes (aonde R é o nível de releitura, como definido na Equação 7.1), distribuindo esta carga pelos diversos nós computacionais do multicomputador. Deve-se ressaltar ainda que D1HF é capaz de reduzir a carga tanto de releituras sequenciais quanto de aleatórias, como será visto nos resultados que serão apresentados no Capítulo 9.

Para aplicações com alta carga de leitura repetitiva a dados, como as discutidas na Seção 7.1, que são o objetivo de D1HF, deveremos ter altos valores para R e portanto uma alta redução de carga no Sistema BASE. Assim, a estratégia P2P proposta para D1HF não só pretende acelerar o acesso de leitura aos dados destas aplicações, como também deverá permitir que sejam usadas soluções simples, comuns e padronizadas de armazenamento em disco para o Sistema BASE, mas para tanto D1HF deverá prover um bom balanceamento da carga de leitura pelas cópias de blocos distribuídas pelos diversos nós do multicomputador.

Deste modo, para o sucesso da estratégia proposta, o uso de D1HT tem um papel fundamental, já que não só permite a localização rápida dos dados distribuídos pelos discos locais do multicomputador, como também dispensa o uso de um servidor de diretórios e/ou metadados, além de ser um dos responsáveis pela distribuição de carga pelos diversos nós do multicomputador. Além disto, D1HT permite que D1HF tenha uma implementação puramente P2P e auto-reorganizável, e, sobretudo, sem custos de equipamentos e *software*, uma vez que D1HF será, em breve, distribuído de maneira livre e gratuita, e sua implementação usa apenas componentes de *hardware* convencionais, já existentes e ociosos.

Assim, de maneira coerente com os objetivos apresentados na Seção 7.3, as estratégias apresentadas permitem que D1HF possa explorar a arquitetura dos multicomputadores Beowulf e a localidade de acesso a dados de aplicações distribuídas, de maneira a permitir acesso rápido a dados que tenham alta carga de leitura repetitiva mesmo que estes estejam armazenados em sistemas de arquivos BASE e dispositivos que sejam comuns e padronizados.

Além de diminuir custos e aumentar desempenho, a nossa proposta facilita a integração de D1HF ao ambiente de produção das corporações, uma vez que permite que outros aplicativos que sejam externos ao multicomputador (incluindo rotinas de *backup*, replicação de dados, visualização volumétrica, etc.) tenham acesso transparente a todos os arquivos através do Sistema BASE, o qual poderá seguir o padrão já localmente estabelecido para sistema de arquivos de rede. Em contraste, os sistemas de arquivos de alto desempenho comumente utilizados em ambientes HPC (como, por exemplo, GPFS [88] e Lustre [86]), além de usar arquitetura Cliente/Servidor e requerer procedimentos de gerenciamento e *hardware* especiais e dedicados para que sejam eficientes, são difíceis de serem integrados à rotina de uma corporação, uma vez que o seu uso depende de instalação de agentes especiais nos clientes de rede, que em geral são incompatíveis com soluções de *backup* e replicação (entre outras) em uso pela corporação. Além disto, estes sistemas de arquivos especiais muitas vezes geram problemas de incompatibilidade ou interoperabilidade com outros computadores que tenham que fazer acesso a estes dados mas usem outros sistemas operacionais. Por exemplo, o sistema de arquivos paralelo de alto desempenho IBM GPFS [28] não é compatível com a solução de backup Legato Networker [59], que está largamente difundida em várias empresas, nem com diversos sistemas operacionais, incluindo Sun Solaris, SGI IRIX, HP-UX e FreeBSD, entre outros. Assim, muitas corporações podem ter que criar custosas exceções ao seu padrão de solução de *backup* para poder tratar dados armazenados em sistemas GPFS. Além disto, é comum que os dados usados por multicomputadores Beowulf sejam pré ou pós trabalhados em estações de trabalho ou centros de visualização, o que pode trazer dificuldades para instalações que usem GPFS em seus multicomputadores, mas que precisem que os mesmos dados sejam visualizados em estações com sistemas operacionais não suportados.

D1HF disponibiliza todas as primitivas POSIX de sistemas de arquivos, mas suas otimizações são implementadas através das primitivas POSIX de `open()`, `read()`, `lseek()` e `close()` que sejam direcionadas a arquivos comuns abertos somente para leitura (i.e., arquivos abertos com opção POSIX `O_RDONLY`), como será discutido nas quatro próximas seções. Todas as demais operações (inclusive acessos de leitura a arquivos especiais, como diretórios) são repassadas ao Sistema BASE. Deve-se ressaltar também que as qua-

tro primitivas que serão discutidas nas próximas seções foram implementadas em D1HF de maneira integralmente compatível com o padrão POSIX, inclusive quanto aos códigos de erro retornados, de modo a tornar o seu uso transparente para as aplicações.

7.5 Abrindo Um Arquivo

Caso esteja sendo aberto um arquivo especial (como, por exemplo, um diretório), ou caso o acesso solicitado não seja unicamente de leitura (i.e., o acesso solicitado não foi definido como POSIX `O_RDONLY`), a primitiva de abertura é repassada diretamente para o Sistema BASE, assim como todas as demais operações direcionadas a este arquivo aberto. Alternativamente, poderíamos ter optado por também direcionar solicitações de acesso de leitura/escrita (i.e., arquivos abertos com a opção POSIX `O_RDWR`) para D1HF, de modo que ele atuasse no arquivo aberto até o momento em que uma operação que modifique o seu conteúdo fosse realizada, procurando assim endereçar arquivos somente de leitura que estejam equivocadamente sendo abertos como leitura/escrita. Podemos avaliar esta alternativa no futuro, mas acreditamos que seria contraproducente.

Por outro lado, quando um processo executando em um nó P abre para leitura um arquivo F [1], D1HF irá inicialmente abrir este arquivo localmente ao nó solicitante através do Sistema BASE. Isto é feito de maneira a confirmar a existência de F, e verificar se o usuário em questão tem a permissão necessária para abri-lo. Este acesso direto ao arquivo no Sistema BASE será mantido aberto, uma vez que, como será visto na próxima seção, D1HF poderá utilizá-lo sempre que ocorram falhas no acesso às cópias remotas dos diversos blocos deste arquivo.

Uma vez que F tenha sido aberto com sucesso através do Sistema BASE local a P, D1HF irá então construir uma chave a partir do nome completo deste arquivo, e encaminhará uma consulta a D1HT para determinar o nó H responsável por esta chave (i.e., o *home node* desta chave), o qual será o responsável por armazenar uma cópia do primeiro bloco do referido arquivo. Assim, P irá abrir um conexão TCP com H, e enviará, através desta conexão, uma mensagem solicitando a abertura de F. Esta conexão será posteriormente utilizada para enviar as requisições de leitura e receber os dados solicitados, como será discutido na Seção 7.6. Futuramente serão avaliadas outras estratégias para seleção dos nós que armazenarão as cópias iniciais dos blocos. Por exemplo, talvez seja mais eficiente que o primeiro nó a realizar uma leitura a um bloco seja o responsável por armazenar a sua cópia inicial, sendo que neste caso os *home nodes* das diversas chaves

[1] Ou seja, quando o processo executa a primitiva POSIX `open( F, O_RDONLY | ... )`, com F sendo acessado através de um sistema de arquivos D1HF.

atuariam apenas como diretórios indicando em que nó(s) cada bloco está guardado.

Ao receber a solicitação de abertura de conexão TCP para leitura de F, H irá iniciar um fluxo de processamento dedicado para esta conexão (i.e., uma *thread* POSIX). Caso ainda não tenha recebido nenhuma solicitação para este arquivo, H deverá retornar a P uma ordem para que seu acesso a F seja feito diretamente ao Sistema BASE. Esta estratégia, que ainda não está presente na versão atual da nossa implementação de D1HF, procura evitar os custos da criação de uma cópia cache de blocos que sejam lidos por apenas um nó, já que D1HF só agrega desempenho quando há releitura de arquivos.

Caso este seja (pelo menos) o segundo acesso distinto a F, e H ainda não disponha de uma cópia cache do seu primeiro bloco, H irá iniciar imediatamente a cópia deste bloco a partir Sistema BASE para o seu disco local, mesmo antes que qualquer acesso de leitura a F tenha sido feito. Caso já exista localmente uma cópia do bloco solicitado, será verificada se a data em que esta cópia foi criada (com precisão de milissegundos) é igual ou posterior à data atual de modificação do arquivo no Sistema BASE. Caso positivo, a versão do bloco em cache local será utilizada, caso contrário ela será eliminada e uma nova versão será criada. O modelo de consistência de dados implementado por D1HF será discutido com mais detalhes na Seção 7.10.

À medida que F seja lido, os seus demais blocos serão copiados para os discos locais dos seus respectivos *home nodes*, segundo procedimento similar ao usado para o seu bloco inicial, sendo que as chaves dos diversos outros blocos serão criadas através da associação do nome original do arquivo ao número serial do bloco. Deste modo, em consequência da função criptográfica usada por D1HT para distribuição das chaves ao longo do anel de identificadores, os diversos blocos de cada arquivo serão distribuídos aleatoriamente pelos nós do multicomputador.

Assim como para o primeiro bloco, D1HF procurará sempre fazer a cópia dos demais blocos de qualquer arquivo antes que elas sejam efetivamente acessadas. Para otimizar este processo, cada par poderá ter várias conexões abertas para diferentes blocos de um mesmo arquivo, sendo a quantidade destas conexões limitadas por um parâmetro de D1HF, de maneira a evitar que tenhamos um grande número de conexões abertas que sejam potencialmente inúteis.

Deve-se notar que à medida que haja disponibilidade de memória principal nos *home nodes* dos diversos blocos, o próprio sistema operacional (não só Linux, como quase todos os outros) irá procurar manter em memória as suas partes mais acessadas, diminuindo assim a latência de acesso aos dados mais concorridos, e minimizando a carga nos discos locais. Assim, a estratégia usada por D1HF habilita o uso de memória principal para otimização de acesso a dados populares de maneira automática e transparente para o sistema

operacional e a aplicação, como será comprovado pelos resultados que serão apresentados no Capítulo 9.

Enquanto na implementação preliminar de D1HF, como será apresentada na Seção 7.11 e avaliada no Capítulo 9, cada bloco tem uma única cópia cache, este sistema foi desenhado de maneira que blocos que sejam acessados concorrentemente por vários nós distintos tenham diversas réplicas, procurando-se assim evitar contenções a blocos muito acessados. Para tanto, após ter sido atingido um nível de concorrência de acesso que poderá ser dinamicamente ajustado, novas réplicas do bloco seriam criadas de maneira a controlar a concorrência por cópia. A criação e o acesso às novas réplicas seriam feitos de modo a tirar proveito da topologia de rede em árvore gorda do multicomputador. Para tanto, as novas réplicas seriam distribuídas pelos nós do multicomputador de maneira a haver no máximo uma cópia por comutador TOR. Assim, os acessos provenientes de um determinado nó seriam direcionados para a cópia que estivesse em um nó conectado ao mesmo comutador TOR, caso exista, de modo a explorar a localidade de acesso às cópias dos blocos. Deste modo, os acessos contidos internamente a um mesmo comutador TOR teriam latências menores, e seria minimizado o uso (e as eventuais contenções) nas conexões dos comutadores TOR ao nível superior da árvore gorda.

7.6 Lendo Dados

Quando um processo P solicita a leitura de um dado de um arquivo F aberto através de D1HF, será inicialmente determinado em que bloco B este dado reside. Caso P ainda não tenha uma conexão aberta para leitura de B com o seu *home node* H, P irá instaurar esta conexão segundo o procedimento apresentado na seção anterior. Uma vez que esta conexão esteja estabelecida, P enviará a H uma requisição para o dado em questão e aguardará a sua resposta. H irá então ler o dado solicitado, enviando-o a P, que então o repassará à aplicação segundo o padrão POSIX.

De modo coerente com o mecanismo descrito acima, os resultados medidos de latência de leitura média de D1HF que serão apresentados na Seção 9.4 dividem a latência de leitura observada pela aplicação em um componente `open`, que reflete o tempo médio despendido para instauração das conexões aos *home nodes*, e um componente `read`, que representa o tempo transcorrido entre o efetivo envio da solicitação de leitura e a chegada de sua resposta.

Na ocorrência de um erro, P irá resolver a leitura requisitada acessando localmente o Sistema BASE. Caso trate-se de um erro temporário (por exemplo, uma falha na comunicação com H), as próximas leituras serão novamente direcionadas para H, sendo que,

dependendo do tipo de erro, poderá ser encerrada a conexão TCP corrente e criada uma nova. No caso de falhas permanentes (por exemplo, *F* foi removido do Sistema BASE durante a execução da aplicação), todos os próximos acessos serão localmente direcionados para o Sistema BASE (e, dependendo da situação, poderão ou não falhar).

7.7 Reposicionando o Ponteiro de Leitura

D1HF mantém um ponteiro de leitura (i.e., *offset*) para cada arquivo aberto, o qual, de maneira coerente com o padrão POSIX, aponta para o início do arquivo quando este é aberto (ou seja, a rotina `open()` inicializa o valor do *offset* com zero). Também de acordo com o padrão POSIX, ao final de cada operação de leitura este ponteiro é redirecionado para o primeiro octeto após os dados lidos. Ainda de acordo com o padrão POSIX, a aplicação pode reposicionar o ponteiro de leitura de um arquivo de maneira arbitrária através da rotina POSIX `lseek()`.

Sempre que o ponteiro de leitura de um arquivo seja redirecionado, D1HF irá verificar se o bloco que está sendo apontado já está aberto localmente (i.e., se o nó local já tem uma conexão TCP aberta com o seu *home node*). Caso contrário, esta conexão será instaurada segundo o procedimento apresentado na Seção 7.5.

Adicionalmente, D1HF procura acompanhar o posicionamento do ponteiro de leitura para determinar os arquivos que são lidos com ponteiros sempre crescentes ou decrescentes. Para estes padrões, D1HF procura sempre instaurar a conexão com o provável próximo bloco antes que ela seja efetivamente necessária.

7.8 Fechando um Arquivo

Quando um nó fecha um arquivo, o que é feito através da rotina POSIX `close()`, D1HF irá fechar o arquivo aberto com o Sistema BASE, bem como as diversas conexões TCP com *home nodes*, além de liberar algumas áreas alocadas em memória.

Deve-se notar que as cópias dos blocos de cada arquivo continuarão residentes nos seus *home nodes* mesmo que todas as suas conexões sejam encerradas. As cópias dos blocos só são eliminadas quando D1HF determina que elas estejam inconsistentes com a versão atual do arquivo no Sistema BASE (como discutido na Seção 7.5), ou através de uma rotina de *garbage collection* (ainda não implementada) que será acionada quando a área em disco local para armazenamento das cópias dos blocos atingir nível de ocupação superior a um limite pré-estabelecido.

7.9 Adesão e Partida de Pares

D1HF foi projetado, e está sendo implementado, de maneira a automaticamente se reorganizar na medida em que ocorram adesões e saídas de pares no sistema, de modo transparente para a aplicação, a menos de eventuais flutuações de desempenho que possam ser decorrentes destes eventos.

A adesão ou partida de um par é automaticamente tratada e propagada por D1HT. Deve-se notar que como consequência de qualquer evento, os *home nodes* de alguns blocos podem mudar. Por exemplo, um novo par que entre em um sistema D1HF passará a ser o *home node* de alguns blocos que estavam sob responsabilidade do seu sucessor. Do mesmo modo, a responsabilidade dos blocos associados a um par que deixe o sistema será repassada para o seu sucessor.

Uma vez que qualquer evento é eficientemente disseminado para todos os pares do sistema segundo EDRA, a reorganização automática dos *home nodes* em D1HF é uma propriedade herdada diretamente de D1HT, que continuamente irá garantir que todos os pares no sistema vejam a mesma associação entre blocos e *home nodes*, mesmo sob ocorrência de eventos.

Deve-se notar, entretanto, que, na ocorrência de um evento, D1HF não se preocupará em copiar os blocos já residentes de seu *antigo home node* para o seu *novo home node*, porque acreditamos que esta operação poderia ser contraproducente. De fato, quando os blocos que tenham *migrado* de *home node* comecem a ser acessados após o respectivo evento, o seu novo *home node* não disporá de suas cópias em disco local, de modo que elas serão reconstruídas a partir das versões originais dos respectivos arquivos que estão armazenadas no Sistema BASE, segundo o procedimento apresentado na Seção 7.5. Ou seja, novas cópias de blocos que tenham que ser criadas em decorrência de adesões e partidas são feitas a partir do Sistema BASE, ao invés do uso das cópias antigas. Assim, uma vez que a criação das novas cópias é feita sob demanda, esta estratégia evita a movimentação de blocos que tenham se tornados obsoletos ou que não venham a ser mais usados.

Acreditamos que as perdas de desempenho causadas por eventos devam ser pequenas para os ambientes que sejam caracterizados por pares *relativamente* estáveis e/ou por dados com vida útil *relativamente* curta, como será discutido a seguir.

Por *pares relativamente estáveis* entendemos aqueles que tenham tamanho de sessão médio suficientemente longo para que as cópias de blocos armazenadas em seus discos possam ser acessadas várias vezes antes que suas titularidades sejam migradas para um novo *home node*, ou ainda, que o custo das migrações sejam irrelevantes frente ao tamanho médio de sessão dos pares do sistema. Por exemplo, nós de multicomputadores

usados em ambientes HPC têm tipicamente tamanhos de sessão da ordem de meses, de maneira que os poucos minutos que possam durar as migrações dos blocos decorrentes de cada evento não são relevantes. Em sistemas VoD populares, cada bloco pode ter acessos frequentes, de maneira que mesmo que os seus pares tenham sessões médias da ordem de minutos, a vida média de cada cópia poderá ser suficiente para que ela seja acessada dezenas de vezes, o que compensaria os custos da sua criação.

Por *dados com vida útil relativamente curta*, entendemos aqueles que se tornem obsoletos em um intervalo de tempo muito menor do que a duração média de sessão dos pares do sistema. Por exemplo, em fluxos de vídeo ou áudio transmitidos ao vivo, cada bloco se torna obsoleto e inútil em questão de segundos. Assim, mesmo que as durações médias de sessão de seus pares sejam da ordem de minutos, os eventos que ocorram não devem ter impacto significativo no desempenho e custos do sistema, especialmente por que a estratégia sob demanda que propomos nesta seção irá automaticamente evitar custos de movimentação (ou *recriação*) de cópias de blocos que tenham se tornados obsoletos.

7.10 Consistência de Dados

D1HF implementa consistência de dados *close-to-open*, típica de NFS [65]. Mas, uma vez que este sistema não atua nos acessos de escrita, estes são feitos segundo o modelo de consistência original do Sistema BASE. Desta maneira, D1HF não modifica o modelo de consistência observado por aplicações que originalmente acessavam seus dados através de NFS.

Por outro lado, aplicações que requerem um modelo de consistência mais forte do que *close-to-open* para alguns (ou todos) arquivos de entrada, não devem acessá-los via D1HF, o que não é preocupante por dois motivos. Primeiro por que o acesso a estes arquivos através do Sistema BASE estará sempre disponível para qualquer aplicação. Segundo, por que modelos de consistências mais fortes são tipicamente necessários para os casos em que os arquivos de entrada podem ser modificados enquanto a aplicação esteja executando, e portanto o uso de D1HF para estes arquivos provavelmente não seria proveitoso, já que teríamos que continuamente invalidar as cópias de dados distribuídas nos diversos nós do multicomputador à medida que a sua versão no arquivo BASE fosse modificada, o que reduziria (ou simplesmente eliminaria) a efetividade do mecanismo de cache implementado por D1HF.

Aplicações de *media streaming* são, no entanto, um caso à parte, em especial a transmissão de conteúdo *ao vivo*. Neste caso, o conteúdo de entrada é continuamente gerado, mas dados já criados não são modificados, em um padrão de acesso do tipo tubo (ou *pipe*),

onde se têm um processo produtor de conteúdo (i.e., escritor) e vários consumidores (i.e., leitores). Assim, a consistência de leitura provida por D1HF para estes processos consumidores será satisfatória, desde que seja garantido que nenhuma leitura seja feita além do final corrente do tubo, o que acreditamos que seria simples de ser implementado.

7.11 Implementação

Como parte do trabalho desta tese, implementamos as funcionalidades de D1HF como descritas neste capítulo, de maneira compatível com o padrão POSIX, resultando em cerca de 4.500 linhas de código C++. A versão atual da implementação já foi exaustivamente testada em centenas de nós executando Linux 2.6, usando-se NFSv3 como Sistema BASE. No entanto, julgamos que esta versão ainda não está madura o suficiente para ser usada em produção corporativa, apesar de ter-se mostrado estável e eficiente durante a avaliação experimental com até 800 nós físicos distintos de um multicomputador, que será apresentada no Capítulo 9. Durante estes experimentos, D1HF não só demonstrou a sua correção, como ainda atingiu desempenho até oito vezes superior a um sistema que representa o estado da arte em termos de servidores NFS para ambientes corporativos de produção.

Apesar de mecanismos de pré-carga serem simples de implementar e estarem presentes em quase todos os sistemas de arquivos tradicionais e, principalmente, os padrões de acesso que estamos endereçando serem potencialmente passíveis de significantes ganhos de desempenho através do uso deste mecanismo, como foi discutido na Seção 7.1 e será observado nos resultados que apresentaremos no Capítulo 9, a versão atual de D1HF ainda não implementa este tipo de otimização, que será uma de nossas prioridades para seu desenvolvimento futuro.

De maneira a facilitar as nossas avaliações experimentais e a futura integração de D1HF em ambientes de produção, desenvolvemos duas maneiras para o seu uso:

D1HF montado : Nesta opção, que foi desenvolvida com uso de FUSE [98], D1HF é montado como um sistema de arquivos comum, podendo assim ser acesso por qualquer aplicativo sem modificação, incluindo utilitários Linux com `ls`, `du`, `grep`, `find`, etc.. Enquanto o módulo FUSE já está incorporado no kernel da versão Linux 2.6.21 (e posteriores), versões mais antigas ainda requerem a instalação deste *kernel module*.

D1HF incorporado na aplicação (ligado) : Criamos esta alternativa para endereçar situações em que FUSE não está disponível. Neste caso, deve-se fazer uma pequena

modificação no código fonte da aplicação de maneira a inserir uma única linha (`#include <D1HF.h>`), e recompilá-la. Além de dispensar a instalação de FUSE, esta opção deve potencialmente prover melhor desempenho, uma vez que evita os custos inseridos por FUSE, mas, obviamente, só pode ser usada nos casos onde se tem acesso ao código fonte da aplicação. Além disto, atualmente esta opção só está disponível para códigos escritos em C e C++.

Além da implantação de uma das opções acima, D1HF requer a execução de um processo *daemon* D1HF e um processo D1HT em cada nó do multicomputador. Na verdade, de maneira a otimizar o balanceamento de carga e melhorar a disponibilidade, sugerimos que sejam executados vários pares de processos D1HF/D1HT por nó (ao menos um par de processos por núcleo de processamento). Caso todos os pares de processos D1HF/D1HT de um nó falhem, os acessos a D1HF neste nó serão direcionados ao Sistema BASE.

Deve-se notar ainda que o uso transparente de D1HF por aplicações já existentes deverá, no caso geral, demandar a sua disponibilização em modo montado, o que, por sua vez, irá requerer a disponibilidade de FUSE nos diversos nós do multicomputador. Não acreditamos, porém, que este pré-requisito irá comprometer de maneira significativa o nosso objetivo de fácil integração em ambientes corporativos, como apresentado na Seção 7.3. Primeiro, por que em multicomputadores modernos este pré-requisito será automaticamente atendido a partir do simples uso de uma versão atual de Linux, que deverá ser o caso comum em ambientes HPC. Segundo, por que mesmo para multicomputadores Linux antigos, a instalação de FUSE, apesar de não ser trivial, é muito mais simples do que a instalação dos pré-requisitos de diversos outros sistemas de arquivos de alto desempenho, como GPFS e Lustre. Deve-se notar que a instalação do módulo kernel de FUSE não requer a reinicialização (i.e., *reboot*) dos nós.

Por outro lado, apesar de existir versões de FUSE para diversos sistemas operacionais (incluindo FreeBSD, OpenSolaris, NetBSD e MacOS), ele ainda não é suportado por alguns sistemas populares (em especial, MS Windows). De qualquer maneira, há outros requisitos de D1HF, bem como outros sistemas de arquivos de alto desempenho (incluindo Lustre), que também não têm suporte em Windows. Além disto, D1HF permite o acesso a dados disponibilizados em rede por qualquer Sistema BASE aderente ao padrão POSIX, independente do sistema operacional em uso no servidor que efetivamente aloje os arquivos (inclusive Windows), apesar da implementação corrente só ter sido testada sobre Sistema BASE NFS.

De maneira a prover acesso transparente tanto ao Sistema BASE quanto ao sistema de arquivos D1HF (seja montado ou incorporado na aplicação), os seus arquivos ficam disponíveis com a adição do prefixo `/D1HF/` aos nomes originais. Por exemplo, caso o Sis-

tema BASE `/fileread` (que contenha o arquivo `/fileread/test.c`) seja montado com D1HF, o mesmo arquivo `test.c` poderá ser acessado através do Sistema BASE usando-se o caminho original `/fileread/test.c`, ou através de D1HF usando-se o caminho `/D1HF/fileread/test.c`. Acreditamos que este tipo de facilidade seja fundamental, porque ela permite que o usuário de uma aplicação paralela selecione de maneira absolutamente simples quais são os arquivos que devam ser lidos através de D1HF, evitando assim o seu uso para arquivos que sejam acessados segundo padrões que não são propícios de otimização por suas estratégias.

Em D1HF, o tamanho dos blocos é um parâmetro que deve ser cuidadosamente escolhido. Blocos muito grandes irão restringir o paralelismo e o balanceamento da carga de leitura a dados muito acessados. Por outro lado, tamanhos pequenos irão gerar uma maior quantidade de blocos, o que aumentará os custos de criação e manutenção de conexões e réplicas. Para o primeiro bloco estes custos são especialmente críticos, já que para os demais existem mais oportunidades para que a criação das cópias e conexões sejam realizadas antes que venham a ser realmente necessárias, como discutido na Seção 7.7. Assim, de maneira a permitir ajustes mais precisos, nossa implementação de D1HF permite dois tamanhos distintos de blocos, sendo um o tamanho do primeiro bloco de cada arquivo, e o outro sendo o tamanho de todos os demais blocos de todos os arquivos. Na sua versão atual estes parâmetros devem ser escolhidos em momento de compilação de D1HF, mas pretendemos estudar maneiras mais flexíveis para sua definição.

Na medida em que a nossa implementação de D1HF se torne mais madura e tenha incorporado algumas funcionalidades que julgamos obrigatórias, nós disponibilizaremos seu código fonte para uso livre e gratuito.

Capítulo 8

Resultados Experimentais e Analíticos de D1HT

Neste capítulo apresentaremos de maneira objetiva o conjunto de resultados experimentais e analíticos obtidos para D1HT durante o trabalho desta tese, que serão posteriormente discutidos no Capítulo 10. Os resultados que apresentaremos neste capítulo incluem não só avaliações comparativas das demandas de manutenção de tabelas de roteamento de D1HT, como também medições experimentais das latências de suas consultas, comparando-as com as obtidas por três outros sistemas reais.

Começaremos com a apresentação dos resultados experimentais obtidos com as nossas implementações de D1HT e 1h-Calot, que validaram as suas análises teóricas quantitativas e nos levaram a diversas conclusões relevantes. Em seguida, na Seção 8.2, apresentaremos os resultados obtidos com estas já validadas análises, que nos permitiram comparar os custos de manutenção de D1HT com os obtidos com OneHop e 1h-Calot, para sistemas com até dez milhões de pares e diferentes dinâmicas de pares, além de estudar os efeitos do mecanismo de Quarentena.

8.1 Resultados Experimentais

Nesta seção, serão apresentados os resultados experimentais obtidos com as nossas implementações reais de D1HT e 1h-Calot. Enquanto outros trabalhos já haviam comparados os custos de manutenção de SHDHTs e MHDHTs (p.ex., [42, 82, 99]), os experimentos que serão apresentados nesta seção são únicos, não só por serem os primeiros a incluir avaliações conduzidas em dois ambientes completamente distintos (um CPD HPC e uma rede de computadores mundialmente dispersa), mas também por que formam a mais completa e extensa comparação entre sistemas DHT já publicada (com até 4.000 pares e 2.000

nós físicos), além de incluir a primeira comparação entre as latências obtidas com três diferentes DHTs e um servidor de diretórios.

Daremos continuidade a esta seção com a discussão da nossa implementação de 1h-Calot e da metodologia utilizada nos experimentos, para então apresentarmos as medições de demandas de banda passante de manutenção no ambiente disperso (PlanetLab) e no CPD HPC, seguidos dos resultados dos experimentos comparativos de latência.

8.1.1 Implementação de 1h-Calot

De maneira a permitir a comparação experimental de D1HT com a única outra SHDHT puramente P2P que é capaz de suportar ambientes dinâmicos, foi desenvolvida uma implementação do sistema 1h-Calot, uma vez que tal implementação não havia sido feita pelos seus próprios autores.

A nossa implementação de 1h-Calot foi construída tendo como ponto de partida o código fonte de D1HT, procurando-se manter ao máximo a similaridade entre estas duas implementações, com o objetivo de garantir que eventuais diferenças que viessem a ser observadas nos resultados experimentais obtidos para estes dois sistemas não fossem devidas a questões de implementação. Desta maneira, nossa implementação de 1h-Calot também é executada como um processo estanque, e as mensagens de manutenção e consultas são implementadas sobre UDP, com controle de perdas e retransmissões implementado explicitamente.

Além dos diversos tipos de mensagens presentes em D1HT, 1h-Calot também faz uso de mensagens enviadas com periodicidade fixa com o objetivo único de detectar falhas em sucessores e predecessores (i.e., *heartbeats*), sendo que este tipo de mensagem foi também implementado sobre UDP, mas dispensa controle de perda e retransmissão. Uma vez que cada par em 1h-Calot envia quatro *heartbeats* por segundo (que não têm sua entrega confirmada), e cada mensagem de manutenção informa um único evento (e tem sua entrega confirmada), os resultados analíticos para 1h-Calot, que pretendemos validar experimentalmente, são dados pela equação:

$$B_{Calot} = r \cdot (v_m + v_a) + 4 \cdot n \cdot v_h/60 \tag{8.1}$$

onde v_m, v_a, e v_h são, respectivamente, os tamanhos das mensagens de manutenção, confirmação de entrega e *heartbeats*, r é a taxa de eventos (de acordo com a Equação 4.5 que foi apresentada na página 32), n é o tamanho do sistema, e B_{Calot} é a demanda média de banda passante de manutenção por par em 1h-Calot.

Assim como D1HT, 1h-Calot usa árvores logarítmicas para disseminação dos eventos com cada par criando até ρ sub-árvores de propagação, de maneira similar a apresentada na Figura 4.1 da página 26. Por outro lado, ao contrário de D1HT, 1h-Calot não faz uso de TTLs para construção destas sub-árvores, sendo que cada uma das sub-árvores de cada par é construída a partir da definição explícita do intervalo de identificadores (IDs) que deverá cobrir. Por exemplo, na propagação da falha do par P_p ilustrada na Figura 4.1 da página 26, enquanto em D1HT a mensagem enviada de P a P_4 informa apenas o seu TTL (o que é suficiente para P_4 definir as suas sub-árvores de propagação), em 1h-Calot esta mensagem deveria incluir o limite superior de abrangência das sub-árvores que P_4 deverá construir, ou seja, o ID de P_8 (que pode ser obtido a partir do seu IP). Deste modo, no exemplo ilustrado pela Figura 4.1, a mensagem enviada de P a P_4 em 1h-Calot conteria os endereços IP (incluindo número de porta) de P_p (par que sofreu o evento que está sendo propagado) e de P_8 (limite superior de abrangência das sub-árvores de disseminação a serem construídas por P_4), e então P_4 iria construir suas sub-árvores de maneira a cobrir o intervalo de identificadores $]ID(P_4), ID(P_8)[$. Mais detalhes sobre o modo de construção das árvores logarítmicas para propagação de eventos em 1h-Calot podem ser obtidos em [99].

Deste modo, o formato das mensagens de manutenção da nossa implementação de 1h-Calot mantém os quatro primeiros campos do cabeçalho de D1HT, como mostrado na Figura 6.1 da página 46, ou seja, os campos *Type*, *SeqNo*, *PortNo* e *SystemID*, que têm funções idênticas nas duas implementações. Por outro lado, como cada mensagem em 1h-Calot reporta apenas um evento, elas não necessitam de nenhum dos quatro contadores de eventos presentes nas mensagens de D1HT. Assim, cada mensagem de manutenção em 1h-Calot deve transportar o endereço IP completo (incluindo número de porta) do par que sofreu o evento, com a identificação do tipo de evento (adesão ou partida) sendo feita através do campo *Type*. Além disto, todas as mensagens de manutenção em 1h-Calot devem transportar o endereço do par *finger* (incluindo número de porta) a ser usado como limite de construção da próxima sub-árvore de disseminação logarítmica.

Desta maneira, enquanto as mensagens de manutenção em D1HT têm tamanho variável de acordo com a quantidade de eventos que transporta, todas as mensagens de manutenção da nossa implementação de 1h-Calot têm tamanho fixo de 48 octetos, já incluindo os 28 octetos dos cabeçalhos IPv4 e UDP, ou seja, $v_m = 48$ octetos na Equação 8.1. Por outro lado, os *heartbeats* e as mensagens de confirmação de entrega de 1h-Calot têm o mesmo formato e tamanho das mensagens de confirmação de entrega de D1HT, ou seja, tamanho fixo de 36 octetos (também já incluindo os cabeçalhos IPv4 e UDP), de modo que $v_a = v_h = 36$ octetos na Equação 8.1.

8.1.2 Metodologia

As avaliações experimentais de D1HT e 1h-Calot conduzidas nesta tese foram feitas com as implementações reais que desenvolvemos para estes dois sistemas. Em cada experimento avaliamos ambas as SHDHTs segundo uma combinação pré-determinada de taxa de eventos r e tamanho de sistema n. Para todos os tipos de experimentos foi usado tamanho médio de sessão (S_{avg}) de 174 minutos para o cálculo da taxa de eventos segundo a Equação 4.5 (apresentada na página 32), uma vez que este valor é representativo de Gnutella, e foi usado em avaliações de SHDHTs anteriores (p.ex., [23, 55, 56]). Para alguns tipos de experimentos nós também usamos $S_{avg} = 60$ minutos, de maneira a avaliar D1HT e 1h-Calot segundo dinâmicas mais desafiadoras.

Para todos os experimentos as medições de consumo de banda passante consideraram apenas o tráfego para manutenção das tabelas de roteamento e detecção de falhas, uma vez que os outros custos envolvidos, como o tráfego das consultas e transferências das tabelas de roteamento, devem ser idênticos para os dois sistemas estudados. Para todos os experimentos as tabelas de roteamento foram definidas com tamanho fixo de seis mil entradas, ocupando assim menos de 40 KB de memória principal de cada par, o que é absolutamente desprezível para nós computacionais usados em HPC, ou mesmo para computadores domésticos.

Cada experimento teve duas fases, sendo a primeira fase usada para crescer o sistema até o tamanho determinado, permitindo assim avaliar a eficácia do mecanismo de adesão, enquanto as medições foram feitas sempre ao longo da segunda fase.

Cada sistema sempre iniciou a primeira fase com apenas oito pares, e manteve-se uma taxa de adesão constante de um par por segundo até ser atingido o tamanho de sistema determinado, o que resultou em taxas de crescimento agressivas, que geraram cargas desafiadoras para os mecanismos de adesão. De fato, de acordo com a Equação 4.5 (apresentada na página 32), a taxa de eventos no início desta fase é equivalente a tamanhos médios de sessão inferiores a um minuto, que são muito menores do que aqueles que qualquer SHDHT se propõe a suportar. Uma vez que a realização de consultas permite que os pares façam correções adicionais às suas tabelas de roteamento, não foram executadas consultas durante a primeira fase dos experimentos, de maneira a garantir que a precisão das tabelas ao final desta fase era devida unicamente à correção dos mecanismos de adesão e propagação de eventos.

A segunda fase de todos os experimentos teve sempre a duração de 30 minutos, durante a qual manteve-se o tamanho do sistema constante, com todos os pares executando consultas aleatórias. Cada experimento foi executado três vezes, e os resultados que apresentaremos são as médias aritméticas destas três medições.

Em ambas as fases de todos os experimentos os sistemas foram submetidos a adesões e partidas concorrentes de acordo com a Equação 4.5 segundo o tamanho médio de sessão escolhido (60 ou 174 minutos). Para tanto, dividiu-se cada fase em intervalos com duração S_{avg}, e forçou-se que cada par sofresse exatamente uma partida aleatoriamente distribuída ao longo de cada um destes intervalos. De maneira a manter o tamanho do sistema constante, cada par retornou ao sistema três minutos depois de sua partida, usando o mesmo endereço IP e ID, o que nos permitiu também avaliar ambos os sistemas em um cenário com adesões e partidas concorrentes. Metade de todas as partidas foi forçada com o sinal POSIX `SIGKILL`, de modo a não permitir que o par alerte seus vizinhos sobre a sua saída, nem propague os eventos que tenha acumulado durante o intervalo Θ corrente.

Deve-se notar que, mesmo que nossos experimentos tenham exercitado os mecanismos de adesão ao extremo, além de impor taxas de adesões e partidas desafiadoras, tanto D1HT quanto 1h-Calot foram capazes de resolver mais de 99% das consultas com um único salto em todos os experimentos, o que mostrou na prática que os problemas discutidos na Seção 4.3 não devem ter impactos significativos na fração de falhas de roteamento destes sistemas.

8.1.3 Experimentos de Banda Passante no Ambiente PlanetLab

De modo a avaliar os custos de manutenção e o comportamento dos dois sistemas em um ambiente mundialmente disperso, compartilhado e conectado via Internet, conduzimos experimentos com ambos SHDHTs em 200 nós físicos do PlanetLab [6]. Acreditamos que estes experimentos são importantes por permitir a avaliação de D1HT e 1h-Calot em um ambiente similar ao enfrentado por sistemas P2P populares utilizados na Internet, representando assim uma importante classe de aplicações que esperamos que D1HT possa suportar.

Os experimentos no PlanetLab foram conduzidos com cinco ou dez pares de cada sistema por nó físico, o que nos levou a tamanhos de sistema de 1.000 e 2.000 pares. Em todos estes experimentos usou-se tamanho médio de sessão de 174 minutos, e cada par executou uma consulta por segundo durante a segunda fase de todos os experimentos.

Os custos de manutenção de tabelas de roteamento para ambos os sistemas são apresentados na Figura 8.1, onde são mostradas as somas das demandas de banda passante de todos os pares de cada sistema. Desta figura podemos observar que as duas SHDHTs avaliadas tiveram resultados similares para o menor tamanho de sistema, enquanto com 2.000 pares as demandas de manutenção de 1h-Calot foram 46% maiores do que as de D1HT. Os resultados experimentais adicionais que serão mostrados na Seção 8.1.4, bem como os resultados analíticos que serão apresentados na Seção 8.2, complementarão os resultados

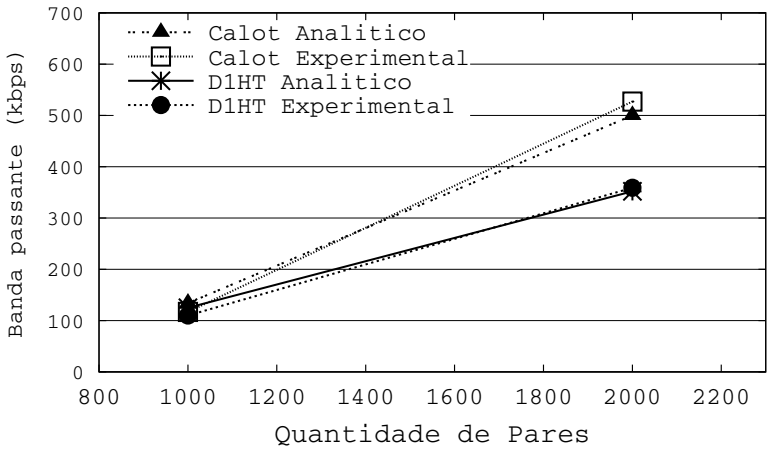


Figura 8.1: Demandas de rede de manutenção de tabelas de roteamento de D1HT e 1h-Calot medidas no PlanetLab com tamanho médio de sessão de 174 minutos (representativo de Gnutella). São mostradas as demandas medidas nos experimentos com as implementações reais dos dois sistemas, e aquelas previstas através de suas análises quantitativas teóricas.

apresentados nesta seção, e mostrarão que as diferenças de custos de manutenção entre D1HT e 1h-Calot crescem continuamente com o tamanho do sistema.

A Figura 8.1 também mostra que as análises quantitativas teóricas dos dois sistemas foram capazes de prever com boa precisão os seus resultados experimentais, o que difere de resultados preliminares que indicavam que a análise quantitativa de D1HT era até 25% conservadora em alguns casos [56]. Acreditamos que estas diferenças se devem a alguns fatores. Primeiro, por que as demandas efetivas de banda passante de D1HT sofreram acréscimos decorrentes da introdução do mecanismo para encerramento de Intervalos Θ segundo a Equação 4.11 (apresentada na página 36), que de fato não havia sido utilizado nos experimentos reportados em [56]. Adicionalmente, uma vez que a análise quantitativa de D1HT é fortemente dependente de $\rho = \lceil \log_2(n) \rceil$, seus resultados tendem a ser menos conservadores para valores de ρ próximos a $log_2(n)$ (ou seja, quando n é ligeiramente inferior a uma potência de dois), como é o caso de todos os experimentos apresentados nesta tese, apresentando resultados mais conservadores quando ρ é significativamente diferente de $log_2(n)$, como é o caso dos resultados apresentados em [56].

8.1.4 Experimentos de Banda Passante em Ambiente HPC

Uma vez que esperamos que as SHDHTs possam servir como uma importante ferramenta para aplicações sensíveis à latência, nós também avaliamos os custos de manutenção de D1HT e 1h-Calot em um ambiente formado por cinco multicomputadores de um CPD HPC para processamento sísmico [62], que são apresentados na Tabela 8.1.

Multicomputador	# Nós	CPU	S.O.
A	731	Intel Xeon 3.06GHz single core	Linux 2.6
B	924	AMD Opteron 270 dual core	Linux 2.6
C	128	AMD Opteron 244 dual core	Linux 2.6
D	99	AMD Opteron 250 dual core	Linux 2.6
F	509	Intel Xeon E5470 quad core	Linux 2.6

Tabela 8.1: Multicomputadores usados nos experimentos com D1HT e 1h-Calot no ambiente HPC.

Neste ambiente experimental, cada nó de cada multicomputador tem uma conexão 1000baseT a um comutador Ethernet TOR não bloqueante, sendo que cada comutador TOR concentra entre 16 e 48 nós e tem uma conexão com capacidade entre 2 Gbps e 10 Gbps a um comutador Ethernet núcleo não bloqueante.

Cada par executou uma consulta aleatória por segundo durante a segunda fase de todos os experimentos, e fizemos avaliações com dois valores distintos de duração média de sessão (60 e 174 minutos). Fizemos experimentos com 1.000, 2.000 e 4.000 pares, sendo que o maior tamanho de sistema foi obtido executando-se dois pares por nó, executando-se apenas um par em cada nó nos demais casos.

Todos os resultados experimentais que serão apresentados nesta seção foram obtidos com os multicomputadores sendo usados na produção normal do CPD, onde os nós tipicamente estão com todos os seus núcleos de processamento sob carga máxima de processamento (i.e., 100% de utilização de CPU), como consequência da execução de aplicações sísmicas paralelas. Não obstante, conseguimos executar todos os nossos experimentos sem acarretar uma única interferência na produção normal deste CPD, confirmando na prática que é absolutamente viável utilizar D1HT mesmo em ambientes de produção corporativa submetidos a altas cargas de processamento. De fato, além do uso de menos de 40 KB para armazenamento das tabelas de roteamento, o consumo médio de CPU por par de D1HT for inferior a 0,02% em todos os experimentos apresentados nesta seção (incluindo os ciclos usados para a realização das consultas e pelo protocolo de adesão).

Os resultados das medições de consumo de banda passante para manutenção das tabelas de roteamento neste ambiente com tamanhos médios de sessão de 60 e 174 minutos são mostrados nas Figuras 8.2 e 8.3, respectivamente. Em ambas as figuras são apresentados para D1HT e 1h-Calot tanto os resultados experimentais medidos, quanto os calculados através de suas análises quantitativas teóricas. Os resultados apresentados nestas figuras são as somas das demandas de banda passante de todos os pares de cada sistema.

Destas duas figuras podemos nitidamente observar que as análises quantitativas teóricas dos dois sistemas avaliados foram capazes de prever as suas reais demandas de ma-

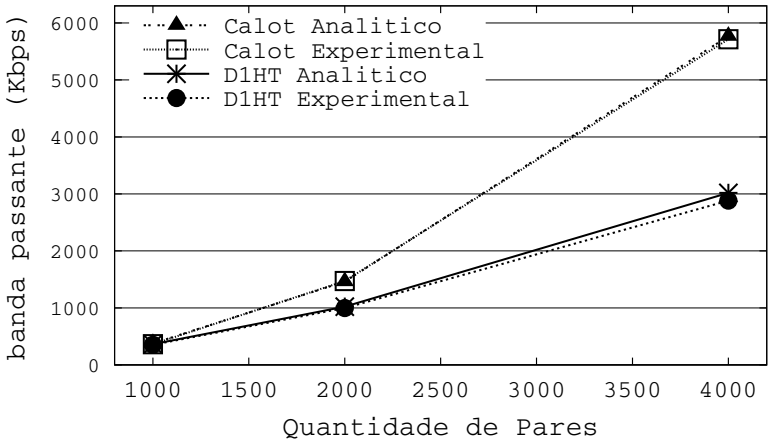


Figura 8.2: Demandas de rede de manutenção de tabelas de roteamento (em kbps) de D1HT e 1h-Calot medidas no ambiente HPC com tamanho médio de sessão de 60 minutos. São mostradas as demandas medidas nos experimentos com as implementações reais dos dois sistemas, e aquelas previstas através de suas análises quantitativas teóricas.

nutenção no ambiente HPC, para ambos os tamanhos médios de sessão utilizados. Deste modo, estes resultados, em conjunção com aqueles apresentados na Figura 8.1 para um ambiente radicalmente distinto, indicam claramente a validade das análises quantitativas elaboradas para estes dois sistemas, o que julgamos ser um resultado importante, pois nos trará confiança para o uso destas análises quantitativas em exercícios sobre os custos de manutenção destes dois sistemas, como os que serão apresentadas na Seção 8.2.

As Figuras 8.2 e 8.3 também mostraram que D1HT teve custos de manutenção de banda passante menores que os de 1h-Calot para todos os tamanhos de sistema e durações médias de sessão estudadas, confirmando, mais uma vez, que D1HT é capaz de prover uma ferramenta DHT mais leve do que 1h-Calot.

8.1.5 Experimentos de Latência em Ambiente HPC

Nesta seção iremos apresentar os nossos resultados experimentais de medições de latência de consultas obtidos com até 4.000 pares no ambiente HPC. Além das nossas implementações de D1HT e 1h-Calot, foram também medidas as latências para implementações reais de uma MHDHT e de um servidor de diretórios (que chamaremos de Dserver).

Procurando-se evitar a inserção de desvios nos resultados devidos a questões de implementação, o servidor Dserver utilizado foi basicamente um sistema D1HT com um único par, o qual foi habilitado para receber consultas de clientes externos. Inicialmente Dserver foi executado em um nó do Multicomputador B (apresentado na Tabela 8.1), mas como um primeiro indício das limitações de escalabilidade desta solução Cliente/Servidor, este

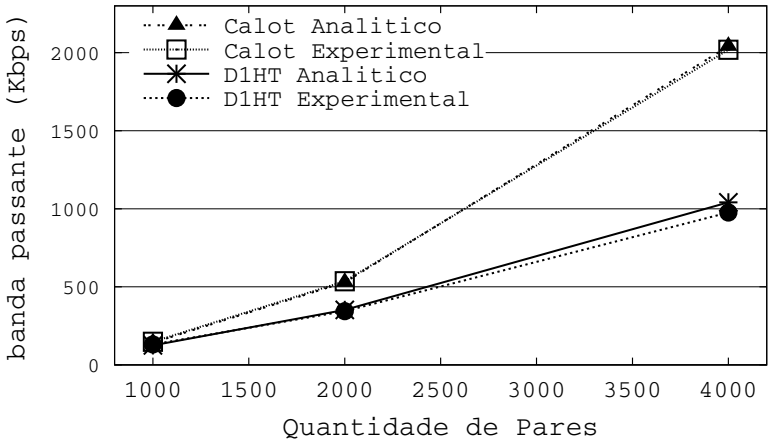


Figura 8.3: Demandas de rede de manutenção de tabelas de roteamento (em kbps) de D1HT e 1h-Calot com tamanho médio de sessão de 174 minutos (representativo de Gnutella). São mostradas as demandas medidas nos experimentos com as implementações reais dos dois sistemas, e aquelas previstas através de suas análises quantitativas teóricas.

nó atingiu 100% de uso de CPU ao servir consultas de 1.600 pares. Em virtude disto, nós selecionamos um nó dedicado do Cluster F para a execução de Dserver.

A MHDHT avaliada foi Chimera [10], não somente por que trata-se de uma implementação de Pastry [83], uma das mais proeminentes MHDHTs já propostas, como também por que Chimera não requer a instalação de nenhum pré-requisito nos multicomputadores de produção que utilizamos (p.ex., Java ou Python). Uma vez que Chimera usa o algoritmo de Pastry com base 4, espera-se que seja capaz de resolver as consultas com até $log_4(n)$ saltos.

De maneira a verificarmos preliminarmente se as latências dos quatro sistemas avaliados poderiam diferir devido a questões de implementação, executamos os quatro sistemas com apenas dois pares dedicados, e todas as latências de um salto medidas foram similares e próximas a 0,14 ms.

Ao contrário dos custos de manutenção de tabelas de roteamento, as latências das consultas são significativamente sensíveis à carga imposta ao ambiente experimental. Assim, para evitar possíveis interferências em nossos resultados, todos os experimentos de latência foram conduzidos em 400 nós dedicados do Multicomputador A, e foram medidas as latências das consultas tanto com os nós ociosos quanto sob carga total de uso de CPU, através da execução de duas instâncias do programa burnP6 [69] por nó.

Uma vez que os nós estavam dedicados para os nossos experimentos, pudemos aumentar a taxa de consultas executada por cada par na segunda fase dos experimentos para 30 consultas/segundo, de maneira a poder investigar as latências sob cargas mais desafiadoras. Os pares D1HT e 1h-Calot foram submetidos a taxas de eventos representativas

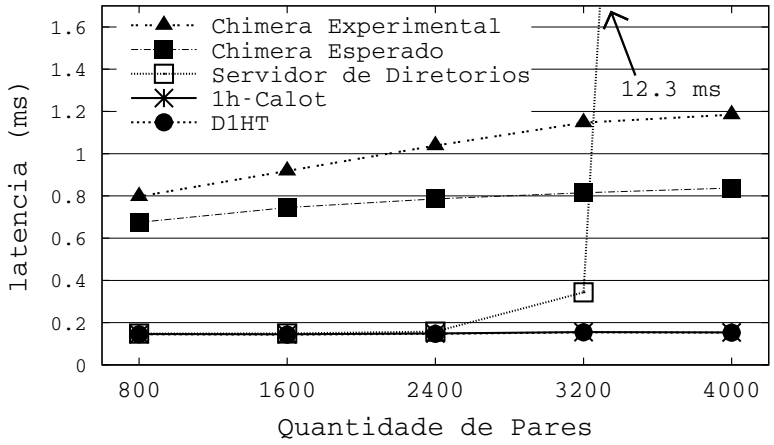


Figura 8.4: Latências médias das consultas medidas nos experimentos conduzidos no ambiente HPC com os servidores ociosos. A figura apresenta as latências medidas para D1HT, 1h-Calot, Dserver e Chimera, com até 4.000 pares distribuídos por 400 servidores ociosos do Multicomputador A. No gráfico são também mostradas as latências estimadas para Chimera.

de sistemas com duração média de sessão de 174 minutos nas duas fases de todos os experimentos. Por outro lado, os pares Chimera e clientes Dserver não sofreram quaisquer eventos, ou seja, os sistemas Chimera e Dserver não foram submetidos a adesões e partidas de pares e clientes durante as nossas medições.

De maneira a permitir o estudo de sistemas maiores, nós artificialmente executamos até dez instâncias de cada sistema avaliado por nó, mesmo sabendo que este alto nível de concorrência poderia afetar os resultados medidos (como veremos adiante). Por outro lado, como dispúnhamos de uma janela de tempo limitada, em cada experimento nós executamos os quatro sistemas concorrentemente, variando a quantidade de pares de cada sistema de dois até dez por nó, de maneira a estudar sistemas com 800, 1.600, 2.400, 3.200 e 4.000 pares (ou, para sermos mais precisos, clientes para o caso de Dserver). Por exemplo, os experimentos com tamanho de sistema de 2.400 pares foram obtidos executando-se concorrentemente seis pares D1HT, seis pares 1h-Calot, seis pares Chimera e seis clientes Dserver em cada um dos 400 nós dedicados do Multicomputador A.

As Figuras 8.4 e 8.5 mostram, respectivamente, as latências medidas para os diversos sistemas com nós ociosos e sob carga total de CPU. Uma vez que as latências medidas de Chimera foram maiores do que as esperadas, na Figura 8.4 nós também mostramos as latências calculadas para Chimera assumindo que cada consulta requer $log_4(n)$ saltos, com cada salto durando 0,14 ms. Acreditamos que as diferenças entre as latências medidas e esperadas de Chimera se devam a questões de implementação, mas mesmo as suas latências esperadas são significativamente superiores que as medidas para Dserver (com os menores sistemas) e as SHDHTs (para todos os tamanhos de sistema), o que indica

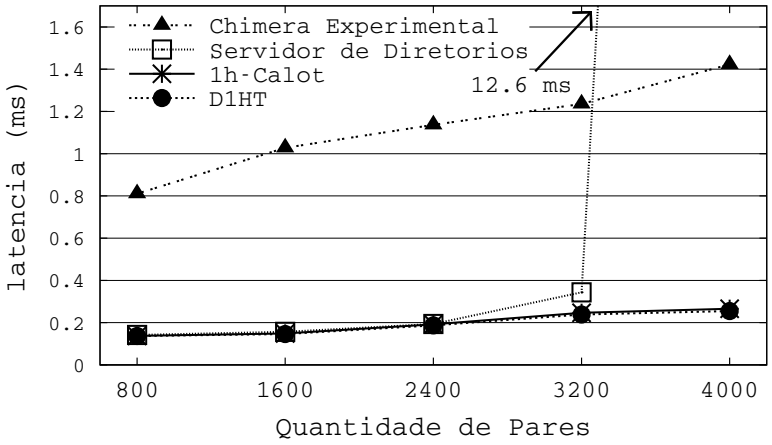


Figura 8.5: Latências médias das consultas medidas nos experimentos conduzidos no ambiente HPC com servidores sob carga total de processamento. A figura apresenta as latências medidas para D1HT, 1h-Calot, Dserver e Chimera, com até 4.000 pares distribuídos por 400 servidores do Multicomputador A, com a execução simultânea de duas instâncias do programa burnP6 por servidor, de maneira a forçar uso de 100% de CPU.

que este tipo de solução não é adequada para aplicações sensíveis à latência. Mesmo que as latências de Chimera possam ser reduzidas com uso de bases maiores (p.ex., 16), seu desempenho continuaria sendo sensivelmente inferior ao das SHDHTs.

Das latências apresentadas nestas figuras, podemos observar que, com exceção de Chimera, todos os outros sistemas apresentaram latências semelhantes para os menores tamanhos de sistema (até 2.400 pares), o que já era esperado uma vez que D1HT resolvem mais de 99% das consultas com um único salto, enquanto Dserver resolve todas as consultas com um salto e tem código similar aos das SHDHTs. Mas Dserver começa a enfrentar o seu limite de escalabilidade já com 3.200 pares, com latências 120% e 40% superiores às de D1HT com, respectivamente, nós ociosos e sob carga total de processamento. Com 4.000 pares Dserver apresentou latências que são uma ordem de magnitude maiores do que as providas por D1HT, o que mostra claramente a limitação de escalabilidade deste tipo de solução Cliente/Servidor. Mesmo que a capacidade de Dserver possa ser melhorada com uso de um servidor mais bem provisionado de recursos, ou mesmo através da distribuição da carga das consultas por um conjunto de servidores dedicados, isto, além de tornar esta solução ainda mais onerosa e complexa, iria apenas postergar o seu limite de escalabilidade. Devemos ressaltar que, enquanto a nossa avaliação foi realizada com até 4.000 pares, sistemas distribuídos reais podem chegar a centenas de milhares, ou mesmo milhões, de participantes.

Comparando-se as latências apresentadas nas Figuras 8.4 e 8.5, podemos observar que, como esperado, a carga de CPU nos nós trouxe degradação das latências de todos

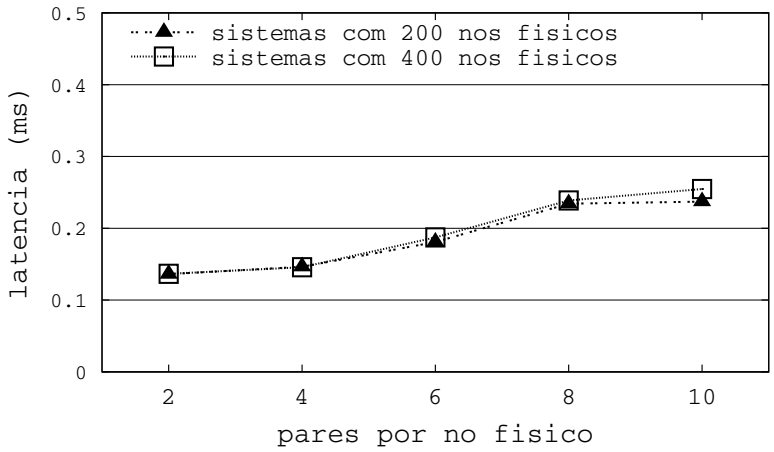


Figura 8.6: Latências médias das consultas de D1HT medidas nos experimentos conduzidos no ambiente HPC com 200 e 400 servidores sob carga total de processamento.

os sistemas avaliados. Inesperadas porém, foram as degradações das latências de D1HT, 1h-Calot e Dserver (este último até 2.400 clientes) observadas com o aumento do tamanho do sistema e servidores sob alta carga de processamento, como mostrado na Figura 8.5. De modo a verificar se este comportamento inesperado estava relacionado com o artificial aumento de carga gerada pela execução de até uma dezena de instâncias de cada sistema nos nós já sob carga total de processamento, nós executamos estes mesmos experimentos com 200 nós, também variando o número de instâncias de cada sistema entre duas e dez por nó, todas concorrentes com duas instâncias do programa burnP6. Estes resultados são apresentados na Figura 8.6 junto com os resultados originais medidos com 400 nós, de maneira a permitir a comparação das latências medidas com o mesmo nível de concorrência por nó, independente do tamanho total do sistema. Deve-se frisar que, apesar de nos experimentos com 200 nós termos também executados os quatro sistemas concorrentemente, nesta figura apresentamos somente os resultados medidos de D1HT, de maneira facilitar a sua visualização e nossas conclusões.

Confirmando a nossa hipótese, os resultados apresentados na Figura 8.6 indicam que a degradação de latência observada está altamente relacionada com o nível de concorrência de pares por nó (e a consequente sobrecarga causada nos nós já sob 100% de uso de CPU), já que as latências medidas com 200 e 400 nós e a mesma quantidade de pares por nó foram muito próximas, mesmo que os sistemas com 400 nós físicos tivessem o dobro do tamanho total. Por exemplo, com quatro pares por nó, as latências de D1HT medidas com 200 nós (sistema com total de 800 pares) e 400 nós (sistema com total de 1.600 pares) foram ambas iguais a 0,15 ms. Com oito pares por nó, as latências com 200 nós (sistema com total de 1.600 pares) e 400 nós (sistema com total de 3.200 pares)

foram, respectivamente, 0,23 ms e 0,24 ms. Estes resultados indicam que as latências das consultas em D1HT não devem variar com o tamanho do sistema, mas podem degradar com o seu uso em pares sobrecarregados. Deve-se notar no entanto que, como pode ser observado na Figura 8.5, mesmo com os nós sob alta carga de processamento, as latências de D1HT deverão ser similares (para sistemas pequenos) ou melhores (para sistemas médios ou grandes) do que as providas por Dserver, já que o desempenho desta solução Cliente/Servidor também apresentou degradação nos experimentos com nós sob alta carga de processamento.

Deste modo, os resultados que apresentamos nesta seção indicam que, independente da carga de processamento nos nós, D1HT tem escalabilidade superior à de Dserver, proporcionando latências similares para sistemas pequenos, e significativamente melhores para sistemas médios e grandes.

8.2 Resultados Analíticos

Nesta seção serão apresentados resultados analíticos que quantificam as demandas necessárias para manutenção das tabelas de roteamento em D1HT, e estes resultados serão comparados com aqueles calculados para os sistemas OneHop e 1h-Calot. Como discutido na Seção 2.2, os resultados de 1h-Calot que serão apresentados nesta seção são também válidos paras as SHDHTs 1HS e SFDHT. Iniciaremos com a descrição da metodologia utilizada, para então apresentarmos os resultados analíticos dos três sistemas estudados. Em seguida, na Seção 8.2.3, apresentaremos os ganhos obtidos com uso do mecanismo de Quarentena introduzido nesta tese.

8.2.1 Metodologia

A avaliação que será apresentada nesta seção é baseada em resultados analíticos dos três sistemas estudados. Os resultados de D1HT foram obtidos com as equações apresentadas no Capítulo 4, sendo que para o estudo do mecanismo de Quarentena consideramos os valores de q, r_q e ρ_q, como definidos no Capítulo 5, ao invés de n, r e ρ. Deve-se notar entretanto que, a menos de quando explicitamente mencionado, os resultados de D1HT apresentados neste capítulo não consideram o uso de Quarentena.

Como em trabalhos anteriores (p.ex., [23, 29]), para D1HT e OneHop usamos $f = 1\%$. Como discutido na Seção 4.3, todos os resultados de D1HT consideram o valor conservador de 0,250 segundos para o atraso médio das mensagens (i.e., $\delta_{avg} = 0,250$ seg), que já incorpora as perdas médias causadas por retransmissões.

Parm.	D1HT	OneHop	1h-Calot	Descrição
v_m	40	40	48	Cabeçalho das mensagens de manutenção
v_a	36	36	36	Cabeçalho das confirmações de entrega
v_h	–	–	36	Cabeçalho dos *heartbeats*
b	4	4	–	Descrição de cada evento

Tabela 8.2: Tamanhos (em octetos) usados para as descrições de eventos (b) e cabeçalhos de mensagens. Os tamanhos dos cabeçalhos de mensagens já incluem 28 octetos de maneira a considerar os protocolos de rede (20 octetos para protocolo IPv4) e de transporte (8 octetos para protocolo UDP).

Os resultados analíticos de 1h-Calot foram obtidos com a Equação 8.1, que não considera atrasos nas mensagens. Para OneHop serão apresentados resultados calculados segundo sua análise quantitativa publicada em [23], que também não considera atrasos de mensagens, além de assumir parâmetros ótimos para sua topologia (p.ex., tamanhos de unidades e blocos) e desconsiderar os problemas e custos acarretados por falhas em líderes de unidades e de blocos. Em contraste, todos os resultados que serão apresentados para D1HT são baseados em propriedades formalmente enunciadas e provadas, e consideram a existência de atrasos nas mensagens e falhas de qualquer tipo de nó.

Os resultados dos três sistemas foram calculados com os tamanhos de descrição de eventos e cabeçalhos de mensagens como apresentados na Tabela 8.2, sendo que para D1HT e 1h-Calot usamos os valores discutidos nas Seções 6.5 e 8.1.1, respectivamente. Uma vez que OneHop também faz agregação de eventos, para este sistema consideramos os mesmos formatos das mensagens de D1HT.

É importante ressaltar que os parâmetros apresentados na Tabela 8.2 são realistas, uma vez que refletem os formatos das mensagens usadas nas nossas implementações reais de D1HT e 1h-Calot. Por outro lado, diversos outros estudos de DHTs (e.g., [23, 29, 42, 43, 44]), incluindo a avaliação analítica apresentada pelos autores de OneHop, foram baseados em tamanhos irreais, uma vez que consideraram um custo fixo máximo de 20 octetos para todos os cabeçalhos de mensagens, enquanto só os cabeçalhos dos protocolos de rede e transporte requerem, no mínimo, 28 octetos.

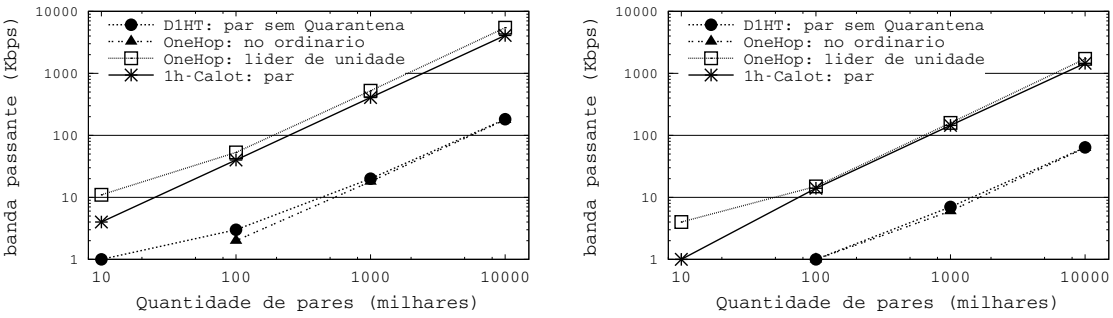
Assim como para os nossos resultados experimentais, em nossos estudos analíticos nós só computamos os custos para manutenção das tabelas de roteamento (incluindo detecção de falhas), consideramos que todos os eventos e destinos das consultas são aleatoriamente espalhados pelo anel de identificadores, e as taxas de eventos foram calculadas a partir dos tamanhos médios de sessão com uso da Equação 4.5 (ou Equação 5.2 para D1HT com Quarentena).

Avaliamos os três sistemas segundo quatro durações médias de sessão (i.e., S_{avg}) distintas. Os valores usados foram 780, 174, 169 e 60 minutos, sendo que os três primeiros foram medidos em estudos sobre BitTorrent [3], Gnutella [85] e KAD [93], respectivamente, e o menor tamanho de sessão foi usado de maneira a avaliar as três SHDHTs segundo um cenário ainda mais dinâmico. Além de ser representativo de aplicações P2P largamente difundidas na Internet, este conjunto de tamanhos médios de sessão é mais amplo do que os usados nas demais avaliações já publicadas de DHTs (p.ex., [23, 29, 42, 44, 53, 55, 56]). Além disto, estudamos as três SHDHTs com tamanhos de sistema variando de 10^4 até 10^7 de maneira a avaliar os seus custos com dimensões que sejam representativas tanto de grandes ambientes HPC como de aplicações P2P populares.

Deve-se ressaltar que enquanto resultados que sejam obtidos com simulações ou implementações reais são considerados mais representativos para avaliações de novas propostas, para estudos de sistemas P2P os resultados analíticos têm valor especial, uma vez que permitem o estudo de sistemas com tamanhos arbitrariamente grandes. Este aspecto é relevante uma vez que não é viável implementar (ou mesmo simular) sistemas com milhões de pares com o único intuito de validar uma proposta de pesquisa. Na verdade, com exceção dos resultados preliminares de D1HT publicados em [56] e dos experimentos apresentados nesta tese, os demais resultados experimentais comparativos com implementações reais de sistemas DHT publicados ficaram restritos a duas centenas de nós físicos (p.ex., [18, 23, 35, 74, 75, 105]), enquanto as simulações se restringiram a um máximo de 20 mil nós virtuais (p.ex., [18, 23, 29, 30, 35, 42, 43, 44, 53, 70, 84, 96, 99, 105]). Desta maneira, com exceção dos experimentos que realizamos ao longo do trabalho desta tese, os estudos comparativos baseados em implementações reais de DHTs já publicados não são representativos nem mesmo de ambientes HPC, que usualmente têm milhares de nós [100]. Além disto, mesmo os resultados das simulações não foram capazes de refletir a realidade de sistemas P2P populares, que são capazes de suportar milhões de usuários. Por outro lado, acreditamos que para serem aceitos como estimativas válidas, os resultados analíticos devem ser baseados em propriedades formalmente e experimentalmente comprovadas, usar parâmetros realistas, e considerar problemas importantes que possam ser enfrentados por implementações reais (como atrasos em mensagens), como é o caso dos resultados analíticos que são apresentados para D1HT neste capítulo.

8.2.2 Análise Comparativa

Nesta seção apresentaremos as demandas de banda passante de saída para manutenção de tabelas de roteamento para D1HT, OneHop e 1h-Calot. Serão comparadas as demandas médias de um par D1HT (sem Quarentena) contra aquelas de um par 1h-Calot, e o melhor



(a) S_{avg}=60 min.

(b) S_{avg}=169 min (representativo de KAD).

(c) S_{avg}=174 min (representativo de Gnutella).

(d) S_{avg}=780 min (representativo de BitTorrent).

Figura 8.7: Demandas analíticas de banda passante de manutenção das tabelas de roteamento de D1HT (sem Quarentena), 1h-Calot e OneHop, para diferentes durações médias de sessão (S_{avg}) e tamanhos de sistema variando entre 10^4 e 10^7 pares. Ambos os eixos de todos os gráficos são logarítmicos. Não são mostradas demandas inferiores a 1 kbps.

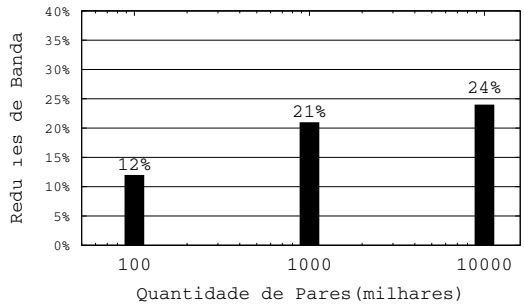
(nó comum) e o pior (líder de unidade) casos de OneHop.

Na Figura 8.7 apresentamos gráficos mostrando as demandas de banda passante de manutenção analíticas das três SHDHTs avaliadas para todos os tamanhos de sistema e durações médias de sessão estudadas.

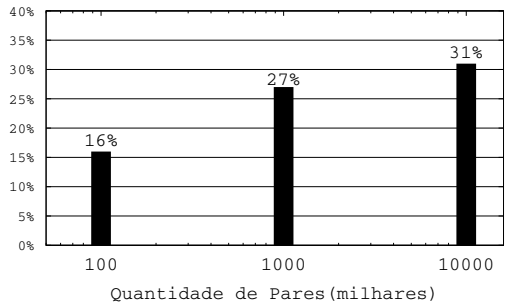
Destes gráficos, podemos nitidamente observar que a estratégia hierárquica de OneHop impõe altos níveis de desbalanceamento de carga entre os nós ordinários e líderes de unidades. Além disto, pares D1HT têm custos que são próximos aos dos nós ordinários de OneHop, e tipicamente uma ordem de magnitude menores que os dos líderes de unidades.

A Figura 8.7 mostra ainda que as demandas de manutenção médias por par em 1h-Calot são pelo menos 200% e tipicamente uma ordem de magnitude maiores do que em D1HT, para os casos estudados.

As demandas médias de manutenção por par em D1HT sem Quarentena para sistemas com um milhão de pares com sessões médias de 60, 169, 174 e 780 minutos são, respectivamente, 21 kbps, 7,3 kbps, 7,1 kbps e 1,6 kbps. Para sistemas com 10 milhões



(a) Ganhos de Quarentena com comportamento KAD (q=0,76n).



(b) Ganhos de Quarentena com comportamento Gnutella (q=0,69n).

Figura 8.8: Gráficos mostrando as reduções analíticas dos custos de manutenção de tabelas de roteamento proporcionadas por Quarentena para sistemas D1HT com diferentes tamanhos, e dinâmicas de pares representativas de KAD (Figura 8.8(a)) e Gnutella (Figura 8.8(b)).

de participantes, estas demandas crescem para 183 kbps, 65 kbps, 63 kbps e 14 kbps. Em contraste, os custos médios por par 1h-Calot e líderes de unidade de OneHop para sistemas com $n = 10^6$ e comportamento similar ao de KAD são superiores a 140 kbps, crescendo para mais de 1400 kbps para sistemas com 10 milhões de pares.

8.2.3 Avaliação de Quarentena

Nesta seção desenvolveremos uma avaliação analítica do mecanismo de Quarentena apresentado no Capítulo 5. Os resultados aqui apresentados foram obtidos com a análise introduzida no Capítulo 4 e complementada no Capítulo 5. Deste modo, para cálculo dos resultados analíticos que apresentaremos nesta seção, usaremos as Equações 5.1 e 5.2 ao invés de, respectivamente, as Equações 4.1 e 4.5, para assim refletir o efeito trazido por Quarentena.

Os resultados analíticos de Quarentena serão baseados em estudos que observaram que 31% das sessões em Gnutella [11] e 24% das sessões em KAD [93] duraram menos de 10 minutos, o que é um valor conveniente para o período de Quarentena T_q. Ou seja, iremos apresentar resultados para sistemas D1HT em que o uso de Quarentena faz com que novos pares tenham que resolver suas consultas com dois saltos até que completem 10 minutos de conexão, para então serem inseridos no anel e terem sua adesão informada a todo o sistema, e assim estarem aptos a resolver suas consultas com um único salto.

Uma vez que os estudos publicados em [11] e [93] observaram também que os tamanhos médios de sessão para Gnutella e KAD são, respectivamente, 174 minutos e 169 minutos, os custos de manutenção com uso de Quarentena foram calculados assumindo-se $q = 0,69 \cdot n$ e $S_{avg} = 174$ minutos para sistemas com comportamento similar ao de Gnu-

tella, e $q = 0,76 \cdot n$ e $S_{avg} = 169$ minutos para os sistemas com dinâmica representativa de KAD.

As reduções de banda passante de manutenção assim calculadas são apresentadas na Figura 8.8, para tamanhos de sistema variando entre 10^5 e 10^7. Podemos ver que as reduções trazidas por Quarentena crescem com o tamanho do sistema, já que quanto menor estes forem, maior será o custo relativo das mensagens com $TTL = 0$, que não são evitadas com Quarentena.

Os resultados apresentados na Figura 8.8 também indicam que o nosso mecanismo de Quarentena é efetivo para redução dos custos associados a pares voláteis, com reduções de 21% e 27% para sistemas com um milhão de pares e dinâmicas similares a, respectivamente, KAD e Gnutella, sendo que estas reduções atingem 24% e 31% para sistemas com 10 milhões de pares.

Capítulo 9

Avaliação Experimental de D1HF

Neste capítulo apresentaremos de modo objetivo o conjunto de resultados experimentais obtidos com D1HF que, assim como os resultados de D1HT que foram apresentados no Capítulo 8, serão discutidos no Capítulo 10. Entre os resultados que apresentaremos neste capítulo, devemos destacar a avaliação experimental do desempenho de D1HF em relação a um servidor de arquivos de última geração, bem como os resultados que nos permitiram afirmar que as consultas encaminhadas a D1HT foram resolvidas de maneira rápida o suficiente para não trazer qualquer prejuízo ao desempenho de D1HF.

Iniciaremos com a descrição da metodologia utilizada, onde discutiremos os padrões de acesso avaliados, que são representativos das aplicações paralelas discutidas na Seção 7.1, e descreveremos o ambiente experimental utilizado, que inclui 800 nós de um multicomputador Beowulf e um sistema de arquivos de rede de última geração. Em seguida, apresentaremos algumas medições que foram feitas com objetivo de caracterizar aspectos relevantes do desempenho dos nós computacionais utilizados. Na Seção 9.3 discutiremos resultados que nos permitiram entender melhor os padrões de acesso utilizados, bem como quantificar as reduções de carga no Sistema BASE proporcionadas pelo uso de D1HF. Na última seção deste capítulo apresentaremos nossos principais resultados comparativos de desempenho obtidos com D1HF, que mostraram que, para um dos padrões de acesso, este sistema alcançou desempenho até oito vezes superior ao do sistema de arquivos comercial utilizado. Ao final apresentaremos medições de latência que demonstram que o uso de D1HT permitiu a rápida localização dos blocos dos arquivos para todos os tamanhos de sistema estudados, tendo portando um papel importante na escalabilidade de D1HF.

Padrão de acesso	**SEQUENCIAL**	**SALTEADO**
Tipo dos acessos	somente leitura	somente leitura
Tamanho cada operação de leitura	15 KB (fixo)	15 KB (fixo)
Reposicionamento médio (`lseek`)	0	150 KB
Duração de cada experimento	leitura total do arquivo	30 min
Quantidade de arquivos de entrada	1	35 mil
Tamanho médio do(s) arquivo(s)	4 GB	40 MB
Soma dos tamanhos dos arquivos	4 GB	1.400 GB
Tamanho de bloco D1HF	20 MB	20 MB
Quantidade total de blocos	200	70 mil

Tabela 9.1: Parametrização dos nossos experimentos com o programa DHFbench (apresentado na Seção 9.1), de maneira a avaliar D1HF e o Sistema BASE segundo padrões de acesso representativos das aplicações paralelas discutidas na Seção 7.1. Na tabela é também mostrado o tamanho de bloco usado nos experimentos, um importante parâmetro de D1HF (como discutido na Seção 7.11), sendo que todos os experimentos foram realizados com o mesmo tamanho para todos os blocos.

9.1 Metodologia

Nesta seção iremos apresentar a metodologia usada nos experimentos conduzidos para avaliação de D1HF. Para viabilizar estes estudos, foi desenvolvido um programa, que chamaremos de DHFbench, que nos permite gerar diversos padrões de leituras distintos segundo os parâmetros selecionados. Este programa não gera nenhum arquivo de saída, de maneira que o seu desempenho é um resultado direto da velocidade de leitura do(s) seus(s) arquivo(s) de entrada, que, assim como diversos outros parâmetros, poderá(ão) ser escolhido(s) de maneira a procurar reproduzir diferentes padrões de acesso.

Nos nossos experimentos, DHFbench foi usado com parametrização que será discutida a seguir, de maneira a permitir o estudo de dois padrões de acesso de leitura distintos, SEQUENCIAL e SALTEADO, que são representativos das aplicações paralelas de imageamento sísmico discutidas na Seção 7.1. Na verdade, o desenvolvimento de DHFbench foi motivado não só pela necessidade de uma ferramenta parametrizável para reprodução de diferentes padrões de acesso, como também para permitir a avaliação de D1HF em modo ligado através da recompilação do seu código. Os resultados que serão apresentados e discutidos na Seção 9.2 indicam que o uso de um programa próprio não causou distorções nos resultados que serão apresentados neste capítulo.

As principais características dos padrões de acesso avaliados são apresentadas na Tabela 9.1. Os parâmetros mostrados nesta tabela são coerentes com as aplicações reais usadas em ambientes HPC, com exceção da duração do experimento, uma vez que as aplicações paralelas têm normalmente tempos de execução que podem chegar a dias ou

semanas, que são simplesmente inviáveis para avaliações experimentais como as que conduzimos. De qualquer maneira, uma vez que os ganhos de desempenhos providos por D1HF advém dos acessos a cópias cache dos dados de entrada, o uso de durações de experimentos pequenas nos leva a resultados de desempenho conservadores, uma vez que no ambiente de produção as diversas cópias dos dados poderão ser potencialmente reusadas mais vezes.

A Tabela 9.1 também mostra o tamanho de bloco usado para D1HF nos experimentos, além das quantidades totais de blocos para os dois padrões de acesso (que nada mais são do que, aproximadamente, a soma do tamanho dos arquivos de cada padrão dividida pelo tamanho do bloco). Apesar de, como discutido na Seção 7.11, a implementação atual de D1HF permitir dois tamanhos de bloco distintos, todos os experimentos aqui apresentados foram feitos com um único tamanho para todos os blocos, devida a limitada disponibilidade de recursos para realização de medidas de desempenho que nos permitissem o ajuste dos tamanhos dos blocos.

Na avaliação do padrão SEQUENCIAL, em cada experimento todos os processos lêem totalmente um mesmo arquivo sequencialmente a partir de seu início. Desta maneira, este padrão avaliado difere ligeiramente do padrão sequencial definido na Seção 7.1, no qual cada processo inicia sua leitura a partir de um ponto distinto do arquivo. Optamos por forçar o mesmo início para todos os arquivos de modo a maximizar a concorrência pelas cópias dos blocos, de maneira a tornar o experimento mais desafiador para D1HF, além de nos permitir uma avaliação preliminar das contenções que podem vir a ser evitadas com a criação de várias cópias de cada bloco (mecanismo discutido na Seção 7.5, mas que ainda não foi implementado).

Na avaliação do padrão SALTEADO, cada processo repete por 30 minutos iterações que consistem em abrir um arquivo escolhido aleatoriamente, lendo-o com um tamanho de salto randômico entre 0 KB e 300 KB, o que reproduz, de modo próximo, o padrão de acesso salteado observado em aplicações paralelas de processamento sísmico, como discutido na Seção 7.1.

Os experimentos foram conduzidos usando-se até 800 nós físicos dedicados do Multicomputador A, apresentado na Tabela 8.1 da Seção 8.1.2. Neste multicomputador, cada bastidor tem dois comutadores Ethernet TOR não bloqueantes e 37 nós computacionais, sendo que cada nó tem uma conexão 1000baseT a um destes dois comutadores. Cada comutador TOR tem duas conexões 1000baseSX ao nível superior da árvore gorda, composto por um único comutador Ethernet não bloqueante, resultando assim em um nível de contenção aproximado de 9:1 para este multicomputador.

Cada nó do Multicomputador A usa Linux 2.6.9 e tem dois processadores com arqui-

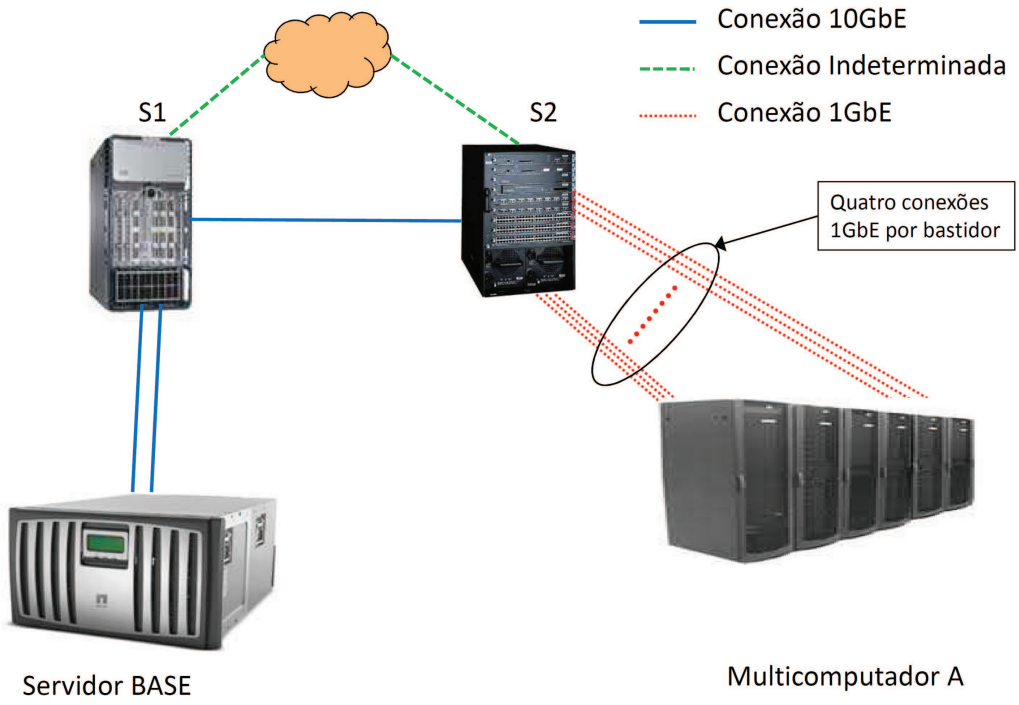


Figura 9.1: Conexões de rede do ambiente de avaliação experimental de D1HF. Cada bastidor do Multicomputador A tem 37 servidores e quatro conexões 1GbE ao comutador S2, resultando em um nível de contenção 9:1. A banda passante de comunicação entre o Multicomputador A e o Servidor BASE é limitada pela conexão 10GbE entre os comutadores S1 e S2. Além das conexões ilustradas na figura, os comutadores S1 e S2 recebem diversas outras conexões.

tetura x86 32 bits, 2GB de memória principal DDR, e um disco IDE 5400 rpm. O único disco interno de cada nó é usado para alojar o sistema operacional, os arquivos de paginação e a área para armazenar as cópias de blocos de arquivo geradas por D1HF. Na Seção 9.2 iremos apresentar algumas avaliações experimentais do desempenho das conexões de rede e dos discos internos dos nós deste multicomputador.

À época dos experimentos, os servidores GPFS deste ambiente estavam sob alta carga de produção, de maneira que não era apropriado o seu uso para as nossas avaliações, uma vez que nossos experimentos poderiam afetar a produção HPC corrente, e vice versa. Por outro lado, tínhamos disponível um poderoso servidor NFS recém-adquirido. Assim, o Sistema BASE usado em nossos experimentos foi baseado no padrão NFSv3 provido por um moderno servidor de arquivos (Servidor BASE), que representa o estado da arte em termos de servidores NFS comerciais para ambientes de produção corporativos, com configuração que inclui 32GB de memória principal, 64GB de memória cache para dados, e 640 discos FC de 15Krpm.

A Figura 9.1 ilustra a conexão do Servidor BASE e dos diversos bastidores do Mul-

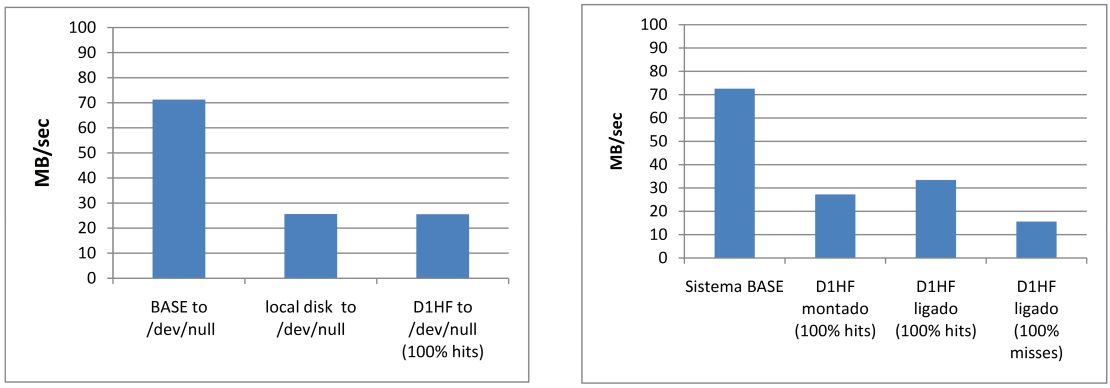
ticomputador A. Deve-se notar que o Servidor BASE tem duas conexões 10GbE, que são usadas para suportar a carga gerada por diversos multicomputadores, incluindo todos aqueles listados na Tabela 8.1, mas o desempenho de sua comunicação com o Multicomputador A está limitado pela conexão 10GbE entre os comutadores Ethernet S1 e S2.

Em cada experimento avaliamos um padrão de acesso (SEQUENCIAL ou SALTEADO), com um tamanho determinado de sistema (1, 100, 200, 400 ou 800 nós computacionais), e um sistema de arquivos (D1HF ou Sistema BASE). A menos de quando especificado em contrário, todas as cópias de blocos residentes nos caches D1HF locais aos nós utilizados foram removidas antes de cada experimento. Além disto, antes de cada experimento executou-se em cada nó um programa para alocação e inicialização de um vetor com o tamanho de sua memória principal (2 GB), de maneira a forçar o sistema operacional a eliminar cópias de blocos que estejam residentes na memória principal.

Todos os experimentos apresentados neste capítulo foram realizados com nós computacionais dedicados. Apesar do Servidor BASE não ter sido usado em modo dedicado, houve pouca concorrência por seus recursos, uma vez que à época dos experimentos este servidor ainda estava recém instalado, e sendo colocado paulatinamente em produção. Por outro lado, os comutadores S1 e S2, que recebem diversas outras conexões além daquelas ilustradas na Figura 9.1, foram compartilhados com a produção normal do CPD. Deste modo, procurando isolar possíveis interferências causadas nos nossos resultados por eventuais compartilhamentos de recursos, cada experimento foi executado cinco vezes, e foram descartados os resultados que fossem mais de 15% inferiores à medição que apresentou o melhor desempenho, sendo que os resultados que serão apresentados são a média aritmética das medições válidas (i.e., a média das medições que não foram descartadas). Nos poucos casos onde foram descartadas mais de duas das cinco primeiras medições de um determinado experimento, este foi repetido até termos pelo menos três medições consideradas válidas.

9.2 Caracterização do Desempenho de um Nó Computacional

Nesta seção iremos apresentar e discutir várias medições realizadas em um único nó computacional do Multicomputador A, que foram feitas de maneira a caracterizar o desempenho do seu disco interno, a velocidade de acesso ao Sistema BASE e possíveis diferenças de desempenho entre o uso de D1HF ligado e montado. Deve-se notar, entretanto, que não é objetivo de D1HF acelerar o desempenho de leitura de um único nó, e sim a banda



(a) Comando `dd of=/dev/null`

(b) Programa DHFbench dom padrão SEQUENCIAL.

Figura 9.2: Medições de velocidade de leitura, em MB/sec, feitas em um nó do Multicomputador A. A Figura 9.2(a) mostra as velocidades medidas com o comando `dd` lendo um arquivo com tamanho de 4GB direto do Sistema BASE, através de D1HF montado, e direto do disco local do nó, sempre direcionando a sua saída para o dispositivo lógico `/dev/null` (i.e., parâmetro `of=/dev/null`). A Figura 9.2(b) mostra velocidades medidas com o programa DHFbench executando segundo o padrão SEQUENCIAL com uso de D1HF (montado e ligado) e direto do Sistema BASE. Os resultados de D1HF com 100% *hits* representam medições com todos os acessos sendo satisfeitos a partir do disco local, enquanto para os resultados com 100% *misses* todos os acessos implicaram na realização de cópias dos blocos a partir do Sistema BASE para o disco local.

agregada de leitura atingida por sistemas distribuídos com centenas de participantes, de maneira que é absolutamente esperado que o seu uso seja contraproducente em arquivos acessados por um único nó, especialmente por que a sua versão atual não implementa o mecanismo proposto na Seção 7.5 para minimizar os custos de acesso para arquivos que não sofram releituras. Mesmo assim, acreditamos que os resultados que mostraremos e discutiremos nesta seção sejam úteis, não só por permitirem uma melhor caracterização do ambiente usado nos nossos experimentos, como também para facilitar a análise de outros resultados que serão apresentados neste capítulo.

A Figura 9.2(a) mostra várias medições feitas com o comando Linux `dd` lendo um arquivo com tamanho de 4GB do Sistema BASE diretamente, do disco local do nó, e através de D1HF montado. Uma vez que o arquivo de saída nestes experimentos foi direcionado para o dispositivo lógico `/dev/null` do Linux, esta saída era simplesmente descartada sem gerar qualquer atividade de escrita, seja em disco ou em memória. Desta maneira, assim como para o programa DHFbench, as velocidades apresentadas nesta figura são um resultado direto do desempenho dos dispositivos usados para leitura do seu arquivo de entrada. Assim, o uso do comando `dd` nestes experimentos procura reproduzir o padrão SEQUENCIAL como parametrizado segundo a Tabela 9.1 para um único nó, de maneira a não só nos permitir avaliar a versatilidade do uso de D1HF montado de maneira transpa-

rente por aplicações e ferramentas não modificadas (como o próprio comando `dd`), como também para comparar o desempenho atingido pelo nosso programa DHFbench (como será apresentado na Figura 9.2(b)) com o do comando `dd`.

Da primeira coluna da Figura 9.2(a) podemos ver que o desempenho de leitura direta ao Sistema BASE foi na verdade contido pela capacidade da conexão 1000baseT ao comutador TOR, demonstrando que o Sistema BASE é suficientemente rápido para alimentar um único nó. A figura mostra também que o desempenho de leitura a partir do já obsoleto disco local (cerca de 25 MB/s) é quase três vezes inferior ao desempenho conseguido com o acesso direto ao Sistema BASE, sendo que esta diferença seria muito maior se o desempenho do Sistema BASE não tivesse sido contido pela conexão de rede. Por conseguinte, o desempenho de D1HF sendo usado neste único nó foi limitado pela velocidade do disco local, como mostra a similaridade das velocidades apresentadas nas duas últimas colunas desta figura.

Os resultados apresentados na Figura 9.2(a) nos levam a fazer algumas observações. Primeiro que, ao contrário da estratégia comumente utilizada por mecanismos cache, que normalmente fazem uso de dispositivos mais rápidos (em termos de latência e/ou banda passante) do que aqueles usados para armazenamento das versões originais dos dados, os dispositivos (discos locais) usados por D1HF para alojar as cópias cache dos blocos deverão ser, em geral, mais lentos do que o Sistema BASE, se considerarmos os seus desempenhos como vistos por um único nó. No entanto, devemos ressaltar que o objetivo de D1HF é o de, entre outros, acelerar o acesso a dados com alta concorrência de leitura por múltiplos nós de aplicações distribuídas, e os resultados que apresentaremos na Seção 9.4 mostrarão que este objetivo pode ser plenamente atingido mesmo que D1HF faça uso de dispositivos tão lentos quanto os discos locais do Multicomputador A. Além disto, acreditamos que a restrição de desempenho causada pelos discos locais na velocidade de leitura de D1HF como mostrada na Figura 9.2(a) deva representar uma situação de pior caso em relação ao desempenho dos discos locais de multicomputadores modernos.

Na Figura 9.2(b) mostramos várias medições feitas em um único nó com o programa DHFbench segundo o padrão SEQUENCIAL como parametrizado segundo a Tabela 9.1. Foram feitas medições lendo-se diretamente do Sistema BASE e através de D1HF montado e ligado. Para D1HF montado e ligado foram feitas medições com o cache local já completamente povoado, de maneira a permitir a sua comparação na situação em que todos os acessos são satisfeitos a partir de cópias locais dos dados (100% *cache hits*), que deverá ser o caso comum deste padrão de acesso (como será visto na Seção 9.3). Para D1HF ligado foram também feitas medições sem nenhuma cópia local dos blocos (100% *cache misses*), de maneira a nos permitir comparar o desempenho de D1HF nestas duas

situações extremas (100% *misses* e 100% *hits*).

Da comparação dos resultados obtidos com o comando `dd` e o programa `DHFbench` lendo tanto diretamente do Sistema BASE quanto através de D1HF, como mostrados nas Figuras 9.2(a) e 9.2(b), podemos ver que o programa DHFbench quando usado segundo o padrão SEQUENCIAL é tão (ou mais) eficiente quanto o comando `dd`, indicando que o uso deste programa nos nossos experimentos não deve ter gerado distorções em relação aos resultados que seriam obtidos com uso de comandos e ferramentas tradicionais (como o próprio comando `dd`).

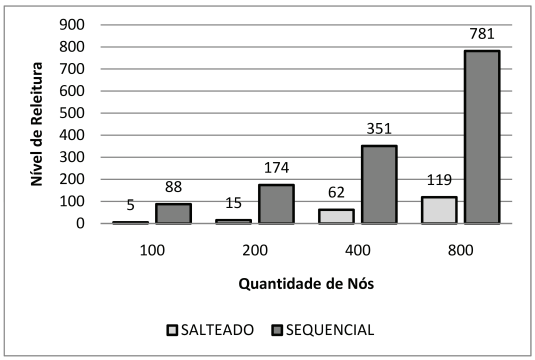
Os resultados apresentados na Figura 9.2(b) nos permitem ainda comparar o desempenho de D1HF quando usado em modos ligado e montado, além das velocidades observadas na ausência ou presença total de *cache misses*. Em relação a esta última comparação, o desempenho com 100% de *cache hits* foi, como esperado, significativamente superior. A comparação de desempenho entre os modos montado e ligado mostra uma superioridade de cerca de 20% para este último, que acreditamos ser causada pelos custos adicionais inseridos por FUSE. Iremos estudar melhor esta questão, inclusive com uso de versões mais maduras de FUSE, como as inseridas nas versões Linux 2.6.21 e posteriores.

9.3 Caracterização dos Padrões de Acesso

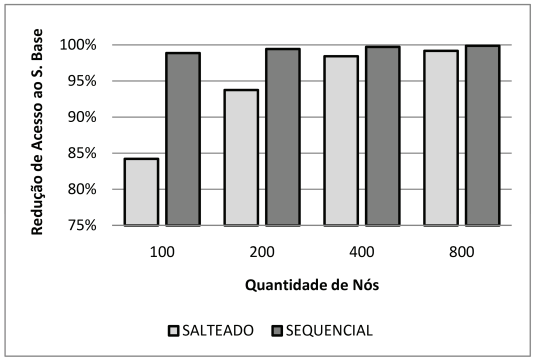
Nesta seção iremos apresentar alguns resultados que nos permitem caracterizar melhor o comportamento dos padrões de acesso estudados, quando exercitados através de execuções paralelas. Os resultados apresentados nesta seção foram obtidos executando-se uma instância do programa DHFbench por nó computacional do Multicomputador A, segundo os padrões de acesso SEQUENCIAL e SALTEADO com a parametrização apresentada na Tabela 9.1.

Na Figura 9.3(a) apresentamos as taxas de releitura dos padrões estudados, calculadas a partir de resultados medidos e com uso da Equação 7.1. Podemos ver que ambos os padrões de acesso apresentam altos níveis de releitura, como já era esperado em função do que foi discutido na Seção 7.1. Deve-se notar que o padrão SEQUENCIAL mostrou níveis de releitura muito superiores aos obtidos pelo padrão SALTEADO, o que também já era esperado em função da grande diferença entre as somas dos tamanhos dos arquivos lidos por estes padrões, como mostrado na Tabela 9.1.

Dos resultados apresentados na Figura 9.3(b) podemos observar que D1HF foi capaz de reduzir expressivamente a carga de leitura no sistema BASE utilizado, atingindo reduções superiores a 99% para ambos os padrões com os maiores tamanhos de sistema. Deste modo, os resultados apresentados nesta figura demonstram que D1HF conseguiu atingir



(a) Releituras a dados de entrada



(b) Reduções no acesso ao Sistema BASE providas por D1HF

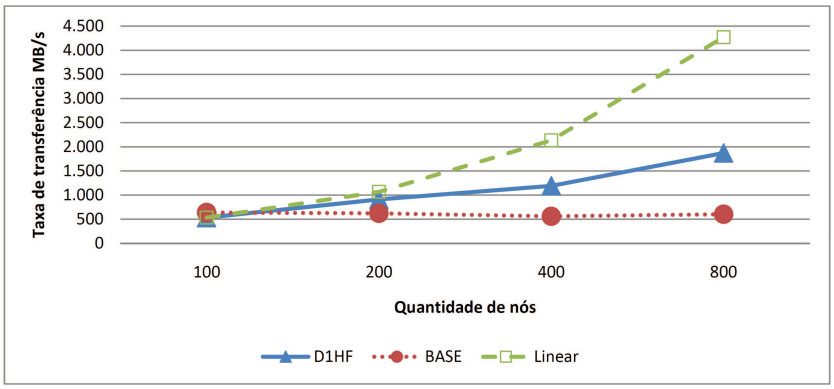
Figura 9.3: Esta figura mostra, para ambos os padrões de acesso avaliados e tamanhos de sistema entre 100 e 800 nós computacionais, os níveis de releitura aos dados de entrada (de acordo com a Equação 7.1 apresentada na página 55), bem como as reduções nos acessos ao Sistema BASE decorrentes do redirecionamento destes às cópias dos blocos criadas por D1HF.

um de seus principais objetivos, uma vez que as reduções de carga alcançadas permitem que os arquivos de entrada sejam armazenados em sistemas de discos e de arquivos comuns, mesmo que a carga total de leitura gerada pela aplicação seja altíssima. Mais especificamente, os resultados que serão apresentados na Seção 9.4 mostram que ambos os padrões de acesso podem gerar demandas de leitura superiores a 1.700 MB/s, que são desafiadoras mesmo para sistemas de arquivos paralelos, enquanto as reduções providas por D1HF, e apresentadas na Figura 9.3(b), permitiram que a carga de leitura no Sistema BASE fosse reduzida a no máximo 3 MB/s, carga esta que pode ser facilmente atendida por sistemas de arquivos NFS implantados sobre servidores comuns.

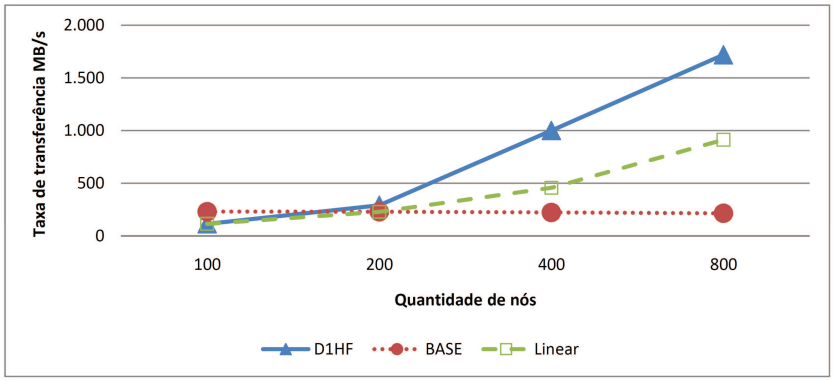
9.4 Experimentos Comparativos de Desempenho e Escalabilidade

Nesta seção mostraremos vários resultados medidos com os padrões de acesso SEQUENCIAL e SALTEADO, executando-se uma instância do programa DHFbench por nó computacional do Multicomputador A, segundo a parametrização apresentada na Tabela 9.1, variando-se o tamanho do sistema entre 100 e 800 nós.

De maneira a permitir comparações relevantes, foram feitas medições com o programa DHFbench acessando diretamente o Sistema BASE e com o uso de D1HF. Uma vez que a versão Linux 2.6.9 dos nós utilizados não incorpora nativamente o módulo kernel de FUSE, e como não foi possível instalá-lo em todos os nós do Multicomputador A previamente à execução dos nossos experimentos, todas as medições de D1HF apresentadas



(a) Padrão SEQUENCIAL



(b) Padrão SALTEADO

Figura 9.4: Banda passante agregada de leitura de arquivos, em MB/s, dos padrões de acesso SEQUENCIAL e SALTEADO, segundo o número de nós computacionais usados. Nas figuras são mostrados os resultados dos processos lendo diretamente do sistema de arquivos BASE e através de D1HF, além das curvas teóricas de crescimento linear construídas a partir dos resultados com 100 nós para D1HF.

nesta seção foram feitas no modo ligado.

A Figura 9.4 mostra os resultados que creditamos como os mais importantes da avaliação experimental apresentada neste capítulo, e que serão analisados e discutidos nos próximos parágrafos. Esta figura apresenta os resultados medidos de banda passante agregada de leitura acessando-se o Sistema BASE diretamente e através de D1HF, para os dois padrões de acesso estudados e diferentes tamanhos de sistema. Para ambos os padrões de acesso, é nítida a superior escalabilidade de D1HF, que chega a apresentar crescimento superlinear para o padrão SALTEADO, atestando o sucesso da estratégia traçada em seu projeto.

Em contraste com a expressiva escalabilidade alcançada por D1HF, podemos observar na Figura 9.4 que, para ambos os padrões de acesso, o Sistema BASE apresentou escalabilidade praticamente nula a partir do menor tamanho do sistema avaliado nos nossos

experimentos. Devemos porém ressaltar que, para o nosso ambiente experimental, acreditamos que a escalabilidade do Sistema BASE segundo o padrão SEQUENCIAL tenha sido contida pela capacidade da conexão 10GbE entre os comutadores S1 e S2 ilustrados na Figura 9.1. Por outro lado, é interessante notar que a limitação trazida por esta conexão entre os comutadores S1 e S2 não afetou o desempenho de D1HF, uma vez que ele foi capaz de explorar a localidade de acesso aos dados de maneira a satisfazer a maioria das leituras sem impor tráfego a esta conexão.

Já para o padrão SALTEADO, o desempenho do Sistema BASE foi limitado por sua própria capacidade, de maneira que, mesmo tratando-se de uma solução moderna e bem provisionada de recursos, acreditamos que o Sistema BASE avaliado não seja capaz de prover desempenho superior ao apresentado na Figura 9.4(b) para o padrão SALTEADO mesmo em outros ambientes de avaliação.

Da Figura 9.4 podemos inicialmente observar que D1HF apresentou desempenho ligeiramente inferior ao Sistema BASE para os menores tamanhos de sistema, o que não é surpreendente, já que D1HF foi projetado de modo a usufruir da localidade de acessos repetitivos que ocorre sobretudo em sistemas médios e grandes, de maneira que seus custos podem torná-lo contraproducente para ambientes de menor escala. Por outro lado, a sua superior escalabilidade permitiu que ele alcançasse os ganhos de 210% para o padrão SEQUENCIAL e 710% para o padrão SALTEADO, em relação ao Sistema BASE para os maiores tamanhos avaliados (800 nós), o que atesta experimentalmente o sucesso da estratégia utilizada. Devemos ainda ressaltar que os experimentos aqui apresentados avaliaram uma implementação preliminar de D1HF, além de não ter sido possível realizar os devidos ajustes de seus parâmetros, de maneira que acreditamos que suas futuras versões (e novas parametrizações) poderão ser mais competitivas para sistemas menores.

Embora não tenhamos dados para comprovar esta teoria, acreditamos que o crescimento superlinear de desempenho alcançado por D1HF no padrão SALTEADO tenha sido decorrente da maior disponibilidade de memória obtida com o crescimento do sistema, que aumentou a probabilidade das cópias dos blocos serem acessadas a partir da memória principal dos nós, evitando assim acessos aos lentos discos locais. Deve-se notar que, de acordo com os dados apresentados na Tabela 9.1, a soma dos tamanhos de todos os arquivos lidos segundo este padrão de acesso totaliza cerca de 1.400 GB, enquanto a capacidade total de memória do sistema avaliado nesta seção varia de 200 GB (com 100 nós) até 1.600 GB (com 800 nós).

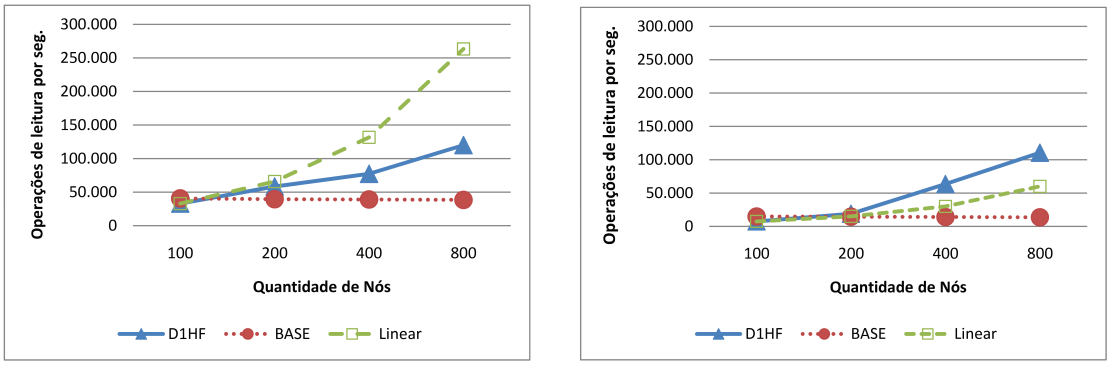
Quanto à escalabilidade apresentada por D1HF para o padrão SEQUENCIAL, devemos notar que, por um lado, ela não foi mascarada pelo aumento da quantidade de memória decorrente do crescimento do sistema, já que mesmo o menor sistema avaliado

(100 nós) dispunha de memória suficiente para armazenar todo o arquivo de entrada. Por outro lado, para este padrão de acesso a escalabilidade foi comprometida pela metodologia que escolhemos para os experimentos, já que procuramos exercitar ao máximo a contenção causada por múltiplos acessos à mesma porção do arquivo de entrada, como discutido na Seção 9.1, mesmo que a implementação atual de D1HF ainda não disponha da habilidade para criar e distribuir múltiplas cópias de cada bloco dos arquivos lidos.

O efeito da disponibilidade de memória principal para armazenamento das cópias dos blocos criados por D1HF, como discutido acima, pode nos fazer questionar a representatividade dos experimentos realizados em relação aos resultados que seriam obtidos com uso de D1HF por aplicações em produção. Quanto a esta questão, devemos notar que, ao contrário dos experimentos aqui apresentados, em produção normal haverá uma concorrência pelos recursos de memória entre D1HF e a aplicação. Por outro lado, os nossos experimentos foram conduzidos com tamanhos de problema atuais mas com nós computacionais tecnologicamente defasados. Desta maneira, se considerarmos que nós computacionais atuais tipicamente têm ao menos 24 GB de memória principal, a disponibilidade de apenas 10% desta memória já produziria um efeito maior do que a disponibilidade total da memória dos nós computacionais utilizados nos nossos experimentos. Desta maneira, acreditamos que, mesmo para ambientes de produção, D1HF poderá ser capaz de usufruir da memória principal dos nós para minimizar acessos a discos locais. Além disto, tanto o Sistema BASE aqui avaliado quanto os sistemas de arquivos comumente usados em ambientes de produção (incluindo sistemas de arquivos paralelos) também fazem uso de memória principal disponível para armazenar dados recentemente acessados, de modo que, assim como para D1HF, o seu desempenho também pode ser afetado por eventuais concorrências no uso da memória principal.

A Figura 9.5 apresenta as taxas de operações de leitura alcançadas por ambos os sistemas de arquivos avaliados segundo os dois padrões de acesso. Uma vez que todos os experimentos foram feitos com um único tamanho fixo para todas as operações de leitura (como mostrado na Tabela 9.1), os números apresentados nesta figura são um resultado direto daqueles mostrados na Figura 9.4, mas nos permitem mostrar de maneira clara as altas taxas suportadas por D1HF tanto para padrão SALTEADO quanto para o padrão SEQUENCIAL (superiores a 110 mil operações/segundo para ambos os padrões de acesso), para os maiores tamanhos de sistema.

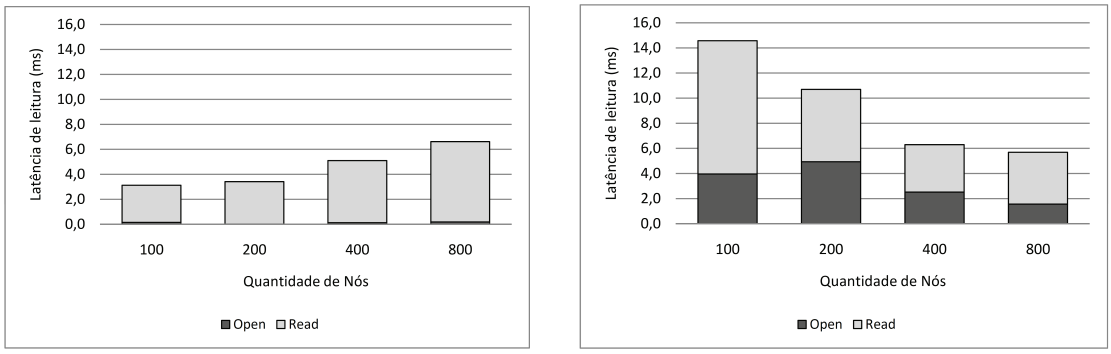
A Figura 9.6 apresenta as latências médias de leitura medidas para ambos os padrões de acesso estudados e os diferentes tamanhos de sistema. Uma vez que o objetivo do programa DHFbench é o de simplesmente ler o(s) seu(s) arquivos(s) de entrada o mais rápido possível, sem nenhum processamento ou qualquer outra atividade de entrada ou



(a) Padrão SEQUENCIAL

(b) Padrão SALTEADO

Figura 9.5: Taxas agregadas de acessos de leitura a arquivos, em operações por segundo, dos padrões de acesso SEQUENCIAL e SALTEADO, segundo o número de nós computacionais usados. Nas figuras são mostrados os resultados dos processos lendo diretamente do sistema de arquivos BASE e através de D1HF, além das curvas teóricas de crescimento linear construídas a partir dos resultados com 100 nós para D1HF.

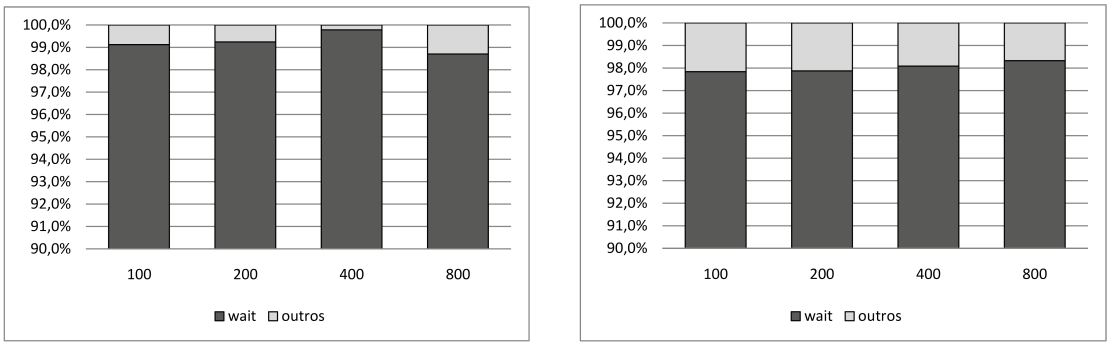


(a) Padrão SEQUENCIAL

(b) Padrão SALTEADO

Figura 9.6: Latências médias dos acessos de leitura aos arquivos com uso de D1HF, em milissegundos, para os padrões de acesso SEQUENCIAL e SALTEADO, como observadas pela aplicação. As latências mostradas incorporam tanto os tempos gastos nas operações de leitura propriamente ditas (`read`), como o custos de instauração de conexões com os *home nodes* dos diversos blocos lidos (`open`), com a distribuição ilustrada nas figuras.

saída, as latências médias mostradas nesta figura foram obtidas dividindo-se o tempo de execução total dos processos pela quantidade de operações de leitura realizadas, de modo a refletir todos os custos inseridos por D1HF. Assim, como detalhado nesta figura e de maneira coerente com o discutido na Seção 7.6, as latências apresentadas incluem não só o tempo despendido nas operações de leitura propriamente ditas, como também o tempo que a aplicação ficou aguardando as instaurações das conexões com os *home nodes* dos diversos blocos, sendo que, de acordo com o mecanismo apresentado na Seção 7.5, esta última parcela inclui as latências das consultas feitas a D1HT para determinação do *home node* de cada bloco.



(a) Padrão SEQUENCIAL

(b) Padrão SALTEADO

Figura 9.7: Detalhamento dos tempos de leitura dos padrões SEQUENCIAL e SALTEADO (ou seja, detalhamento das parcelas *read* mostradas na Figura 9.6).

Da Figura 9.6 podemos ver que, de maneira coerente com os resultados já apresentados, para o padrão SEQUENCIAL, que mostrou escalabilidade sublinear, as latências medidas cresceram com o aumento do sistema, e para o padrão SALTEADO, que mostrou crescimento superlinear de desempenho, ocorreu efeito inverso. O detalhamento apresentado nesta figura nos permite observar que, para ambos os padrões e todos os tamanhos de sistema, a latência foi dominada pelo tempo das efetivas operações de leitura. Como era esperado, para o padrão SALTEADO o tempo de espera pela instauração de conexões foi mais significativo, uma vez que neste padrão o tamanho médio do salto (`lseek`) entre duas operações de leitura é dez vezes maior do que as próprias leituras (como pode ser visto da Tabela 9.1), de modo que, em média, para cada instauração de conexão menos de 10% do respectivo bloco é efetivamente lido.

Uma vez que a Figura 9.6 mostrou que o tempo das operações de leitura domina as latências para os dois padrões de acesso estudados, apresentamos na Figura 9.7 o detalhamento desta parcela da latência, quantificando o tempo de espera pelos dados lidos, ou seja, o tempo transcorrido entre a transmissão do pedido de leitura para o *home node* do bloco corrente e a chegada dos dados solicitados. Desta figura podemos ver que, para ambos os padrões de acesso e todos os tamanhos de sistema, a parcela de espera representa pelo menos 97% do tempo de leitura, o que é um indicativo preliminar de que estas latências podem ser minimizadas com uso de mecanismos de pré-carga que, como discutido na Seção 7.11, ainda não estão presentes na implementação de D1HF que foi aqui avaliada.

Por fim, iremos agora discutir a influência de D1HT no desempenho observado de D1HF em nossos experimentos. Como as latências das consultas feitas através de D1HT estão incluídas nos custos para instaurações das conexões aos *home nodes* dos blocos, da Figura 9.6(a) podemos facilmente concluir que D1HT foi capaz de resolver as consultas de maneira rápida o suficiente para não comprometer o desempenho de D1HF segundo o

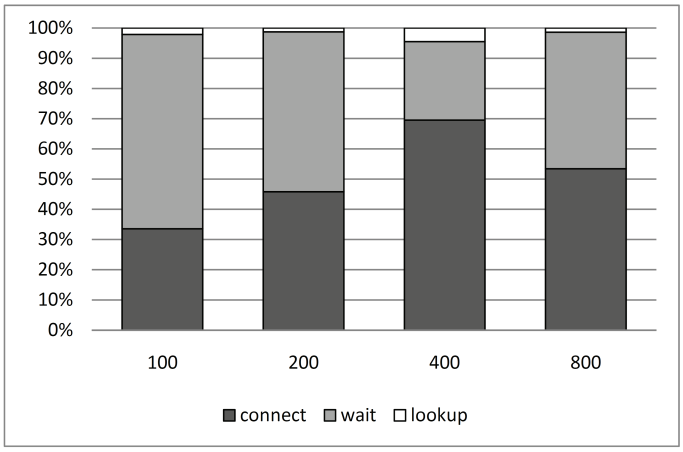


Figura 9.8: Detalhamento do tempo gasto para instauração das conexões com os *home nodes* dos blocos para o padrão SALTEADO (ou seja, detalhamento das parcelas *open* mostradas na Figura 9.6(b)). A latência das consultas direcionadas a D1HT está embutida na parcela *lookup* apresentada nesta figura.

padrão SEQUENCIAL, já que o tempo gasto com resolução de consultas está incorporado à parcela `open`, que a figura mostra ser desprezível para este padrão de acesso. O mesmo não podemos concluir para o padrão SALTEADO a partir da simples análise da Figura 9.6(b), e portanto apresentamos na Figura 9.8 o detalhamento do tempo despendido para instauração de conexões com os *home nodes* para este padrão, que nos permite concluir que, também no padrão SALTEADO, D1HT não comprometeu o desempenho de D1HF.

Mesmo que os resultados obtidos indiquem que, para os experimentos apresentados neste capítulo, o uso de uma MHDHT ou de um servidor de diretórios pudesse ser capaz de prover desempenho satisfatório, devemos ressaltar que estes experimentos foram realizados com até 800 processos, enquanto os padrões de acesso que endereçamos são utilizados em aplicações distribuídas que são também executadas em escalas significativamente maiores. Assim, enquanto o conjunto de resultados experimentais e analíticos de D1HT e D1HF que apresentamos ao longo desta tese indicam que D1HT seria capaz de suportar D1HF para sistemas muito maiores sem comprometer seu desempenho, uma MHDHT iria apresentar uma contínua degradação de latência, enquanto um servidor de diretórios iria, a partir de algum tamanho de sistema, enfrentar seu limite de escalabilidade. Desta forma, acreditamos que o uso de uma DHT puramente P2P, de baixa latência, auto-reorganizável e com reduzidos custos como D1HT, foi um fator importante para não só garantir que D1HF também tenha uma arquitetura puramente P2P e auto-reorganizável, como também para provê-lo de consultas que sejam rápidas o suficiente mesmo para sistemas vastos e dinâmicos.

Capítulo 10

Discussão

Neste capítulo iremos discutir nosso amplo conjunto de resultados obtidos ao longo do trabalho desta tese, como apresentados nos Capítulos 8 e 9.

Vamos primeiro debater a validação experimental das propriedades de D1HT, incluindo a sua análise quantitativa teórica, para então analisarmos os seus resultados comparativos com as outras SHDHTs estudadas. Em seguida, na Seção 10.3, iremos analisar o uso de D1HT como substrato do sistema de arquivos introduzido nesta tese.

A partir das discussões desenvolvidas nas três primeiras seções deste capítulo, iremos então, nas Seções 10.4 e 10.5, avaliar a adequação e utilidade de D1HT para os contextos atuais e futuros da Computação de Alto Desempenho e de aplicações de grande escala na Internet. Ao final deste capítulo iremos discutir o uso de D1HT como uma SHDHT de propósito geral.

10.1 Validação das Propriedades e da Análise Quantitativa de D1HT

Nesta tese nós introduzimos D1HT, uma inovadora SHDHT, e seu algoritmo para disseminação de eventos (EDRA). A partir de dois teoremas que introduzimos, enunciamos e demonstramos, fomos capazes de provar formalmente as suas propriedades de correção e balanceamento de carga, e desenvolver a análise quantitativa teórica dos custos de manutenção de tabelas de roteamento.

Mesmo considerando que D1HT é a única SHDHT que tem suas propriedades provadas formalmente, nós ainda realizamos vários experimentos de maneira a provar experimentalmente estas propriedades que já haviam sido demonstradas em teoria.

Os diversos experimentos conduzidos nos ambientes de computação de alto desempe-

nho e Planetlab, apresentados nas Seções 8.1.3 e 8.1.4, comprovaram que D1HT consegue consistentemente resolver mais de 99% das consultas com um único salto em diversas combinações de tamanho de sistema e ambiente de execução, mesmo na presença de altas taxas de adesões e partidas concorrentes de pares. Deste modo, estes experimentos foram capazes de atestar a correção do seu protocolo de adesão e, principalmente, de EDRA.

Os experimentos apresentados na Seção 8.1.4 demonstraram também que D1HT é leve, ao ponto de ter sido usado em conjunto com a produção normal de um CPD corporativo, sem gerar interferências. Nestes experimentos, os pares D1HT apresentaram consumo médio de CPU inferior a 0.02%, e menos de 40 KB de memória para tabelas de roteamento.

Os nossos resultados ainda comprovaram experimentalmente que as análises quantitativas teóricas de D1HT, que apresentamos no Capítulo 4, e de 1h-Calot, que havia sido introduzida em [99], conseguem estimar com precisão as suas demandas de banda passante de manutenção, mesmo tendo sido exercitadas em dois ambientes radicalmente distintos, com diferentes tamanhos de sistema e durações médias de sessão. Esta robusta validação experimental foi essencial para nos dar confiança no uso da análise quantitativa teórica destas duas SHDHTs em exercícios de custos de manutenção para grandes sistemas, com tamanhos que seriam inviáveis até mesmo para simulações.

Além disto, os experimentos apresentados na Seção 8.1.5 mostraram que D1HT consegue prover baixa latência de consultas mesmo com os pares sob altos níveis de carga computacional, e que as latências de suas consultas se mantêm constantes à medida que o sistema cresce.

Em suma, nosso conjunto de resultados experimentais conseguiu comprovar as seguintes propriedades de D1HT:

(i) D1HT é capaz de resolver mais de 99% das consultas com um único salto mesmo em ambientes com dinâmicas de pares similares às de aplicações P2P populares.

(ii) A análise quantitativa teórica desenvolvida para D1HT consegue prever suas demandas de rede para manutenção de tabelas de roteamento.

(iii) D1HT tem pequenas demandas de processamento e memória, podendo ser usado mesmo em servidores corporativos sob alta carga de processamento sem afetar a sua produção.

(iv) As latências das consultas em D1HT se mantém constantes à medida que o sistema cresce.

10.2 Comparação com Outras DHTs

Os experimentos conduzidos para avaliação comparativa de demandas de manutenção de tabela de roteamento nos ambientes de computação de alto desempenho e Planetlab, apresentados nas Seções 8.1.3 e 8.1.4, comprovaram que para sistemas de pequena e média escala D1HT tem custos iguais ou inferiores aos de 1h-Calot. Além disto, como discutido acima, estes experimentos validaram as suas análises quantitativas teóricas, o que nos permitiu que estas comparações experimentais fossem estendidas com estudos analíticos.

Deve-se notar que, em virtude da validação experimental das análises quantitativas de D1HT e 1h-Calot discutida na Seção 10.1, os resultados analíticos apresentados na Seção 8.2.2 são valiosos, uma vez que nos permitiram estudar as três únicas SHDHTs que suportam ambientes dinâmicos (D1HT, OneHop e 1h-Calot) segundo custos calculados a partir de análises quantitativas validadas experimentalmente[1], para tamanhos de sistema representativos de aplicações P2P paralelas, que seriam inviáveis até mesmo para simulações. Além disto, estes estudos foram conduzidos com tamanhos médios de sessão que são representativos de aplicações P2P populares. Os resultados obtidos comprovaram que D1HT apresenta custos de manutenção que são consistentemente menores do que os dos outros dois sistemas, com reduções típicas de uma ordem de magnitude, especialmente para os tamanhos de sistema e durações de sessão que são representativos das aplicações P2P populares.

A superioridade de D1HT em termos de demandas de manutenção foi devida à habilidade de EDRA para agrupar diversos eventos por mensagem de manutenção fazendo uso de uma arquitetura puramente P2P, enquanto em 1h-Calot cada par envia em média uma mensagem para cada evento, e OneHop recorre a uma topologia hierárquica com altos níveis de desbalanceamento de carga para conseguir fazer agregações de eventos.

Devemos ainda ressaltar que, como discutido na Seção 2.2, os resultados analíticos que apresentamos para 1h-Calot são também válidos para aos sistemas 1HS e SFDHT, de maneira que os custos de D1HT devem também ser uma ordem de magnitude menores do que os destas duas outras SHDHTs.

A nossa comparação de D1HT com outras DHTs foi então estendida com experimentos que compararam as latências alcançadas por D1HT com as providas por outras três soluções para resolução de consultas, sendo uma SHDHT, uma MHDHT e um servidor de diretórios. Os resultados destes experimentos, que foram apresentados na Seção 8.1.5, nos permitiram afirmar que, como já esperado, as MHDHTs incorrem em latên-

[1] A análise quantitativa teórica de OneHop já havia sido comprovada em um trabalho anterior [23].

cias muito superiores às conseguidas por SHDHTs, não sendo portanto adequadas para aplicações com desempenho crítico. Estes experimentos ainda comprovaram que, como também já era esperado, D1HT tem escalabilidade superior à solução Cliente/Servidor avaliada, sendo portanto mais apropriada para sistemas de larga escala. Adicionalmente, estes experimentos mostraram que mesmo para sistemas pequenos D1HT consegue prover desempenho próximo ao do servidor de diretórios, mesmo em ambientes com alta carga de processamento e frequentes adesões e partidas de nós.

Os nossos estudos analíticos também comprovaram que o mecanismo de Quarentena introduzido nesta tese é efetivo para a redução dos custos gerados por pares voláteis em sistemas com comportamento representativo de aplicações P2P reais e populares.

Em resumo, nosso conjunto de resultados experimentais e analíticos comparando D1HT com outras DHTs e um servidor de diretórios, mostraram que:

(i) D1HT tem custos de manutenção consistentemente inferiores aos das demais SHDHTs (i.e., OneHop, 1h-Calot, 1HS e SFDHT), com reduções de uma ordem de magnitude para sistemas grandes, em virtude da sua singular habilidade de agrupar eventos com uma arquitetura puramente P2P, o que lhe permite associar baixos custos de manutenção com bom balanceamento de carga;

(ii) D1HT é capaz de prover latências menores que as MHDHTs;

(iii) D1HT é mais escalável do que diretórios com arquitetura Cliente/Servidor;

(iv) Para sistemas pequenos, D1HT provê latências de consulta similares às conseguidas por diretórios com arquitetura Cliente/Servidor, mesmo que o ambiente esteja sob alta carga de processamento e sujeito a adesões e partidas.

(v) Para sistemas grandes, D1HT provê latências de consulta menores do que diretórios com arquitetura Cliente/Servidor, em virtude de sua superior escalabilidade.

10.3 Validação de D1HT como Substrato de Sistemas Distribuídos

De modo a validar a habilidade e adequação de D1HT como uma ferramenta útil para construção de sistemas distribuídos, nesta tese nós introduzimos, projetamos, desenvolvemos e avaliamos o sistema de arquivos D1HF, que foi apresentado no Capítulo 7.

Para embasar o projeto de D1HF, nós estudamos o padrão de acesso de algumas aplicações paralelas chaves de ambientes HPC, bem como a arquitetura dos multicomputadores

que são o padrão de fato em termos de plataforma de computação para estes ambientes. Mais do que isto, nós analisamos as oportunidades de sinergia entre estas aplicações e plataformas de processamento estudadas, procurando resolver aspectos críticos das aplicações com uso de componentes ociosos dos multicomputadores. Além disto, consideramos também aspectos que são fundamentais em ambientes corporativos de produção, procurando facilitar a integração da nossa proposta com padrões já existentes e consolidados, bem como sua interoperabilidade com outros sistemas e ambientes.

Como resultado de todo este estudo e análise, fomos capazes de propor um sistema de arquivos que é totalmente aderente ao padrão POSIX e pode ser usado de maneira transparente por aplicações e ambientes já existentes, sendo capaz de se integrar facilmente a rotinas e padrões já implantados em uma corporação. O mais importante, este sistema pode acelerar padrões de acesso que são críticos em aplicações HPC chaves, sem a necessidade de qualquer contratação ou dispêndio tanto em componentes de *hardware* quanto de *softwares*.

Para atingir estes diversos objetivos, D1HF se propõe a, de maneira transparente para aplicação e demais sistemas e padrões em uso no ambiente de produção, explorar a localidade de acessos observada em algumas aplicações paralelas, de maneira a manter os dados mais populares em uma estrutura de cache construída com uso de discos, memória principal, interconexões e comutadores que estejam ociosos.

Para que consigamos realmente usufruir da ociosidade destes diversos componentes, o uso de uma arquitetura puramente P2P é conveniente, caso contrário estaríamos impondo a necessidade de servidores dedicados, que têm custos, impõe limites de escalabilidade e necessitam de dimensionamento e gerenciamento especial. Além disto, estes servidores dificultariam a implantação da nossa solução, poderiam criar exceções aos padrões do ambiente operacional, e se tornariam pontos críticos de falha. Idealmente, o sistema proposto deveria ainda ser auto-reorganizável, de maneira a se integrar mais facilmente às rotinas já estabelecidas no ambiente e minimizar esforços de gerenciamento.

Desta maneira, uma vez que as cópias caches dos dados ficam distribuídas por diversos componentes de multicomputadores com até milhares de nós, o sucesso da estratégia proposta por D1HF depende da disponibilidade de uma ferramenta puramente P2P e auto-reorganizável que permita a rápida localização de dados em grandes sistemas distribuídos. Além disto, esta ferramenta deve proporcionar a distribuição destas cópias cache com bom balanceamento de carga.

Em virtude das diversas características e requisitos acima discutidos, D1HT se tornou uma peça chave para o sucesso do sistema de arquivos D1HF, uma vez que ele provê bom balanceamento de carga e baixas latências aliados com boa escalabilidade, permitindo

assim a rápida localização de informações largamente distribuídas com uso de uma arquitetura puramente P2P e auto-reorganizável. Desta maneira, D1HF não só representa um caso real de sistema que demanda ferramentas com o perfil de D1HT, como também nos proporciona uma oportunidade para sua avaliação como componente de soluções úteis e facilmente integráveis em ambientes HPC.

Deste modo, nós implantamos D1HF como parte do trabalho desta tese, de maneira a nos permitir não só a avaliação das estratégias propostas para este novo sistema de arquivos, como também para habilitar a confirmação do desempenho, utilidade e adequação de D1HT como um substrato útil para aplicações distribuídas que requerem uma conjunção de desempenho e escalabilidade.

Assim, conduzimos vários experimentos com uma implementação inicial de D1HF com até 800 nós de um multicomputador usado em um ambiente de computação de alto desempenho, segundo padrões de acesso observados em aplicações paralelas reais. Estes resultados experimentais, que foram apresentados no Capítulo 9, mostraram que D1HF é capaz de prover banda passante de leitura até oito vezes superior à proporcionada por um sistema de arquivos que representa o estado da arte em termos de servidores NFS comerciais para ambientes corporativos de produção. Nossos experimentos mostraram ainda que D1HF é capaz de reduzir em até 99% a carga de leitura no sistema utilizado para armazenamento efetivo dos dados para os perfis de acesso estudados. Além disto, nós instrumentamos o código de D1HF de maneira a permitir o detalhamento das latências dos seus acessos, que mostrou que D1HT conseguiu resolver as consultas de maneira rápida para todos os tamanhos de sistema avaliados, e não deve comprometer o desempenho de D1HF mesmo para sistemas consideravelmente maiores.

Acreditamos então que nossos resultados experimentais obtidos com D1HF são relevantes, não só por que mostraram que este sistema de arquivos é capaz de prover altos níveis de desempenho de leitura para uma classe importante de aplicações HPC, mas principalmente porque confirmaram que D1HT pode ser usado como uma ferramenta útil para construção de sistemas P2P escaláveis e auto-reorganizáveis para ambientes e aplicações sensíveis à latência.

10.4 Uso de D1HT em Ambientes HPC

Em relação ao potencial uso de D1HT em ambientes corporativos de desempenho crítico, consideramos que a discussão acima e os resultados que apresentamos nos Capítulos 8 e 9 nos permitem construir conclusões animadoras.

Primeiro, por que, como já discutido na Seção 10.1, nossos experimentos mostraram

ser possível usar D1HT em CPDs corporativos conjuntamente com a sua produção normal sem causar interferências, mesmo que seus servidores estejam sob alta carga de uso de processamento. Além disto, D1HT apresentou utilizações de CPU e memória desprezíveis.

O mais importante, no entanto, foi o desempenho observado para D1HT em nossos experimentos de medições de latência em ambiente corporativo, apresentados na Seção 8.1.5 e discutidos na Seção 10.3. Os resultados obtidos nestes experimentos indicam claramente que, mesmo com os nós sob alta carga de processamento, D1HT é capaz de prover latências absolutamente próximas às que são alcançadas por um servidor de diretórios em sistemas pequenos, e, como consequência de sua superior escalabilidade, obter resultados melhores para sistemas médios ou grandes.

Por outro lado, a MHDHT avaliada não conseguiu prover latências comparáveis com as atingidas por D1HT, além de ter tido desempenho inferior ao servidor de diretórios para tamanhos pequenos de sistema, comprovando que MHDHTs não são adequadas para aplicações com desempenho crítico.

Complementado estes resultados, os experimentos que apresentamos no Capítulo 9 com um sistema de arquivos desenvolvido nesta tese, mostraram efetivamente que D1HT pode ser usado como um substrato útil para construção de sistemas que são capazes de melhorar o desempenho de aplicações paralelas em ambientes HPC de produção.

Desta maneira, acreditamos que o conjunto de resultados obtidos com D1HT e D1HF mostram que, já hoje, D1HT pode ser usado como uma ferramenta útil para centros corporativos de computação de alto desempenho.

Além disto, acreditamos que as tendências futuras devem tornar D1HT cada vez mais atraente e útil para ambientes HPC. Para embasar esta hipótese, chamamos a atenção para o contínuo e acelerado crescimento dos ambientes HPC, como é nitidamente ilustrado pela Figura 10.1, que apresenta a evolução ao longo dos últimos 17 anos da dimensão dos maiores computadores do mundo segundo a lista TOP500 [100]. Podemos, por exemplo, observar que no início deste século a maioria dos sistemas classificados entre os 500 maiores do mundo tinha entre 65 e 256 processadores. Já em 2005, a maioria dos computadores desta lista tinha entre 513 e 2048 processadores. Na sua última edição, esta maioria passou a ser composta por sistemas com dimensões entre 4 mil e 8 mil processadores.

Desta maneira, a Figura 10.1 nos mostra não só que o crescimento dos ambientes HPC ocorre sob um ritmo acentuado, como também que este processo de expansão contínua já dura quase duas décadas. Para agravar esta situação, devemos ainda considerar que conjuntamente com o crescimento dos sistemas, deveremos também observar ganhos acelera-

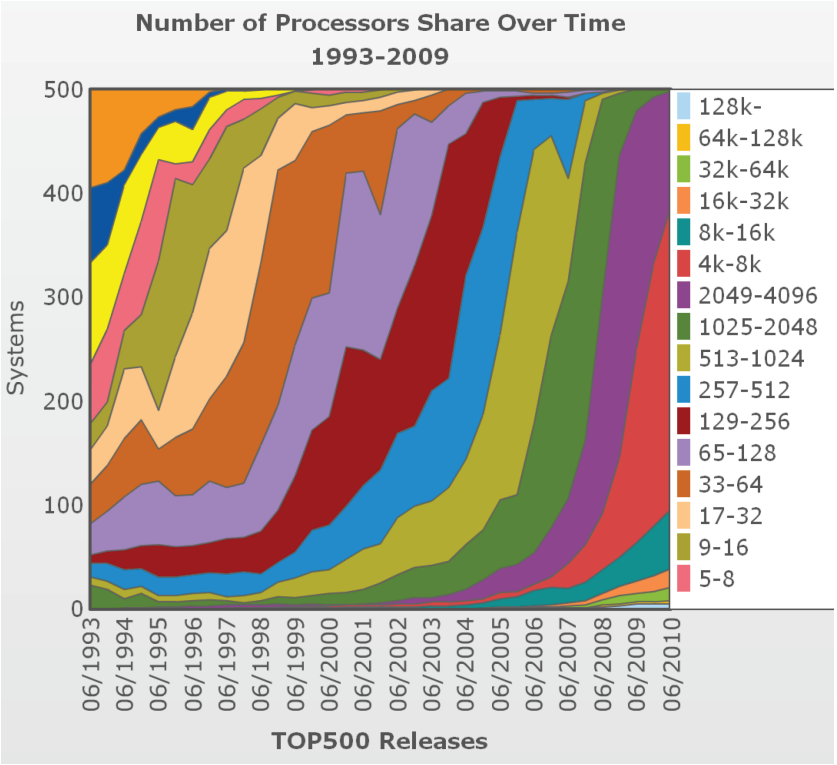


Figura 10.1: Distribuição das dimensões dos computadores da lista TOP500 [100] ao longo dos anos. Ilustração disponível a partir de http://www.top500.org/.

dos em termos de disponibilidade de banda passante, enquanto as melhorias das latências de rede devem ocorrer em ritmo muito mais lento [64]. Frente a estas tendências, podemos afirmar que as limitações de escalabilidade e os custos das soluções Cliente/Servidor tendem a se tornar cada vez mais críticas, e as falhas de componentes devem ser cada vez mais corriqueiras, de modo que a sua habilidade de aliar baixas latências e alta escalabilidade com uso de uma arquitetura puramente P2P e auto-reorganizável tendem a, com o passar dos anos, tornar D1HT uma ferramenta cada vez mais atrativa e útil para centros de computação de alto desempenho.

10.5 Uso de D1HT na Internet

Acreditamos que também para aplicações de grande escala na Internet, as conjunturas atuais e futuras mostram que D1HT pode hoje ser útil, e que sua atratividade tende a crescer com o tempo, como discutiremos a seguir.

Nossos resultados analíticos, que foram apresentados na Seção 8.2.2, mostraram que os custos de D1HT para sistemas com um milhão de pares, e comportamento representativo de Gnutella e KAD, seriam de aproximadamente 7 kbps por par, que são reduzidos

para menos de 2 kbps para sistemas com este mesmo tamanho e comportamento similar ao de BitTorrent.

Uma vez que ainda em 2004, a velocidade média de conexão dos pares BitTorrent era próxima a 240 kbps [68], acreditamos que os custos inferiores a 2 kbps requeridos por D1HT já seriam hoje plenamente aceitáveis, ou mesmo desprezíveis, para sistemas com um milhão de pares e comportamento similar ao desta popular aplicação. Mais do que isso, questionamos se neste caso seria razoável empregar soluções que, como as MHDHTs, buscariam minimizar esta já diminuta demanda de rede em prejuízo da latência das consultas. Deve-se notar que D1HT poderia permitir o desenvolvimento de um novo meio para busca de arquivos compartilhados em comunidades BitTorrent, que seria mais rápido e eficaz do que o atual mecanismo baseado em sítios na Internet.

Além disto, estudos mais recentes mostraram que a maioria das conexões domésticas tem capacidade de pelo menos 512 kbps, e ocupação muito baixa [22, 50]. Desta maneira, acreditamos que mesmo os custos em torno de 7 kbps de D1HT para sistemas com um milhão de pares e comportamento representativo de Gnutella e KAD possam já ser aceitáveis para a maioria dos pares domésticos, especialmente se considerarmos que os resultados apresentados na Seção 8.2.3 indicam que estas demandas poderiam ser reduzidas em cerca de 20% com o uso de Quarentena. Deste modo, D1HT poderia viabilizar uma significante melhoria no desempenho e eficácia dos atuais mecanismos de buscas de Gnutella e KAD, que são, respectivamente, baseados em inundação e MHDHTs.

Devemos ainda considerar que, conforme os resultados apresentados na Seção 8.2.2 e a discussão conduzida na Seção 10.2, as demais SHDHTs já propostas (i.e., OneHop, 1h-Calot, 1HS e SFDHT) iriam acarretar custos de manutenção uma ordem de magnitude maiores do que os D1HT. Mais especificamente, para sistemas com comportamento de KAD ou Gnutella e um milhão de pares, OneHop, 1h-Calot, 1HS e SFDHT iriam requerer mais de 140 kbps de banda passante de manutenção, o que torna o seu uso proibitivo, especialmente se considerarmos que estes sistemas não iriam trazer melhorias para as latências das consultas em relação a D1HT.

Deste modo, acreditamos que hoje D1HT já se mostra como uma ferramenta útil e viável para importantes aplicações de grande escala na Internet, mas as MHDHTs podem ainda ser mais apropriadas para sistemas altamente dinâmicos ou extremamente grandes, ou mesmo para ambientes com fortes limitações de banda passante. Por exemplo, as demandas de manutenção de D1HT próximas a 65 kbps para sistemas com dez milhões de pares e dinâmicas similares a KAD e Gnutella podem ainda ser demasiadas para pares domésticos atuais.

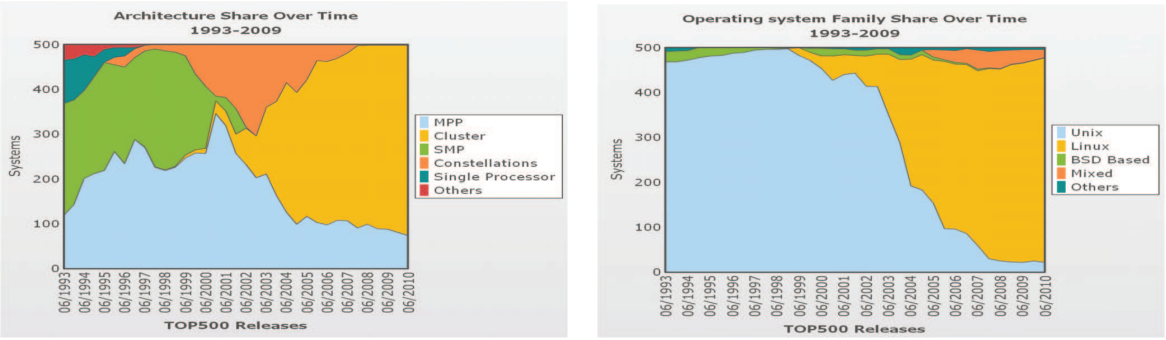
Uma vez que a discussão acima foi conduzida segundo o contexto atual, gostaríamos

de complementá-la de modo a refletir aspectos futuros. Para tanto, devemos considerar que, como já discutido, é esperado que a disponibilidade de banda passante e as capacidades de processamento continuem crescendo continuamente, enquanto os ganhos em termos de redução de latência deverão ocorrer em um ritmo significativamente mais lento [64]. Este horizonte traz, ao menos, duas importantes implicações para nossa discussão. Primeiro, por que os acelerados ganhos em termos de banda passante tendem a tornar os custos de D1HT cada vez mais aceitáveis, de maneira à continuamente habilitar o seu uso em espectros cada vez mais amplos de aplicações e ambientes. Segundo, por que as expectativas de desempenho dos usuários e aplicações deverão crescer de maneira consonante com os ganhos de disponibilidade de banda passante e de capacidade de processamento, enquanto as melhorias relativamente marginais em termos de latência poderão frustrar estas expectativas. Desta maneira, acreditamos que cada vez mais deveremos priorizar soluções que buscam a minimização da latência, de modo que as MHDHTs, que hoje são usadas para diversas aplicações distribuídas, poderão continuamente perder espaço para SHDHTs, especialmente D1HT, que consegue prover baixa latência aliada com bom balanceamento de carga e custos uma ordem de magnitude menores que as demais alternativas.

10.6 D1HT: Uma SHDHT de Propósito Geral

Acreditamos que a habilidade que D1HT tem para atender aos requisitos de ambientes tão díspares quanto os representados por CPDs HPC e aplicações P2P populares seja uma de suas propriedades mais importantes, já que o histórico recente dos ambientes HPC mostra o domínio de soluções construídas por componentes de *hardware* e *software* populares, atestando assim a importância da economia de escala e da versatilidade das soluções.

Como já discutido na Seção 7.2 e ilustrado na Figura 10.2(a), os multicomputadores construídos com componentes *commodities*, que incluem também soluções baseadas em processamento heterogêneo com GPGPUs ou processadores de consoles de *videogames*, estão dominando a computação de alto desempenho há mais de uma década. Isto mostra que mesmo as grandes somas que são continuamente investidas nesta área por poderosas organizações (como centros de pesquisa, organismos governamentais e companhias de petróleo) em geral não são suficientes para justificar o desenvolvimento de componentes para uso específico em HPC. Este fenômeno é decorrente da forte economia de escala na fabricação e do contínuo e acelerado desenvolvimento destes componentes populares, e se manifesta também em termos de *softwares*. Na verdade, devemos observar que a adoção de *softwares* populares em ambientes HPC, que inclui, mas não se limita ao, sistema



(a) Arquitetura dos Computadores

(b) Sistemas Operacionais

Figura 10.2: Histórico das arquiteturas e dos sistemas operacionais dos 500 maiores computadores para cálculos numéricos do mundo segundo a lista TOP500 [100]. Gráficos disponíveis a partir de http://www.top500.org/.

operacional Linux (como mostrado na Figura 10.2(b)), pode estar ainda mais fortemente difundida do que o já altíssimo domínio alcançado pelos componentes *commodities* de *hardware*. Para exemplificar esta realidade, a lista TOP500 mais recente [100] mostra que 424 de seus integrantes são multicomputadores *commodities* (ou seja, 85%), enquanto o sistema operacional Linux é usado por 455 dos computadores desta lista (ou seja, 91%).

Estes fatos corroboram a nossa argumentação em favor do desenvolvimento e implantação de ferramentas que possam ter amplo emprego em uma diversidade de ambientes, uma vez que quanto mais popular seja o seu uso maior deverá ser a sua probabilidade de sucesso. Assim, creditamos como um diferencial importantíssimo a singular habilidade que D1HT tem para aliar ótimas latências com baixos custos, e ainda ser versátil o suficiente a ponto de ser potencialmente usado de maneira eficiente em uma ampla diversidade de ambientes.

Mais especificamente, enquanto os resultados que apresentamos na Seção 8.1.4 mostram que 1h-Calot incorre em custos de manutenção que são plenamente aceitáveis para ambientes corporativos, os resultados apresentados na Seção 8.2.2, e discutidos na Seção 10.2, indicam que esta (e outras) SHDHTs não seriam viáveis para aplicações de larga escala da Internet.

Deste modo, assim como ocorre com diversos outros componentes de *hardware* e *software*, acreditamos que mesmo para ambientes HPC torna-se muito mais atrativo o uso de uma solução versátil como D1HT, do que investir no desenvolvimento e manutenção de uma SHDHT dedicada para este ambiente, especialmente se considerarmos que D1HT já se encontra disponível para uso gratuito. Além disto, nos já comprovamos que D1HT é capaz de resolver as consultas com um único salto, e portanto apresenta o mesmo nível de desempenho que qualquer outra SHDHT que venha a ser desenvolvida poderia atingir.

Capítulo 11

Conclusões e Trabalhos Futuros

Neste capítulo são apresentadas as principais conclusões decorrentes deste trabalho de tese, que foram embasadas por um amplo conjunto de resultados experimentais e analíticos, e pelas discussões que conduzimos no Capítulo 10 frente aos contextos atuais e futuros da Computação de Alto Desempenho e de aplicações de grande escala na Internet. Iremos também listar as oportunidades de trabalhos futuros que foram identificadas ao longo desta tese, e que julgamos como necessárias ou relevantes.

11.1 Conclusões

Enquanto, ao longo do tempo, as questões de latência tendem a se tornar mais críticas do que a disponibilidade de banda passante, as primeiras DHTs propostas optaram por sacrificar o desempenho de suas consultas de maneira a minimizar o consumo de banda de rede, tornando-as impróprias para aplicações com desempenho crítico. Por outro lado, a menos do sistema proposto nesta tese, todas as recentes DHTs de baixa latência têm altíssimos custos de manutenção, que as tornam inviáveis para aplicações de grande escala populares na Internet.

Frente a este contexto, nesta tese introduzimos o inovador sistema D1HT, que tem uma arquitetura puramente P2P e auto-reorganizável, e é a primeira DHT que procura combinar baixas latências de consultas com reduzidos custos de manutenção e bom balanceamento de carga, mesmo para sistemas com dinâmicas caracterizadas por taxas de adesões e partidas de pares representativas de aplicações de grande escala na Internet. De maneira a validar estas características, nós introduzimos, enunciamos e demonstramos dois teoremas que atestaram as propriedades de correção, baixa latência e balanceamento de carga de D1HT, e nos permitiram ainda o desenvolvimento na análise teórica quantitativa de seus custos de manutenção.

Como parte do trabalho desta tese, nós então implementamos D1HT e a única outra SHDHT puramente P2P que suporta ambientes dinâmicos, de maneira a não só validar experimentalmente as características do sistema que introduzimos, como também para permitir comparações experimentais relevantes.

Assim, mesmo considerando que D1HT é a única DHT de baixa latência a ter suas propriedades demonstradas formalmente, nós conduzimos o maior conjunto de avaliações experimentais já publicadas de DHTs, que nos permitiram provar também experimentalmente todas estas suas importantes propriedades. Estes experimentos foram realizados com até 4.000 pares e 2.000 nós físicos, e validaram as características de D1HT para diferentes tamanhos de sistema e dinâmicas de adesões e partidas pares, tendo sido conduzidos em dois ambientes radicalmente distintos (um Centro de Computação de Alto Desempenho e uma rede mundialmente dispersa na Internet). Nossos experimentos comprovaram ainda que D1HT tem reduzidas demandas de processamento e memória principal que habilitam seu uso mesmo em servidores de ambientes de produção corporativa que estejam sob alta carga de processamento.

Desta maneira, fomos capazes de concluir, tanto teoricamente a partir dos teoremas que introduzimos, quanto experimentalmente em virtude do nosso conjunto de experimentos, que a DHT introduzida nesta tese é capaz de (i) resolver mais de 99% das consultas com um único salto mesmo em ambientes dinâmicos; (ii) demandar baixos custos de manutenção das tabelas de roteamento; (iii) assegurar bom balanceamento de carga de manutenção; e (iv) se adaptar a mudanças na dinâmica do ambiente. Todas estas propriedades foram alcançadas com uso de uma arquitetura puramente P2P e auto-reorganizável.

Nossos experimentos foram então estendidos de maneira a proporcionar a primeira comparação experimental de três diferentes DHTs e um servidor de diretórios. Os resultados obtidos demonstraram que D1HT consistentemente provê latências iguais ou melhores que todos os outros sistemas avaliados, e tem escalabilidade muito à conseguida pela solução Cliente/Servidor.

Uma vez que nossos experimentos comprovaram a análise quantitativa teórica de D1HT, nós estendemos nossos resultados experimentais com estudos analíticos comparando os custos de D1HT com os das únicas duas outras DHTs de baixa latência que se propõe a suportar ambientes dinâmicos. Estes estudos analíticos têm especial valor por que foram baseados em análises quantitativas validadas experimentalmente, e utilizaram parâmetros oriundos de implementações de sistemas DHTs e medições do comportamento de pares de aplicações populares reais.

Os nossos resultados analíticos nos permitiram concluir que D1HT tem custos menores que os de todos os outros sistemas avaliados, com reduções superiores a uma ordem de

magnitude para tamanhos de sistema e comportamento de pares que são representativos de aplicações P2P de larga escala populares na Internet, como Gnutella, KAD e BitTorrent. Mais importante, estes estudos também mostraram que para estas aplicações P2P populares, D1HT requer demandas de manutenção que são aceitáveis para a maioria das conexões domésticas atuais, mesmo em sistemas de larga escala com até um milhão de pares, enquanto as demais DHTs estudadas apresentam custos que as tornam inviáveis para aplicações com estes comportamentos e dimensões.

Além disto, introduzimos também o mecanismo de Quarentena, que reduz os impactos negativos causados por pares voláteis, e pode ser usado para outros objetivos, como prevenção contra ataques maliciosos e multidões repentinas. Nossos resultados analíticos mostraram que, apesar de simples, o mecanismo de Quarentena pode reduzir em até de 31% os custos de manutenção de D1HT em ambientes com dinâmica similar às de aplicações P2P populares, como Gnutella.

Uma vez tendo comprovado as propriedades e menores custos de D1HT frente às demais SHDHTs para diferentes ambientes, tamanhos de sistema e comportamento de pares, julgamos então tomar como próximo passo a validação efetiva de D1HT como uma ferramenta útil para construção de sistemas distribuídos de grande escala com desempenho crítico. Para tanto, como parte do trabalho desta tese, nós estudamos diversas características de algumas aplicações paralelas chaves em centros de computação de alto desempenho corporativos, bem como a plataforma de computação comumente usada nestes ambientes, de maneira a embasar o projeto de um sistema distribuído útil e relevante.

Em consequência destes estudos, nós introduzimos, projetamos e desenvolvemos um sistema de arquivos puramente P2P e auto-reorganizável, que, com uso de D1HT como uma ferramenta fundamental, procura maximizar o desempenho de padrões de acesso a arquivos que representam o componente crítico de entrada e saída de dados de algumas aplicações distribuídas chaves. Para tanto, este sistema de arquivos faz uso unicamente de discos, memória principal, conexões e comutadores normalmente ociosos dos multicomputadores que representam o padrão de fato da computação de alto desempenho, dispensando assim contratações de componentes adicionais.

Este sistema de arquivos, que foi batizado de D1HF, foi então submetido a uma avaliação experimental, que comparou o seu desempenho contra o de um sistema de arquivos comercial de última geração. Nossos resultados experimentais mostraram que, para os padrões de acesso estudados, D1HF pode atingir desempenho até oito vezes superior ao do sistema comercial avaliado, e que D1HT foi uma peça importante para garantir a sua escalabilidade e viabilizar a sua implementação segundo uma arquitetura puramente P2P e auto-reorganizável. Nossos estudos mostraram ainda que o uso de D1HF pode ser contra-

producente para padrões de acesso e tamanhos de sistema que não são adequados às suas estratégias, o que não é preocupante, uma vez que seu uso poder ser ativado de maneira seletiva para os arquivos apropriados, de maneira simples e transparente.

Como resultado do projeto, desenvolvimento e, sobretudo, da avaliação experimental comparativa de D1HF, podemos concluir que D1HT pode efetivamente ser usado como uma ferramenta útil para aplicações distribuídas de desempenho crítico.

Desta maneira, em virtude das propriedades que nós enunciamos e provamos formalmente, das implementações reais de sistemas DHT e de arquivos que realizamos, e do nosso conjunto de resultados experimentais e analíticos, julgamos que podemos concluir que D1HT já é hoje uma ferramenta adequada, atraente e útil para o ***contexto presente*** dos Centros de Computação de Alto Desempenho e das aplicações de grande escala na Internet, além de apresentar características que superam todas as demais DHTs de baixa latência já introduzidas.

Além disto, uma vez que já foi demonstrado que *com o passar do tempo, o aumento da largura de banda passante é tipicamente maior do que o quadrado da redução das latências* [64], e que ao longo das duas últimas décadas as dimensões dos sistemas de computação de alto desempenho têm crescido aceleradamente, concluímos ainda que existem sólidas tendências que devem no ***contexto futuro*** tornar o uso, adequação e utilidade de D1HT cada vez mais atrativos tanto para Centros de Computação de Alto Desempenho como para aplicações de grande escala na Internet.

Destacamos ainda que a versatilidade que habilita o emprego de D1HT em ambientes radicalmente distintos, tanto em seus contextos presentes quanto futuros, é uma de suas características mais importantes, podendo alavancar o seu uso como um substrato útil e escalável para espectros cada vez mais amplos de aplicações e ambientes distribuídos. Como um passo em direção a este objetivo, nós disponibilizamos o código fonte de D1HT para uso livre e gratuito [54], de maneira a facilitar a sua utilização bem como o desenvolvimento de novas pesquisas nesta área.

Acreditamos então ter concluído o ciclo ideal desta tese, uma vez que, após a introdução de um sistema DHT inovador, nós comprovamos formalmente, analiticamente e experimentalmente as suas propriedades de correção, balanceamento de carga, custos e desempenho, bem como a sua superioridade frente às demais SHDHTs. Além disto, nós implementamos e avaliamos experimentalmente um sistema distribuído real fundamentado na DHT que introduzimos. Nossos diversos resultados nos permitiram concluir que a nossa principal contribuição já é hoje adequada e atraente para diferentes ambientes computacionais reais, e que tende a ser cada vez mais útil e importante. Fechando este ciclo, nós disponibilizamos o seu código fonte para uso livre e gratuito.

11.2 Trabalhos Futuros

Nesta seção iremos apresentar de maneira concisa diversas oportunidades de trabalhos futuros que foram identificadas e discutidas ao longo do trabalho e da apresentação desta tese, e que consideramos relevantes ou obrigatórias. Listaremos a seguir estas oportunidades, agrupadas por assunto, sempre indicando as seções e/ou capítulos onde foram identificadas e discutidas.

(i) Quarentena

   (a) Desenvolvimento e avaliação do mecanismo de Quarentena (Capítulos 5 e 6).

(ii) Introdução e Desenvolvimento de Novas Tabelas Hash Distribuídas

   (a) Introdução, desenvolvimento e avaliação experimental de D2HT, uma nova DHT puramente P2P construída a partir de D1HT. Cada par em D2HT mantém uma tabela de roteamento com $O(\sqrt{n})$ entradas, e resolve as consultas com dois saltos (Seção 2.3).

   (b) Extensão de D2HT de modo a incluir informações de localidade, de maneira que os diversos pares possam direcionar (ao menos) o primeiro salto de cada consulta para um par próximo em termos de latência (Seção 2.3).

(iii) Desenvolvimento da Implementação de D1HT

   (a) Implantação da estratégia que permite a incorporação de D1HT à aplicação, como alternativa ao seu uso como um processo estanque (Seção 6.2).

   (b) Implementação de opção para uso de localidade em D1HT (Capítulo 3).

(iv) Desenvolvimento, Avaliação e Disponibilização da Implementação de D1HF

   (a) Implementação de mecanismo para pré-carga de dados para padrões de acesso regulares (Seções 7.1, 7.11 e 9.4).

   (b) Criação de múltiplas cópias cache para blocos muito acessados, e seu uso de maneira a usufruir de sua localidade (Seções 7.5 e 7.11).

   (c) Implementação de mecanismo que minimiza os custos de acessos a arquivos que não sofram releituras (Seções 7.5, 7.11 e 9.2).

   (d) Avaliação experimental de D1HF em modo montado (Seção 9.2).

(e) Implementação e avaliação experimental de estratégias alternativas para definição do nó que armazenará a primeira cópia de cada bloco (Seção 7.5).

(f) Avaliação experimental de D1HF para aplicações distribuídas usadas em Intranets corporativas (Seção 7.3).

(g) Disponibilização do código fonte de D1HF para uso livre e gratuito (Seção 7.11).

Glossário

API : Acrônimo do termo em inglês *Application Program Interface*. Uma API representa um conjunto de rotinas que disponibilizam as funcionalidades de um *software* para programas aplicativos.

Beowulf : Multicomputador formado por um aglomerado de computadores idênticos, construído com componentes que são padrão de mercado (i.e., *commodities*), e que é utilizado para processamento científico paralelo [94]. Todos os multicomputadores usados em nossos experimentos, e que foram apresentados na Tabela 8.1, seguem esta arquitetura, que foi discutida na Seção 7.2.

Conhecer um Evento : Dizemos que um par *conhece* (ou *toma conhecimento de*) um evento em um sistema D1HT, quando ele detecta a saída ou adesão de seu predecessor, ou quando ele recebe uma mensagem notificando o evento.

CPD : Acrônimo do termo Centro de Processamento de Dados.

$\mathbb{D}$ **:** Conjunto de participantes de um sistema D1HT, como definido no Capítulo 3.

DHT : Acrônimo do termo em inglês *Distributed Hash Table*, ou seja, Tabela *Hash* Distribuída.

Duração de Sessão : Mesmo que *tamanho de sessão*.

EDRA : Acrônimo do termo em inglês *Event Detection and Propagation Algorithm*. Algoritmo definido no Capítulo 4, usado por D1HT para propagação dos eventos em D1HT.

Evento : Adesão ou partida de um participante do sistema. Ver também a definição de *Conhecer um Evento*.

ID : Identificador inteiro (no intervalo $[0 : N]$) de uma chave ou par, como definido no Capítulo 3. De acordo com as técnicas de *consistent hashing* [37], o ID de uma

chave é obtido aplicando-se uma função criptográfica ID() ao valor da chave, e o ID de um par é obtido aplicando-se a mesma função ID() ao seu endereço IP.

ID() : Função que gera o ID de uma chave ou par. Tipicamente, ID() é a função criptográfica SHA-1 [60].

Intervalo Θ : Intervalo de tempo com duração Θ, como definido na Seção 4.1, durante o qual EDRA acumula as notificações de eventos recebidas, para agrupá-las em até ρ mensagens, e assim minimizar as demandas de banda passante para manutenção das tabelas de roteamento.

ISP : Acrônimo do termo em inglês *Internet Service Providers*, ou seja, empresa provedora de serviços de Internet.

HPC : Acrônimo do termo em inglês *High Performance Computing*, ou seja, Computação de Alto Desempenho.

Latência de Consulta : No contexto desta tese, definimos *latência de consulta* de uma DHT como sendo a duração do intervalo de tempo para que o nó origem obtenha a resposta de uma consulta, ou seja, a duração do intervalo do tempo transcorrido entre o momento em que a consulta é iniciada em seu *nó origem*, até o momento em que este mesmo *nó origem* recebe a sua resposta.

MHDHT : Acrônimo do termo em inglês *Multi-Hop DHT*, ou seja, DHT que necessita de vários saltos para resolver cada consulta.

n : Quantidade de pares em um sistema D1HT, ou seja, $n = |\mathbb{D}|$.

N : Limite superior dos IDs em D1HT, sendo que obrigatoriamente $N \gg n$. Uma vez que, tipicamente, a função criptográfica usada para determinação dos IDs é a SHA-1 [60], em geral $N = 2^{160} - 1$.

Nível de Bloqueio : Mesmo que Nível de Contenção.

Nível de Contenção : O nível de contenção de um multicomputador (ou de sua árvore gorda) é definido na Seção 7.2 como sendo a razão entre a soma das capacidades das conexões dos diversos nós computacionais aos comutadores TOR, e a capacidade das conexões destes comutadores ao nível superior da árvore gorda. Em geral, os multicomputadores são construídos de maneira simétrica, de maneira que o resultado desta razão é o mesmo para todos os seus comutadores TOR. Caso contrário, deverá ser considerada para o multicomputador a pior (i.e., a maior) razão dentre

todos os seus comutadores. Apesar deste termo ter sido usado nesta tese com foco em redes Ethernet, esta mesma definição é válida para outras tecnologias, incluindo Infiniband e FC.

Nível de Releitura: Nível de releitura a arquivos de entrada de uma aplicação, como definido na Equação 7.1, Seção 7.1.

P2P : Acrônimo do termo em inglês *peer-to-peer*, ou seja, par-a-par.

Nó Destino : Nó destino de uma consulta é o nó da DHT que tem a resposta para a respectiva consulta.

Nó Origem : Nó origem de uma consulta é o nó da DHT que iniciou (ou emitiu) a respectiva consulta, sendo também o nó ao qual a sua resposta deverá ser retornada. Assim, a DHT deverá ser responsável por encaminhar cada consulta a partir de seu *nó origem* até seu *nó destino*, e por então retornar a sua resposta ao *nó origem*.

$pred(p,i)$ **:** Função que define o i_{esimo} predecessor de p, onde $i \in \mathbb{N}$ e $p \in \mathbb{D}$, sendo que $pred(p,0) = p$, e $pred(p,i)$ é o predecessor de $pred(p,i-1)$ para $i > 0$. Função introduzida na página 24, Capítulo 4.

R **:** Mesmo que *nível de releitura*.

S_{avg} **:** Tamanho médio de sessão dos pares em um sistema P2P, como definido na Seção 4.4.

Sistema Auto-reorganizável : No contexto desta tese, consideramos como *auto-reorganizáveis* aqueles sistemas que são capazes de adaptar-se automaticamente à adesão, saída ou falha de qualquer de seus participantes [83].

Sistema BASE : Sistema de arquivos diretamente subjacente a um sistema D1HF, como definido na Seção 7.4.

Sistema Dinâmico : No contexto desta tese, consideramos como *dinâmicos* aqueles sistemas com uma frequência significativa de adesões e partidas de participantes.

SHDHT : Acrônimo do termo em inglês *Single-Hop DHT*, ou seja, DHT que é capaz de resolver consultas com um único salto.

$succ(p,i)$ **:** Função que define o i_{esimo} sucessor de p, onde $i \in \mathbb{N}$ e $p \in \mathbb{D}$, sendo que $succ(p,0) = p$, e $succ(p,i)$ é o sucessor de $succ(p,i-1)$ para $i > 0$. Função introduzida na página 24, Capítulo 4.

$stretch(p,k)$ **:** Para qualquer $p \in \mathbb{D}$ e $k \in \mathbb{N}$, $stretch(p,k) = \{\forall p_i \in \mathbb{D} \mid p_i = succ(p,i) \wedge 0 \leq i \leq k\}$ [51]. Função apresentada na página 24, Capítulo 4.

Tamanho de Sessão : Como definido na Seção 4.4, é o intervalo de tempo que um par fica continuamente conectado ao sistema P2P, ou seja, o intervalo de tempo entre a entrada de um par no sistema e a sua subsequente saída. Mesmo que *duração de sessão*.

Taxa de Releitura : Mesmo que *nível de releitura*.

TOR : Acrônimo do termo em inglês *Top Of Rack*, ou seja, topo de bastidor. Termo normalmente usado para denominar comutadores de rede (comumente Ethernet) que são usados como folhas em topologias em árvore gorda, típicas de multicomputadores Beowulf e outras classes de agregados computacionais. Nestas soluções, cada bastidor aloja alguns nós computacionais (em geral entre três e seis dezenas), que são diretamente conectados a um comutador TOR localizado em seu próprio bastidor (usualmente com padrão 1000baseT), que por sua vez tem conexões ao nível imediatamente superior da árvore gorda. Pode-se ainda implementar redundância de comutadores TOR, instalando-se (ao menos) dois destes comutadores por bastidor, e com cada nó computacional tendo duas (ou mais) conexões distribuídas por comutadores distintos. Apesar desta denominação ser comumente usada para comutadores Ethernet, topologias similares são também usadas com redes Infiniband ou FC.

TTL : Acrônimo do termo em inglês *Time To Live*. EDRA usa os contadores TTL de uma maneira diferente da usual, já que um evento que tenha sido conhecido com $TTL = l$, $l > 0$, não será propagado através de apenas uma mensagem com $TTL = l - 1$, e sim através de l mensagens, cada uma com um valor de TTL no intervalo $[0 : l)$.

δ_{avg} **:** Atraso médio das mensagens em um sistema D1HT, como definido na Seção 4.3.

Θ **:** Duração de um intervalo Θ, como definido na Seção 4.1, e que é ajustada dinamicamente segundo mecanismo apresentado na Seção 4.6.

ρ **:** Logaritmo do tamanho de um sistema D1HT, como definido na Equação 4.1, Seção 4.1.

Referências Bibliográficas

[1] I. Abraham, A. Badola, D. Bickson, D. Malkhi, S. Maloo, and S. Ron. Practical locality-awareness for large scale information sharing. In *Proceedings of IPTPS*, February 2005.

[2] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. *SIGOPS Operating Systems Reviews*, 36(SI):1–14, 2002.

[3] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on cooperation in BitTorrent communities. In *Proceedings of the 3rd SIGCOMM Workshop on Economics of P2P Systems*, August 2005.

[4] H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, 2003.

[5] Luiz A. Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Science. Morgan & Claypool Publishers, 2009.

[6] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *Proceedings of NSDI*, March 2004.

[7] A. Bellissimo, P. Shenoy, and B. Levine. Exploring the use of BitTorrent as the basis for a large trace repository. Technical Report 04-41, Department of Computer Science, U. of Massachusetts, June 2004.

[8] D.A. Bryan, B.B. Lowekamp, and C. Jennings. SOSIMPLE: A Serverless, Standards-based, P2P SIP Communication System. In *Proceedings of the 2005*

International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA), June 2005.

[9] Andrea Calvagna, Livio Di Lorenzo, and Giuseppe Tropea. Video-on-Demand for P2P Communities. In *Proceedings of The Tenth International Conference on Distributed Multimedia Systems (DMS)*, September 2004.

[10] Chimera. http://current.cs.ucsb.edu/projects/chimera/, January 2010.

[11] J. Chu, K. Labonte, and B. Levine. Availability and locality measurements of peer-to-peer file systems. In *Proceedings of SPIE*, July 2002.

[12] James Cipar, Mark D. Corner, and Emery D. Berger. TFS: a transparent file system for contributory storage. In *Proceedings of the 5th USENIX conference on File and Storage Technologies (FAST)*, pages 28–28, 2007.

[13] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies*, 2001.

[14] Manuel Costa, Miguel Castro, Antony Rowstron, and Peter Key. PIC: Practical Internet Coordinates for Distance Estimation. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, March 2004.

[15] L. Cox and B. Noble. Pastiche: Making backup cheap and easy. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[16] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using Chord. In *Proceedings of IPTPS*, March 2002.

[17] Curt Cramer and Thomas Fuhrmann. Proximity Neighbor Selection for a DHT in Wireless Multi-Hop Networks. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P-2005)*. IEEE Computer Society, August 2005.

[18] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-Area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[19] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[20] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proceedings of IPTPS*, February 2003.

[21] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, 2007.

[22] M. Dischinger, K. Gummadi, A. Haeberlen, and S. Saroiu. Characterizing residential broadband networks. In *Proceedings of IMC*, October 2007.

[23] P. Fonseca, R. Rodrigues, A. Gupta, and B. Liskov. Full information lookups for peer-to-peer overlays. *IEEE Transactions on Parallel and Distributed Systems*, 20(9), September 2009.

[24] P. Fraigniaud and P. Gauron. The content-addressable network D2B. Technical Report LRI-1349, Universie de Paris Sud, January 2003.

[25] G. Gensini, M. Yacoub, and A. Conti. The concept of quarantine in history. *Elsevier Journal of Infection*, 49, 2004.

[26] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[27] Abhishek Ghose, Jens Grossklags, and John Chuang. Resilient Data-Centric Storage in Wireless Sensor Networks. *IEEE Distributed Systems Online*, 4(11), 2003.

[28] IBM GPFS. *Concepts, Planning, and Installation Guide - Version 3.4*. IBM, 2010.

[29] A. Gupta, B. Liskov, and R. Rodrigues. Efficient Routing for Peer-to-Peer Overlays. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[30] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of IPTPS*, February 2003.

[31] Halldor Isak Gylfason, Omar Khan, and Grant Schoenebeck. Chora: Expert-based P2P web search. In *Proceedings of the Workshop on Agents and P2P Computing (AP2PC)*, 2006.

[32] Mohamed Hefeeda, Ahsan Habib, Boyan Botev, Dongyan Xu, and Bharat Bhargava. PROMISE: Peer-to-peer media streaming using CollectCast. In *Proceedings of the 11th ACM International Conference on Multimedia (MM-03)*, pages 45–54, November 2003.

[33] John S. Heidemann and Gerald J. Popek. File-system development with stackable layers. *ACM Transactions on Computer Systems (TOCS)*, 12(1):58–89, February 1994.

[34] Y. Charlie Hu, Saumitra M. Das, and Himabindu Pucha. Exploiting the Synergy between Peer-to-Peer and Mobile Ad Hoc Networks. In *Proceedings of HotOS'03: 9th Workshop on Hot Topics in Operating Systems*, pages 37–42. USENIX, May 2003.

[35] R. Huebsch, J. Hellerstein, N. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of the International Conference on Very Large Databases*, September 2003.

[36] M. Kaashoek and D. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of IPTPS*, February 2003.

[37] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the Symposium on Theory of Computing*, May 1997.

[38] A. Keromytis, V. Misra, and D. Rubenstein. SOS: An architecture for mitigating DDoS attacks. *Journal on Selected Areas in Communications*, January 2004.

[39] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-Peer support for massively multiplayer games. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, March 2004.

[40] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, November 2000.

[41] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the Kazaa Network. In *Proceedings of the 3rd IEEE Workshop on Internet Applications (WIAPP'03)*, 2003.

[42] J. Li, J. Stribling, T. Gil, R. Morris, and F. Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proceedings of IPTPS*, 2004.

[43] J. Li, J. Stribling, R. Morris, and M. Frans. A performance vs. cost framework for evaluating DHT design tradeoffs. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, March 2005.

[44] J. Li, J. Stribling, R. Morris, and M. Kaashoek. Bandwidth-efficient management of DHT routing tables. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[45] L. Li, Z. Chunhong, W. Mi, Y. Zhang, T. Ma, and J. Yang. SFDHT: A DHT Designed for Server Farm. In *Proceedings of GLOBECOM*, November 2009.

[46] Q. Lian, W. Chen, Z. Zhang, S. Wu, and B. Zhao. Z-ring: Fast prefix routing via a low maintenance membership protocol. In *ICNP*, 2005.

[47] J. Liang, R. Kumar, and K. Ross. The FastTrack overlay: A measurement study. *Computer Networks*, 50(6), Apr 2006.

[48] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, July 2002.

[49] M. Madruga, S. Loest, C. Maziero, and A. Santin. Um Sistema de Arquivos Distribuído Peer-to-Peer Tolerante a Faltas usando Arquivos Transparentes. In *Proccedings of Simpósio Brasileiro de Redes de Computadores (SBRC)*, 2009.

[50] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *Proceedings of IMC*, November 2009.

[51] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, July 2002.

[52] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of IPTPS*, March 2002.

[53] A. Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured Superpeers: Leveraging heterogeneity to provide constant-time lookup. In *Proceedings of the 3rd Workshop on Internet Applications*, June 2003.

[54] Luiz R. Monnerat. D1HT source code available under GPL license from http://www.lcp.coppe.ufrj.br/D1HT, December 2009.

[55] Luiz R. Monnerat and Claudio L. Amorim. D1HT: A Distributed One Hop Hash Table. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2006. Previously published as TR ES-676/05, UFRJ, May 2005 (available from http://www.lcp.coppe.ufrj.br).

[56] Luiz R. Monnerat and Claudio L. Amorim. Peer-to-Peer Single Hop Distributed Hash Tables. In *Proceedings of IEEE GLOBECOM*, November 2009. Available from http://www.lcp.coppe.ufrj.br/D1HT/.

[57] Luiz R. Monnerat and Claudio L. Amorim. Low Latency Distributed Hash Tables. 2010. Submitted for publication.

[58] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: A Read/Write Peer-to-Peer File System. In *Proceedings of 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[59] EMC Legato Networker. http://www.emc.com/products/detail/software/networker.htm, August 2010.

[60] NIST. Secure Hash Standard (SHS). *FIPS Publication 180-1*, April 1995.

[61] Overnet. www.edonkey2000.com. October 2005.

[62] J. Panetta, P. Souza, C. Cunha, F. Roxo, S. Sinedino, I. Pedrosa, A. Romanelli, L. Monnerat, L. Carneiro, and C. Albrecht. Computational characteristics of production seismic migration and its performance on novel processor architectures. In *19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, October 2007.

[63] J. Panetta, T. Teixeira, P. Souza, C. Cunha, D. Sotelo, F. Roxo, S. Sinedino, I. Pedrosa, A. Romanelli, L. Monnerat, L. Carneiro, and C. Albrecht. Accelerating Kirchhoff Migration by CPU and GPU Cooperation. In *Proceedings of the 21st International Symposium on Computer Architecture and High Performance Computing*, October 2009.

[64] D. Patterson. Latency lags bandwith. *Communications of ACM*, 47(10):71–75, 2004.

[65] Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel, and David Hitz. NFS version 3: Design and Implementation. In *Proceedings of the Summer 1994 USENIX Technical Conference*, pages 137–151, 1994.

[66] C. Plaxton, R. Rajaraman, and A. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, June 1997.

[67] IEEE POSIX. http://standards.ieee.org/regauth/posix/, August 2010.

[68] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The Bittorrent P2P File-sharing System: Measurements and Analysis. In *Proceedings of IPTPS*, February 2005.

[69] CPUburn Project. http://pages.sbcglobal.net/redelm/. January 2010.

[70] Venugopalan Ramasubramanian and Emin Sirer. Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[71] Venugopalan Ramasubramanian and Emin Sirer. The design and implementation of a next generation name service for the Internet. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. ACM, September 2004.

[72] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2001.

[73] IETF RFC3530. http://tools.ietf.org/html/rfc3530, August 2010.

[74] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The Oceanstore Prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, March 2003.

[75] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proceedings of the 2004 USENIX Technical Conference*, June 2004.

[76] C. Riley and C. Scheideler. A distributed hash table for computational grids. In *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[77] Matei Ripeanu, Adriana Iamnitchi, and Ian T. Foster. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1):50–57, January 2002.

[78] J Risson, A Harwood, and T Moors. Stable high-capacity one-hop distributed hash tables. In *Proceedings of ISCC*, June 2006.

[79] John Risson and Tim Moors. Survey of research towards robust peer-to-peer networks: Search methods. *Computer Networks*, 50(17):3485–3521, 2006.

[80] Vladimir Rocha, Marco A. S. Netto, and Fabio Kon. Descoberta de Recursos em Grades Computacionais Utilizando Estruturas P2P. In *Proceedings of the 24th Brazilian Symposium on Computer Networks (SBRC)*, May 2006.

[81] R. Rodrigues and C. Blake. When multi-hop peer-to-peer routing matters. In *Proceedings of IPTPS*, February 2004.

[82] R. Rodrigues, B. Liskov, and L. Shrira. The design of a robust peer-to-peer system. In *Proceedings of the 10th ACM SIGOPS European Workshop*, September 2002.

[83] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of Middleware*, November 2001.

[84] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[85] S. Saroiu, P. Gummadi, and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking (MMCN)*, January 2002.

[86] Philip Schawn. Lustre: Building a File System for 1,000-node Clusters. In *Proceedings of the 2003 Linux Symposium*, July 2003.

[87] F. Schintke, T. Schutt, and A. Reinefeld. A framework for self-optimizing Grids using P2P components. In *Proceedings of the 14th IEEE DEXA*, 2003.

[88] Frank Schmuck and Roger Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, pages 231–244, 2002.

[89] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govindan, and Deborah Estrin. Data-centric storage in sensornets. *Computer Communication Review*, 33(1):137–142, 2003.

[90] Jonhnny Weslley Silva and Francisco Brasileiro. When BashReduce Met BeeFS. 2010. Submitted for publication.

[91] Kundan Singh and Henning Schulzrinne. Peer-to-peer internet telephony using SIP. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 63–68, June 2005.

[92] Carla A Souza, Ana Clara Lacerda, Jonhnny W Silva, Thiago Emmanuel Pereira, Alexandro Soares, and Francisco Brasileiro. BeeFS: Um Sistema de Arquivos Distribuído POSIX Barato e Eficiente para Redes Locais. In *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2010)*, May 2010.

[93] M. Steiner, T. En-Najjary, and E. Biersack. Long term study of peer behavior in the KAD DHT. *IEEE/ACM Transactions on Networking*, October 2009.

[94] T. Sterling, D. Becker, D. Savarese, and J. Dorband. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of ICPP*, August 1995.

[95] W. Richard Stevens, Alexandros Biliris, and Euthimios Panagos. *UNIX Network Programming, The Sockets Networking API*. Addison-Wesley, Third edition, 2004.

[96] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Frans Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, February 2003.

[97] Stribling, Councill, Li, Kaashoek, Karger, Morris, and Shenker. OverCite: A Cooperative Digital Research Library. In *Proceedings of IPTPS*, February 2005.

[98] M. Szeredi. File System in User Space. http://fuse.sourceforge.net, June 2010.

[99] Chunqiang Tang, Melissa J. Buco, Rong N. Chang, Sandhya Dwarkadas, Laura Z. Luan, Edward So, and Christopher Ward. Low traffic overlay networks with large routing tables. In *Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*, pages 14–25. ACM, June 2005.

[100] TOP500. www.top500.org, August 2010.

[101] Aline Viana, Marcelo de Amorim, Serge Fdida, and José Rezende. An Underlay Strategy for Indirect Routing. *ACM Wireless Networks and Applications*, 10(6):747–758, November 2004.

[102] J. Xu. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, March 2003.

[103] Alexander Yip, Benjie Chen, and Robert Morris. Pastwatch: a distributed version control system. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, May 2006.

[104] E. Zadok. Writing Stackable File Systems. *Linux Journal*, 2003(109):22–25, May 2003.

[105] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A Global-scale Overlay for Rapid Service Deployment. *Journal on Selected Areas in Communications*, January 2004.

[106] F. Zhou, L. Zhuang, B. Zhao, L. Huang, A. Joseph, and J. Kubiatowicz. Approximate object location and spam filtering on peer-to-peer systems. In *Proceedings of Middleware*, June 2003.