Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Avaliação de Desempenho do Hyper-DHT para armazenamento e recuperação em Big Data

Vilson Norberto dos Santos Junior

CASCAVEL
2018

Vilson Norberto dos Santos Junior

Avaliação de Desempenho do Hyper-DHT para armazenamento e recuperação em Big Data

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Dr. Luiz Antonio Rodrigues

CASCAVEL
2018

Vilson Norberto dos Santos Junior

Avaliação de Desempenho do Hyper-DHT para armazenamento e recuperação em Big Data

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Dr. Luiz Antonio Rodrigues (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Guilherme Galante
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Márcio Seiji Oyamada
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 30 de novembro de 2018

AGRADECIMENTOS

Primeiramente a minha família, em especial ao meu pai Vilson Norberto dos Santos, a minha mãe Mery Cristina Silveira dos Santos e a minha avó Maria José dos Santos por todo o suporte, apoio e orações para que eu chegasse até essa etapa da minha vida.

Aos meus amigos e colegas de curso que me acompanharam por toda esta jornada e partilharam das mesmas dificuldades, desafios e felicidades. Agradeço em especial aos meus amigos do CLÃ, amigos estes que levarei para toda a minha vida, que me proporcionaram momentos de alegria, risadas e muitas horas de estudo (normalmente em cima da hora).

Gostaria de agradecer também a Universidade Estadual do Oeste do Paraná - UNIOESTE, mais especificamente ao corpo docente do curso de Ciência da Computação e ao meu professor e orientador Luiz Antonio Rodrigues, por toda a ajuda e atenção, também ao professor Guilherme Galante por me ajudar tanto para permitir minha chegada até aqui me ajudando nos horários para possibilitar minha graduação. De maneira geral, a todos que de alguma forma colaboraram para minha chegada até aqui.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CAN	*Content-Addressable Network*
DHT	*Distributed Hash Table*
DiVHA	*Distributed Virtual Hypercube Algorithm*
EDRA	*Event Detection and Propagation Algorithm*
Hi-ADSD	*Hierarchical Adaptive Distributed System-level Diagnosis*
IP	*Internet Protocol*
P2P	*Peer-to-Peer*

Lista de Símbolos

θ Classe Assintótica
i Índice do Vértice
d Dimensão do Hipercubo

Sumário

Resumo

Big Data é utilizado para diferentes finalidades como armazenamento de dados simples ou análise do comportamento humano, sendo capaz de lidar com diferentes tipos de dados, sendo estes estruturados ou não estruturados. Para que *Big Data* seja utilizado extraindo sua total eficiência uma forma de processar seus dados se torna necessária. Para este processamento muitas vezes são utilizadas as DHTs (*Distributed Hash Table*) que utilizam o conceito de <chave, valor> das tabelas *hash* usuais e realizam o armazenamento dos dados distribuídos em vários nodos do sistema. Este trabalho apresenta uma análise do desempenho de um DHT de salto único denominada HyperDHT nas operações de *lookup* e inserção de dados. Seu modelo é associado a um grafo cuja topologia formada é baseada em um hipercubo virtual. Para realizar uma inserção é necessário primeiramente determinar qual o vértice responsável pela chave a ser inserida, em seguida deve-se encontrar o *peer* responsável por vértice e caso o *peer* responsável esteja disponível será realizada a inserção. A operação de *lookup* ocorre de maneira muito similar, este último implica em indicar precisamente qual *peer* é responsável pela chave buscada. Para a análise desta DHT foi utilizado o simulador PeerSim, nele foram implementadas as DHTs HyperDHT e Chord, estas foram comparadas em diferentes cenários com diferentes tamanhos de rede em questão de quantia de participantes no sistema. Após a realização dos testes foi confirmado que o HyperDHT é melhor no contexto *Big Data* para a realização de *lookup* e *put* que o Chord, esta uma DHT já consolidada. Por conta disso o HyperDHT mostra-se confiável e eficiente para seu uso em *Big Data*.

Palavras-chave: Bid Data, Hipercubo, HyperDHT, Simulação.

Capítulo 1

Introdução

Um sistema distribuído é aquele no qual hardware e o software estão conectados por uma rede de computadores, como uma rede local ou a Internet, e que se comunicam e coordenam as suas ações enviando mensagens entre si a fim de realizar uma determinada tarefa (COULOURIS et al., 2011). Estes sistemas tem com objetivo o compartilhamento de recursos, como hardware, software e dados.

A área de sistemas distribuídos, possui uma série de desafios, entre eles está a troca de informações em ambientes sujeitos a falhas. Neste caso, o sistema é dito tolerante a falhas se é capaz de continuar a sua operação corretamente mesmo em caso de falhas, ainda que de forma degradada. Para a solução destes desafios os sistemas devem implementar técnicas que possibilitem a detecção, correção e/ou mascaramento das falhas.

Nos dias atuais, o avanço da tecnologia nos permite armazenar, organizar e analisar dados de forma muito mais fácil e frequente. O constante crescimento e dependência tecnológica nos obriga a armazenar grandes quantidades de dados, conjunto esse que pode ser classificado como *Big Data*. Sharma e Mangat (2015) definem *Big Data* como uma técnica inovadora para armazenar, distribuir, gerenciar, visualizar e analisar grandes volumes de dados (estruturados e não estruturados) com grande velocidade.

O conceito *Big Data* surgiu no momento em que bancos de dados usuais já não davam mais conta dos novos tipos de dados, os dados sem estrutura, como fotos, vídeos, mídia social e comportamento humano, com estes tipos de dados excedendo a capacidade dos modelos tradicionais de processamento de dados. *Big Data* pode ser utilizado de várias maneiras com os dados pertencendo a sua base, os chamados 5Vs, sendo estes. *Volume* que refere-se a quantia de dados a serem processados, *Variedade* que refere-se a origem dos dados a serem processados

e como podem ser classificados, *Velocidade* que refere-se a velocidade da geração de novos dados a serem processados (GROVER; JOHARI, 2015), *Valor* refere-se ao resultado produzido depois de todo o processamento de dados. Por fim *Veracidade* que refere-se a qualidade dos dados (SHARMA; MANGAT, 2015).

Para que o *Big Data* seja utilizado para suas diferentes finalidades uma forma de processamento dos dados é necessário, usualmente são utilizadas as DHT (*Distributed Hash Table*) para este tratamento. DHTs utilizam topologias virtuais para a organização dos nodos que armazenam os dados, aumentando assim o desempenho das operações de armazenamento e recuperação dos mesmos.

DHTs foram originalmente utilizadas em redes P2P (*Peer-to-Peer*) para manter informações distribuídas sobre múltiplos nós do sistema. Elas podem construir sistemas distribuídos garantindo a descentralização, escalabilidade e tolerância a falhas (KOPPE, 2013). As DHTs utilizam o mesmo conceito das tabelas *hash* usuais, composta por pares do tipo <chave, valor>. Para realizar *lookups* as DHTs utilizam as chamadas tabelas de roteamento, permitindo que o nodo solicitante obtenha o valor relacionado a chave consultada. Todos os nodos de um sistema DHT possuem uma tabela de roteamento onde são armazenadas referências a outros nodos, variando-se as tabelas de acordo com a topologia da DHT. Cada consulta realizada para se obter um valor a partir de uma chave na DHT é chamada de salto.

São divididas em duas categorias as DHTs de múltiplo salto, este tipo opta por utilizar tabelas de roteamento parciais divididas entre os nodos da rede, ou seja, elas optam por soluções nas quais as consultas são realizadas em múltiplos saltos, objetivando diminuir o trafego destinado para manutenção do sistema. A outra categoria de DHT é chamado de salto único. Esta categoria objetiva reduzir ao máximo a latência das consultas, realizando assim de maneira rápida e eficiente a disseminação de eventos. Em DHTs de salto único todos os nodos possuem conhecimento sobre todos os participantes do sistema.

O HyperDHT (KOPPE, 2013) além de ser a DHT utilizada neste trabalho é uma DHT de salto único no qual seu modelo é associado a um grafo cuja topologia formada é baseada em um hipercubo virtual. Tem como objetivo prover uma solução que de forma rápida, eficiente e escalável possa realizar a localização de um objeto ou uma informação distribuída em uma rede.

Diferente das outras DHTs que deixam o posicionamento dos novos participantes por conta da função *hash*, neste sistema o protocolo de entrada posiciona deterministicamente o novo participante no local onde o mesmo é mais necessário, diferenciando-a assim das demais DHTs. Por conta disso, o balanceamento de carga no HyperDHT é mais homogêneo em relação as outras DHTs. O HyperDHT assim como as outras DHTs de salto único mantêm em cada *peer* uma tabela de roteamento completa, com referência para todos os demais participantes da rede, em que cada entrada desta tabela é incluído o identificador de determinado *peer* a seu endereço de rede. Em caso de falha de nodo, seus dados são replicados a um de seus vizinhos.

Este trabalho tem como objetivo analisar o desempenho do HyperDHT nas operações de *lookup* e inserção de dados no contexto de *Big Data*, bem como a disponibilidade dos mesmos em casos de falhas de nodos, comparando-o com DHTs já estabelecidas. O simulador PeerSim foi utilizado para a realização de testes e análise do HyperDHT em cenários com 8 até 32768 participantes em potência de 2 (hipercubos de 2^3 até 2^{15} nodos) para as operações de *put* e *lookup*. O mesmo simulador foi utilizado para a realização dos testes das DHTs comparadas com o HyperDHT para a análise de seu desempenho.

O restante deste trabalho está organizado nos seguintes capítulos. O Capítulo 2 apresenta os principais temas utilizados para a realização deste trabalho, no qual o mesmo aborda a fundamentação teórica para a realização do mesmo. O Capítulo 3 aborda a implementação do HyperDHT no *framework* Peersim e os resultados dos testes de simulação. O Capítulo 4 apresenta a conclusão deste trabalho bem como a sugestão de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os principais temas utilizados para a realização deste trabalho, abordando especialmente as DHTs com foco em *Big Data*. A Seção 2.1 apresenta a definição e algumas informações sobre sistemas distribuídos. A Seção 2.2 aborda os conceitos e definições de *Big Data*. As DHTs, foco deste trabalho, são apresentadas na Seção 2.3, abordando o conceito, definição e exemplos de algumas DHTs importantes e relevantes para esta pesquisa. O HyperDHT, que é utilizado neste trabalho, é discutido na Seção 2.3.3.

2.1 Sistemas Distribuídos

Um sistema distribuído é uma coleção de computadores independentes que aparece para seus usuários como um sistema coerente e único (TANENBAUM; STEEN, 2006), visando combinar a escalabilidade à transparência de localização. Este tipo de sistema tem como um dos principais objetivos a troca de recursos, sendo esses mensagens, dados e outros.

A comunicação entre os processos que compõem o sistema distribuído em uma rede de computadores é feita por troca de mensagens. Tal comunicação pode ocorrer basicamente de três formas: um-para-um (*unicast*), um-para-muitos (*multicast*) e um-para-todos (*broadcast*) (HADZILACOS; TOUEG, 1994). No modelo um-para-um, um par de processos é conectado por cada enlace com o intuito de enviar informações para um destinatário por vez. No modelo um-para-muitos, as informações podem ser enviadas para um grupo de processos. No modelo um-para-todos, as mensagens são enviadas a partir de um emissor para todos os processos do sistema.

A área de sistemas distribuídos, por sua necessidade e complexidade, possui uma série de

desafios. Dentre eles a troca de informações sem que ocorra a falha de dados, ou caso ocorra, que o sistema seja capaz de corrigi-las sem ocasionar grandes danos ao sistema. Para solucionar tais desafios, os sistemas ditos tolerantes devem implementar técnicas que possibilitem a detecção, correção e/ou mascaramento das falhas, sejam elas de processamento, comunicação ou maliciosas.

2.1.1 Detectores de Falhas e Diagnóstico Distribuído

Uma falha em um sistema distribuído pode ocorrer basicamente durante o processamento e/ou durante a comunicação. Os principais tipos de falhas incluem as falhas de colapso (*crash*), temporização, omissão e bizantinas (RODRIGUES, 2014). As falhas por omissão são ocasionadas pela falta de envio e/ou não recebimento das mensagens, podendo ser causada de diferentes formas, sendo estas falta de espaço nos *buffers* do sistema operacional ou mesmo por erros de transmissão. As falhas por temporização acontecem quando *timeouts* ocorrem no sistema, ou seja, a mensagem é entregue com maior atraso que o permitido pelos processos.

Nas falhas por colapso o processo para de responder, ou seja, não executa qualquer ação, sem resposta a estímulos exteriores. Por conta disso o processo não realiza a troca de mensagens ou mesmo executa o processamento. Falhas bizantinas, também conhecidas como arbitrárias, incluem além de falhas de comunicação e processo, as falhas originadas por comportamentos maliciosos (RODRIGUES, 2014). Portanto, um sistema distribuído deve ser capaz de detectar falhas e recuperar-se de forma automática, sem que o usuário perceba a falha do sistema. Um sistema tolerante a falhas está fortemente relacionado a um sistema confiável. A confiabilidade engloba vários requisitos úteis que um sistema distribuído deve seguir, como disponibilidade, confiabilidade, segurança e capacidade de manutenção (TANENBAUM; STEEN, 2006).

O desenvolvimento de soluções tolerantes a falhas é necessário para que o sistema possa ser usado de forma continuada pelo usuário, sem interrupções ou travamento do sistema. Uma das principais abordagens de detecção consiste na troca de mensagens entre os processos, para que assim possam saber entre si os estados dos outros participantes do sistema e realizar o diagnóstico, com o estado (falho ou sem falha) de cada participante. Após certo tempo, caso não obtenham resposta, o processo pode ser considerado falho, iniciando assim o procedimento para a adaptação dos processos no sistema.

O algoritmo Hi-ADSD (*Hierarchical Adaptive Distributed System-level Diagnosis*) (DUARTE; NANYA, 1998) possui uma topologia baseada em um hipercubo, onde o mesmo possui características topológicas importantes como simetria, diâmetro logarítmico e boas propriedades de tolerância a falhas (KRULL; WU; MOLINA, 1992). Neste algoritmo é utilizado o conceito de *clusters*. Neste sistema o número de nodos é definido por 2^d, onde d é a dimensão do hipercubo. A Figura 2.1 apresenta um hipercubo com dimensão *d* = 3.

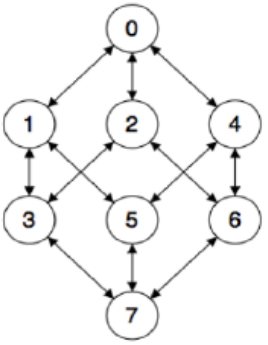

Figura 2.1: Grafo do Algoritmo Hi-ADSD. Fonte: (RUOSO, 2013)

Para obter as informações de diagnóstico sobre os nós de determinado *cluster* é efetuado um teste sobre um nodo pertencente ao *cluster*. Os processos de testes são efetuados a cada intervalo onde cada nodo testará os nodos do *cluster* específico até que um nodo sem falha seja encontrado. Os membros de cada *cluster s*, bem como a ordem em que os processos serão testados por um processo *i* são dados pela função $C_{i,s}$ (DUARTE; NANYA, 1998). A tabela 2.1 apresenta os resultados da função $C_{i,s}$ para um sistema de *d* = 3 dimensões.

Tabela 2.1: Resultado da $C_{i,s}$ para um sistema de *d* = 3 dimensões. Fonte: (DUARTE; NANYA, 1998)

S	$C_{0,s}$	$C_{1,s}$	$C_{2,s}$	$C_{3,s}$	$C_{4,s}$	$C_{5,s}$	$C_{6,s}$	$C_{7,s}$
1	1	0	3	2	5	4	7	6
2	2, 3	3, 2	0. 1	1, 0	6,7	7, 6	4, 5	5, 4
3	4, 5, 6	5, 6, 7, 4	6, 7, 4, 5	7, 4, 5, 6	0, 1, 2, 3	1, 2, 3, 0	2, 3, 0, 1	3, 0, 1, 2

O DiVHA (*Distributed Virtual Hypercube Algorithm*) (BONA et al., 1996) é um algoritmo de diagnóstico para sistemas distribuídos também baseado em hipercubo virtual. Seu objetivo é determinar um conjunto de testes a ser executado de acordo com o estado percebido pelo

sistema. O DiVHA calcula os enlaces do hipercubo, possibilitando assim a auto-organização dos nodos, sem que isso altere o diâmetro logarítmico do hipercubo. Ele é considerado dinâmico pois um nodo pode falhar e se recuperar constantemente (BONA et al., 1996). Este algoritmo utiliza o conceito de clusters assim como o algoritmo Hi-ADSD, no entanto neste é mantido apenas um único nodo testador em cada *cluster*. O algoritmo DiVHA determina o estado de todos os participantes do sistema em até $log_2 N$ rodadas de teste, onde a quantidade máxima de testes por rodada é $N * log_2 N$.

O algoritmo VCube (RUOSO, 2013) apresenta uma nova estratégia de testes para o algoritmo Hi-ADSD onde o mesmo tem como objetivo garantir em uma quantidade máxima de testes logarítmica preservando o mesmo limite para a latência. O hipercubo virtual é criado e mantido de acordo com as informações de diagnósticos obtidas por meio de um sistema de monitoramento de processos (RUOSO, 2013). Neste algoritmo, a cada execução do VCube, cada processo pode testar diferentes processos no sistema para realizar a verificação se os processos estão corretos ou falhos, processos corretos são aqueles que tem sua resposta ao teste realizado recebida corretamente dentro de um intervalo de tempo.

2.2 Big Data

Big Data pode ser definido como uma técnica inovadora para armazenar, distribuir, gerenciar, visualizar e analisar grandes volumes de dados (estruturados e não estruturados) com grande velocidade (SHARMA; MANGAT, 2015). Tal conceito surge no momento em que os bancos de dados usuais já não davam conta dos novos tipos de dados, os dados sem estrutura, como fotos, vídeos, mídia social e comportamento humano, com estes tipos de dados excedendo a capacidade dos modelos tradicionais de processamento de dados.

Big Data é muito importante para entendermos o mundo, sendo frequentemente encontrado em várias fontes incluindo mídia social, redes de sensores, aplicações científicas, negócios inteligentes e web logs (LI; LIAO, 2013). Seu diferencial não é apenas o armazenamento de grandes quantias de dados, mas também a forma que é utilizado, podendo ser de várias maneiras com os dados pertencendo a sua base, os chamados 5Vs, sendo estes:

- Volume: refere-se a quantidade de dados a serem processados;

- Variedade: Os dados podem ser determinados por meio de diversas formas com diferentes estruturas, como fotos, vídeos, arquivos específicos, entre outros. Trata-se da origem dos dados a serem processados e como podem ser classificados, sendo estruturados, não estruturados, semi-estruturados e outros (GROVER; JOHARI, 2015).
- Velocidade: refere-se a velocidade de geração de novos dados a serem processados.
- Valor: refere-se ao resultado produzido depois de todo o processamento de dados.
- Veracidade: refere-se a qualidade dos dados (SHARMA; MANGAT, 2015).

Na perspectiva da computação e engenharia de software, todos os conceitos e componentes em um mundo complexo devem ser armazenados, processados e transformados em dados com estruturas apropriadas em computadores (ZHU, 2018). Esta frase afirma a necessidade do *Big Data*, visto que o mesmo permite, diferente dos bancos de dados usuais, realizar o armazenamento, processamento e transformação dos dados recebidos. Empresas de grande porte aproveitam de diferentes formas o uso de *Big Data*, seja para análise de grandes volumes de dados ou para armazenamento. Para *Big Data* poder ser utilizado em diferentes finalidades, uma forma de tratamento dos dados é necessário, permitindo assim que os desenvolvedores possam de forma inteligente dar uma aplicação a estes dados. Dada a dificuldade do processamento e organização de uma quantidade tão grande de dados, podem ser utilizadas as DHT (*Distributed Hash Tables*).

2.3 Tabelas Hash Distribuídas

DHTs foram originalmente utilizadas em redes P2P (*Peer-to-Peer*) para manter informações distribuídas sobre múltiplos nós do sistema. Elas podem construir sistemas distribuídos garantindo a descentralização, escalabilidade e tolerância a falhas (KOPPE, 2013). São estruturas de dados que realizam o armazenamento de dados distribuídos em vários nodos do sistema. Para isso elas utilizam de o mesmo conceito das tabelas *hash* usuais, composta por pares do tipo <chave, valor>. A descentralização está relacionada a ausência de coordenação central, como ocorre no modelo cliente/servidor. A escalabilidade está relacionada a capacidade de auto-organização das DHTs. E por fim a tolerância a falhas que é garantida pelo modelo utilizado para construção da rede de sobreposição do sistema (KOPPE, 2013).

Para realizar *lookups* em uma DHT as mesmas utilizam as chamadas tabelas de roteamento, permitindo que o nodo solicitante obtenha o valor relacionado a chave consultada. Todos os nodos de um sistema DHT possuem uma tabela de roteamento na qual são armazenadas referencias a outros nodos, variando-se as tabelas de acordo com a topologia da DHT. Cada consulta realizada para se obter um valor a partir de uma chave na DHT é chamada de salto.

As DHTs possuem diferentes topologias, cada qual com suas características. Com o passar dos últimos anos muitas DHTs foram propostas, sendo elas classificadas basicamente em dois tipos: DHT de Múltiplos Saltos e DHT de Único Salto, sendo as mesmas melhor explicadas a seguir.

2.3.1 Tabelas Hash Distribuídas de Múltiplos Saltos

Estas DHTs optam por utilizar tabelas de roteamento parciais divididas entre os nodos da rede, ou seja, elas optam por soluções nas quais as consultas são realizadas em múltiplos saltos, objetivando diminuir o tráfego destinado para manutenção do sistema. Nos tópicos a seguir serão descritas duas DHTs de múltiplos saltos: Chord e CAN, sendo estas muito relevantes entre as DHTs e para este trabalho.

Chord

Chord (STOICA et al., 2001) é um modelo de DHT para mapeamento de chaves entre os nodos em um sistema distribuído. Usando uma topologia em forma de anel, o Chord trabalha de forma descentralizada e simétrica, podendo então adaptar-se automaticamente com a entrada e saída de nodos no sistema. O protocolo Chord especifica como encontrar a localização das chaves, como os novos nodos se juntam ao sistema e como recuperar da falha dos nodos existentes (STOICA et al., 2001).

O Chord usa uma *hash* consistente que distribui as chaves sobre os nodos de maneira a formar um anel, no qual as chaves assumem valores de 2^m, onde m é o número de bits utilizados. Por ser uma DHT, Chord utiliza uma hash que atribui a cada nodo e chave um identificador. A escolha do identificador de um nodo ocorre através da *hash* do nodo e seu endereço de IP, e o identificador da chave é selecionado com base na *hash* da mesma. Uma chave K é atribuída ao primeiro nodo onde o mesmo é igual ou maior que k, um nodo sucessor tem sua busca garantida por meio da função sucessor(k). Esta função possui como retorno o primeiro nodo sucessor,

para aquele portador da chave K no sistema, permitindo assim a equivalência em importância de todos os nodos no sistema e garantindo a descentralização do mesmo. A Figura 2.2 apresenta a operação de *lookup* em um sistema Chord onde as chaves sendo pesquisadas estão localizadas nos três nodos 0, 1 e 3. Neste exemplo a chave 1 está localizada no nodo 1, a chave 2 no nodo 3 e a chave 6 no nodo 0.

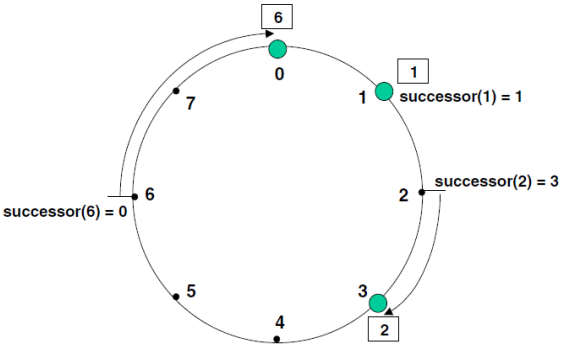

Figura 2.2: Operação básica de *lookup* em um sistema Chord. Fonte: (STOICA et al., 2001)

Chord utiliza uma tabela de roteamento chamada *finger table*, normalmente utilizada para a redução de latência das consultas. Cada nodo possui uma tabela com m entradas no máximo, onde m é o número de bits na chave. Cada entrada é formada pelo identificador e pelo endereço IP associado ao nodo (NETO, 2016). A classe assintótica para o *lookup* de uma chave no Chord é $\theta(logN)$, ou seja, em estado estável com N nodos no sistema, cada nodo mantém informação apenas sobre $\theta(logN)$ outros nodos resolvendo assim todas as operações de *lookup* via $\theta(logN)$ mensagens para os outros nodos.

CAN

CAN (*Content-Addressable Network*) (RATNASAMY et al., 2001) é uma DHT de múltiplos saltos, sua topologia é baseada em um espaço cartesiano virtual multi-dimensional de coordenadas, este espaço completamente lógico, sem alguma ligação com qualquer sistema de coordenadas físico. Neste sistema cada nodo possui um identificador e um valor. Os nodos no CAN se auto-organizam em uma rede de sobreposição que representam esse espaço de coordenada virtual (RATNASAMY et al., 2001). Um nodo aprende e mantém o endereço IP (chave) de todos os nodos que permanecem em zonas de coordenadas adjuntas a sua própria zona, sendo

assim seu nodo retém a coordenada da tabela de roteamento que segura o endereço de IP e a zona de coordenada virtual de cada um de seus vizinhos imediatos no espaço de coordenadas.

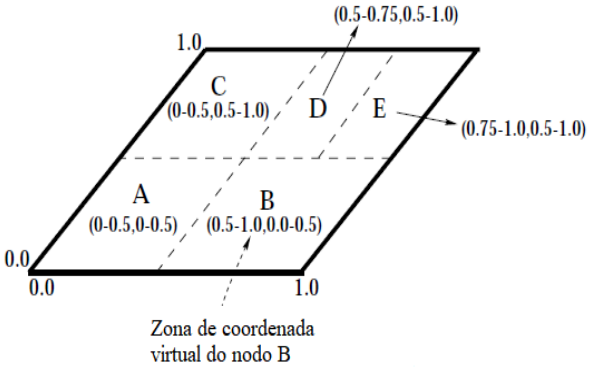

Figura 2.3: Espaço 2D com 5 nodos. Fonte: (RATNASAMY et al., 2001)

No sistema CAN a performance de roteamento é de $\theta(d.n^{(1/d)})$ saltos, sendo d a dimensão do espaço de coordenadas e n o numero de nodos no sistema. Levando em conta que um espaço cartesiano particionado em n regiões equivalentes, a distância média entre dois nodos quaisquer é de $\theta((d/4)(n^{1/d})$ (RATNASAMY et al., 2001). Para a alocação do CAN para seu crescimento incremental, o novo nodo para juntar-se ao sistema deve ser alocado em sua própria porção do espaço de coordenadas. O processo de alocação utiliza três passos:

1. Localização de um nodo já existente no CAN pelo novo nodo;

2. Em seguida com os mecanismos de rota do CAN, achar um nodo onde a zona será separada;

3. Para finalizar, os vizinhos da zona dividida devem ser notificados para que o roteamento possa incluir o nodo novo (RATNASAMY et al., 2001).

Como uma forma de melhorar a disponibilidade dos valores, CAN permite mapear um par chave/valor em diversos pontos diferentes no espaço de coordenadas, ou seja, o par chave/valor é replicado sobre vários nodos do sistema (NETO, 2016). Para os nodos que irão deixar o CAN, é necessário identificá-lo e em seguida o espaço livre no espaço cartesiano é realocado para um dos nodos adjacentes ao que deixou o sistema, realizando em fim a atualização das tabelas de roteamento dos nodos que ainda estão no sistema.

2.3.2 Tabelas Hash Distribuídas de Salto Único

DHTs de salto único objetivam reduzir ao máximo a latência das consultas, realizando assim de maneira rápida e eficiente a disseminação de eventos. Em DHTs de salto único todos os nodos possuem conhecimento sobre todos os participantes do sistema. No tópico a seguir é descrita uma DHTs de salto único, sendo esta D1HT.

D1HT

O D1HT (MONNERAT, 2010) é uma DHT de salto único que possui uma topologia baseada em um anel, por conta disso a associação das chaves aos nodos é feita no sentido horário e de forma ordenada, de forma semelhante a Chord (STOICA et al., 2001), já descrito anteriormente. D1HT utiliza técnicas de *hash* consistente onde o mesmo associa uma chave para cada participante do sistema. Neste modelo o IP e a chave são identificadores onde os mesmos são criptografados pela função SHA-1 (WANG; YIN; YU, 2005).

Neste sistema cada *peer* possui uma tabela de roteamento com os endereços IP de todos os participantes do sistema, podendo assim resolver as consultas com apenas um único salto, desde que a tabela local do *peer* esteja atualizada. O D1HT utiliza o algoritmo EDRA (*Event Detection and Propagation Algorithm*) para a atualização das tabelas de roteamento onde o mesmo permite notificar qualquer evento a todos os participantes da rede em tempo logarítmico. Para um novo participante adentrar o D1HT o mesmo deve conhecer pelo menos um nodo pertencente a rede para em seguida seguir o protocolo de adesão proposto no D1HT. O protocolo de adesão proposto exige que o novo participante atinja seus quatro requisitos, sendo estes (MONNERAT, 2010):

- A adesão de um par só poderá ser considerada completa quando o novo par tiver recebido a tabela de roteamento completa e todas as suas chaves;
- A disseminação na adesão de um par deve ser feita segundo o algoritmo EDRA;
- O novo par deverá ter garantia de recebimento de todas as notificações e eventos que ocorram enquanto sua adesão está sendo disseminada por seu sucessor ou predecessor;
- O protocolo de adesão deverá ser apto a receber extensões.

De modo a minimizar a latência em algumas situações, o D1HT pode fazer uso de informações de localidade mesmo que sua consulta seja resolvida utilizando apenas um único salto. D1HT usa uma arquitetura puramente P2P, mas sua topologia *plana* não impede que seja usado como um componente de soluções hierárquicas que procuraram explorar a heterogeneidade de seus participantes (MONNERAT, 2010).

2.3.3 HyperDHT

O HyperDHT (KOPPE, 2013) além de ser a DHT utilizada neste trabalho é uma DHT de salto único no qual seu modelo é associado a um grafo cuja topologia formada é baseada em um hipercubo virtual, tem como objetivo prover uma solução que de forma rápida, eficiente e escalável possa realizar a localização de um objeto ou uma informação distribuída em uma rede. Para que isso seja possível o HyperDHT utiliza a infraestrutura fornecida pelo DiVHA (*Distributed Virtual Hypercube Algorithm*) (BONA et al., 1996) para a implementação dos serviços necessários para uma DHT de salto único.

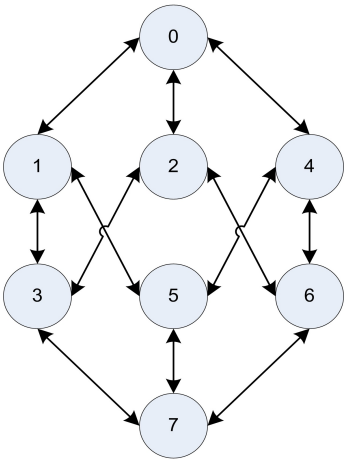

Figura 2.4: Rede virtual HyperDHT de 8 nodos. Fonte: (KOPPE, 2013)

O HyperDHT permite implementar um sistema onde os nodos sem falha conseguem determinar o estado de todos os participantes no tempo máximo $\log_2(N)$ rodadas de testes e o número máximo de testes por rodada é de $N * log_2(N)$ (KOPPE, 2013). As chances de resposta em salto único no HyperDHT são aumentadas graças a garantia de latência máxima vinda do sistema de diagnóstico, mais o fato de todos os nodos terem conhecimento completo do sistema, com isso o sistema consegue utilizar estratégias para maximizar estas chances de resposta. Di-

ferente das outras DHTs que deixam o posicionamento dos novos participantes por conta da função *hash*, seu protocolo de entrada posiciona deterministicamente o novo participante no local onde o mesmo é mais necessário, diferenciando-a assim das demais DHTs.

Por conta disso o balanceamento de carga no HyperDHT é mais homogêneo em relação as outras DHTs. Aos *peers* que vêm a entrar no sistema são atribuídos o posicionamento e a localização posterior dos objetos, realizado na forma de identificadores numéricos (chaves). Por meio de técnicas de *hash* consistente é realizado o particionamento da tabela *hash*.

Como dito anteriormente para a entrada de um *peer* o mesmo deve seguir o procolo de entrada, onde o mesmo com base na topologia atual da rede determina uma posição adequada para a alocação do novo *peer*. Diferentemente da entrada, a saída de um *peer* na rede pode se dar de duas formas, onde na primeira o *peer* anuncia sua saída para seus vizinhos e segue o protocolo de saída de forma que garanta a disponibilidade dos valores em um único salto, ou então ele simplesmente sai do sistema, ocasionando saltos extras para réplicas nas eventuais consultas em chaves pertencentes ao *peer* que saiu (KOPPE, 2013).

O HyperDHT assim como as outras DHTs de salto único mantêm em cada *peer* uma tabela de roteamento completa, com referência para todos os demais participantes da rede, no qual em cada entrada desta tabela estão o identificador de determinado *peer* a seu endereço de rede. Com esta tabela é possível realizar a operação de consulta *lookup(key)*, onde a mesma utiliza a tabela para localizar a chave de determinado *peer* e viabiliza a implementação das funções *put(key, value)* e *get(key)*.

Para manter o balanço de carga entra todos os participantes do sistema, ele utiliza a função criptográfica *SHA-1* de 160 bits para realizar o cálculo das chaves, assim é possível ter um conjunto de no máximo 2^{160} chaves. Esta associação é feita de uma forma onde cada vértice recebe uma faixa de valores determinada por $[i * 2^{160-d}, ..., (i+1) * 2^{160-d}]$, onde *i* representa o índice do vértice e *d* a dimensão do hipercubo.

O HyperDHT é um sistema considerado completamente conectado, ou seja, seus *peers* comunicam-se diretamente, sem utilizar intermediários. O *peer* no sistema pode estar em um de dois estados, sendo estes *disponível* ou *indisponível*, para que um *peer* esteja *disponível* o mesmo deve estar em condições operacionais perfeitas, caso contrário é considerado um *peer indisponível*, que são aqueles que, por algum motivo, apresentam um comportamento diferente

do esperado. O *peer* que passa do estado *indisponível* para o estado *disponível* pode manter as informações que ele armazenava anteriormente, porém o mesmo deve atualizar-se sobre as eventuais mudanças ocorridas na configuração de rede antes de voltar a prover seus serviços.

Os *peers* são capazes de realizar testes em outros *peers* e determinar corretamente o estado do *peer* testado. Os testes citados são realizados periodicamente, dentro de um intervalo de testes fixo determinado em função dos requisitos desejados para o sistema (KOPPE, 2013). De forma parecida com o OneHop DHT, o HyperDHT testa seus *peers* por meio de um procedimento executado pelo *peer* testador para o *peer* testado, assim caso haja falta de resposta ao ultrapassar o limite de tempo esperado sem obter respostas o *peer* que realizou o teste pode considerar o *peer* testado *indisponível*, caso contrário o mesmo o considerará *disponível*.

Em relação à replicação de dados em caso de falhas de nodos, o trabalho de (NETO; RODRIGUES; DUARTE, 2017) comparou diferentes estratégias de replicação para uso no HyperDHT.

Lookup e Put

O procedimento de *lookup* implica em indicar precisamente qual *peer* é responsável pela chave buscada. De forma a possibilitar que cada *lookup* seja realizado em um único salto, o HyperDHT mantém uma tabela de referências para todos os participantes do sistema. Neste sistema as consultas são resolvidas com base apenas no conhecimento local, fazendo com que assim seja possível a realização das operações *put* e *get* utilizando apenas um salto único.

Existe única uma exceção onde são necessários saltos adicionais para concluir as operações de *put* e *get*, essa situação acontece quanto um *lookup* é realizado em uma chave onde seu *peer* responsável se tornou indisponível e a notificação deste evento ainda está em faze de propagação (KOPPE, 2013). O pseudo-código da Figura 2.5 apresenta o algoritmo *lookup*, onde o mesmo recebe como parâmetro a chave k, a dimensão do hipercubo d e o grafo do hipercubo S. Primeiramente utilizando a expressão representada na linha 2 é determinado qual vértice possui a chave k. Em seguida na linha 3 é recebido como retorno da função *find_vertice_owner()* o identificador do *peer* responsável pelo vértice v. E para finalizar na linha 4 este identificador é utilizado para indexar a tabela de roteamento que contém as informações necessárias para realizar o contato com o *peer* i;

```
01. lookup ( k, d, S ) {
02.     v = floor( k / pow( 2, 160-d ) )
03.     i = find_vertice_owner ( v, S )
04.     RETURN address of peer i from routing table
05. }
```

Figura 2.5: Pseudo-código algoritmo *lookup*. Fonte: (KOPPE, 2013)

No HyperDHT o *put* de um dado ocorre de maneira muito similar ao *lookup*. Para realizar um *put* é necessário primeiramente determinar qual o vértice responsável pela chave a ser inserida. Subsequentemente encontrar o *peer* responsável por este vértice assim como mostrado na função *find_vertice_owner()*, após isso o *peer* responsável por este vértice será contatado e caso o mesmo esteja disponível será realizada a inserção.

Capítulo 3

Avaliação Experimental

Este capítulo apresenta a implementação do HyperDHT no simulador Peersim e os resultados dos testes de simulação. Com o objetivo de avaliar o desempenho do HyperDHT nas operações de *lookup* e *put* de dados no contexto do *Big Data*, bem como a disponibilidade dos mesmos em casos de falhas de nodos, serão construídos diferentes cenários de teste, variando-se o número de nodos integrantes do sistema e a quantidade de operações de armazenamento e recuperação. O HyperDHT foi comparado com o Chord, o mesmo possui uma implementação disponível no site do projeto do PeerSim.

3.1 O Simulador PeerSim

O PeerSim (MONTRESOR; JELASITY, 2009) é um simulador *open source* para sistemas P2P desenvolvido em linguagem Java. Foi desenvolvido para ser escalável e dinâmico para grandes redes P2P, permitindo assim simulações de redes P2P estruturadas e não estruturadas.

PeerSim é composto por dois mecanismos de simulação: o mais simplificado, que é baseado em ciclos; e o baseado em eventos. Estes mecanismos possuem uma configuração flexível para a realização das simulações. Os mecanismos de simulação que compõem o PeerSim possuem características para diferentes tipos de simulações, onde o baseado em ciclos é utilizado para permitir maior escalabilidade. No entanto, o mesmo ignora os detalhes da camada de transporte no protocolo de comunicação. Já o mecanismo baseado em eventos é utilizado para simulações mais realísticas, visto que o mesmo fornece suporte a camada de transporte.

3.2 Ambientes de Teste

Os ambientes de testes, tem como característica simular cenários que possuem tráfego de dados semelhantes a uma rede de computadores *Big Data*, sendo composta por *nodos*, *roteadores* e *enlaces* de comunicação para troca de mensagens entre os participantes da rede. Inicialmente para a validação do algoritmo foram realizados testes com redes menores, de 8 até 1024 participantes em potência de 2 (hipercubos de 2^3 até 2^{10} nodos). Em seguida, foram realizados testes com cenários que representam sistemas *Big Data* em termos de número de operações e quantidade de participantes na rede para a avaliação do desempenho do HyperDHT em redes *BigData*.

Os valores utilizados em questões de quantidades de nodos no sistema, volume de entrada e saída foram baseados em grandes redes de dados, como Facebook, Google, Twitter, Instagram e Amazon. O Facebook, por exemplo, conta com pouco mais que 2,2 bilhões de usuários em sua rede, onde cada usuário é um participante no sistema. A cada minuto, em torno de 510 mil comentários são postados, 293 mil status são atualizados e 136 mil fotos são adicionadas ao sistema (NOYES, 2018). É visto que sistemas desse tipo necessitam que a DHT utilizada seja eficiente, sendo estes os cenários mais confiáveis para testes no âmbito de eficiência.

Com estes ambientes nos é permitido calcular o tempo para inserção e busca de arquivos em redes de diferentes tamanhos, permitindo assim testar a mesma DHT em situações variantes. Obtendo então uma visão ampla de seu desempenho, vantagens e desvantagens. Ao todo foram construídos 6 cenários de teste, sendo estes testados em redes de 8 até 32768 participantes em potência de 2 (hipercubos de 2^3 até 2^{15} nodos) e os tempos calculados foram medidos em unidades de tempo, sem um formato específico, sendo os cenários mostrados abaixo:

- Cenário 1: 900 mil *puts* sem a ocorrência de eventos de entrada ou saída.
- Cenário 2: 3 milhões de *lookups* sem a ocorrência de eventos de entrada ou saída.
- Cenário 3: 900 mil *puts* com um evento de saída no tempo 1.
- Cenário 4: 3 milhões de *lookups* com um evento de saída no tempo 1.
- Cenário 5: 900 mil *puts* com eventos de entrada ou saída totalmente aleatórios a cada 10 unidades de tempo e inicialmente metade dos nodos falhos.

- Cenário 6: 3 milhões de *lookups* com eventos de entrada ou saída totalmente aleatórios a cada 10 unidades de tempo e inicialmente metade dos nodos falhos.

3.3 Metodologia

A DHT testada neste trabalho é o HyperDHT. Por conta disso, foi necessário implementá-lo no simulador PeerSim, para que então a avaliação de sua eficiência em redes *Big Data* se tornasse possível. Utilizando o simulador com a topologia ser testada, foram realizados as análises de desempenho nas operações de *put* e *lookup* de dados no contexto de *Big Data*.

Para a análise foram utilizados os dados de tempo e número de mensagens recebidos sobre os desempenho das funções de *put* e *lookup* de dados recebidos dos casos testes em todos os cenários. Comparando então estes dados com as duas DHTs testadas. Obtendo assim uma análise da eficiência do HyperDHT por meio da comparação com o Chord.

3.4 Implementação

A implementação do HyperDHT no PeerSim foi realizada com base em implementações já estabelecidas no simulador, implementações estas desenvolvidas utilizando o mecanismo baseado em eventos e disponíveis no site do projeto PeerSim. A implementação levada principalmente em consideração foi a do DHT Chord que além de ser uma das DHT que serviu de comparação para o HyperDHT, possui duas implementações diferentes no site do projeto PeerSim, facilitando assim o entendimento do que é necessário para a construção de uma DHT no simulador e o que é uma particularidade do desenvolvedor.

Para a construção da DHT foi necessário a criação dos controladores. Estes são responsáveis pela execução de todas as operações no programa. Para a criação dos mesmos foi necessário atribuir as classes de controle o termo *implements Control*. Isto faz com que a classe implemente a interface *Control* fornecida pelo simulador. Estas classes são necessárias pois o simulador chama para execução em seu arquivo de configuração apenas classes de controle, tornando-as necessárias para qualquer execução. Por conta disso, cada operação executada nos testes possui um controlador. Além das classes controladoras foi implementada a classe do protocolo, visto que esta define as características de cada nodo do sistema, classes desse tipo implementam a interface *EDProtocol*.

O HyperDHT, para ter conhecimento de todos os participantes do sistema utiliza um *timestamp*, que representa a informação de estado de todos os nodos do sistema (falho ou sem falha). Este *timestamp* é necessário pois um nodo não deve enviar mensagens ou executar operações em um nodo inativo, e em cada teste executado pelo HyperDHT o *timestamp* é atualizado. Um *timestamp* é um contador inicializado em 0 que recebe incrementos na ocorrência de eventos. Números pares indicam que o nodo está correto e números ímpares indicam que o nodo está falho.

O protocolo de teste, nomeado *Ping Pong*, realiza a troca de mensagens em segundo plano no HyperDHT. Este é necessário para a atualização de conhecimento de todos os participantes do sistema, onde o nodo manda mensagem para todos os nodos de um determinado *cluster*. No entanto, o nodo testador não continua a testar os demais nodos de um *cluster* ao encontrar primeiro com falha. Neste caso o cluster é dito bloqueado e o nodo testador procura obter informações sobre os nodos deste *cluster* ao testar outros *clusters* (KOPPE, 2013). A determinação de quais nodos devem ser testados é obtida pela função CIS, melhor detalhada no trabalho de (KOPPE, 2013). Todos os nodos do sistema realizam seus respectivos testes, de forma que, em caso de alterações no sistema, após certo tempo todos os nodos ficam cientes de toda e qualquer atualização. Nesta troca de mensagens os nodos se atualizam do estado dos nodos do sistema, quais saíram e quais voltaram ao sistema.

Para o Chord não foi necessário toda uma implementação como no caso do HyperDHT, visto que o Chord já estava implementado no simulador PeerSim. Para esta DHT foi necessária a realização de adequações no código para que os cálculos e cenários se tornassem viáveis.

A versão implementada do Chord já trabalha com os métodos padrões da DHT, métodos como *stabilize(), find_sucessor()* entre outros. A função *stabilize()* tem como objetivo atualizar os nodos sobre o estado dos outros nodos do sistema em sua lista de sucessores, esta função roda em segundo plano de forma parecida com o *Ping Pong* do HyperDHT, para que assim os nodos estejam sempre se atualizando. Nesta DHT cada nodo possui uma lista de sucessores. Esta lista ajuda a realizar de forma mais rápida operações onde o acesso a outro nodo é necessário, necessitando assim menos saltos para a realização da consulta. Outra característica do sistema é a *finger table* que contém informações sobre outros nodos do sistema.

As operações de *lookup* e *put* já encontram-se presentes na implementação onde foi neces-

sário então incluir apenas uma nova organização das classes usadas para a implementação da DHT, bem como a realização dos contadores de tempo. Foi necessário também a criação de um novo arquivo de configuração para o Chord permitindo assim seus testes no simulador.

3.5 Resultados e Discussões

Os tempos utilizados para a realização de cada operação foram medidos utilizando o método *CommonState.getIntTime()* disponibilizados pelo próprio PeerSim. Com isso obtemos o resultado em unidades de tempo, sem a necessidade de um padrão especifico para a medição dos tempos. Foram testadas nos seis cenários as DHTs HyperDHT e Chord, para que a análise de desempenho assim se tornasse possível. Para aumentar a precisão do teste foram testados cada cenário 10 vezes, e a escolha dos nodos deu-se se forma aleatória utilizando como função para gerar estes valores a classe Random do Java.

As operações de *lookup* e *put* após os testes em todos os cenários apresentaram médias iguais para cada DHT, permitindo assim que sua representação se dê pelo gráfico apresentado na Figura 3.1, que apresenta uma representação geral para a média dos custos de tempo para as operações. A escolha dos nodos que realizaram cada operação foi realizada de forma totalmente aleatória, de modo a garantir uma média de tempo gasto proporcional para todo o sistema.

Para o tempo das operações do Chord os valores médios sempre giram em torno de 5 pelo fato de que cada nodo realiza várias operações e este valor sempre aparece em torno deste valor. Neste caso a média sempre fica muito próximo a este valor. Isso se dá pelo fato de que cada nodo possui uma lista de sucessores, diminuindo assim consideravelmente o tempo necessário para a busca de outro nodo.

Com este gráfico vemos que o HyperDHT apresenta melhores tempos para a realização das operações de *lookup* e *put*, mostrando-se mais eficiente que o Chord na realização destas operações, onde vemos que o mesmo apresenta um tempo bem superior para a realização das mesmas.

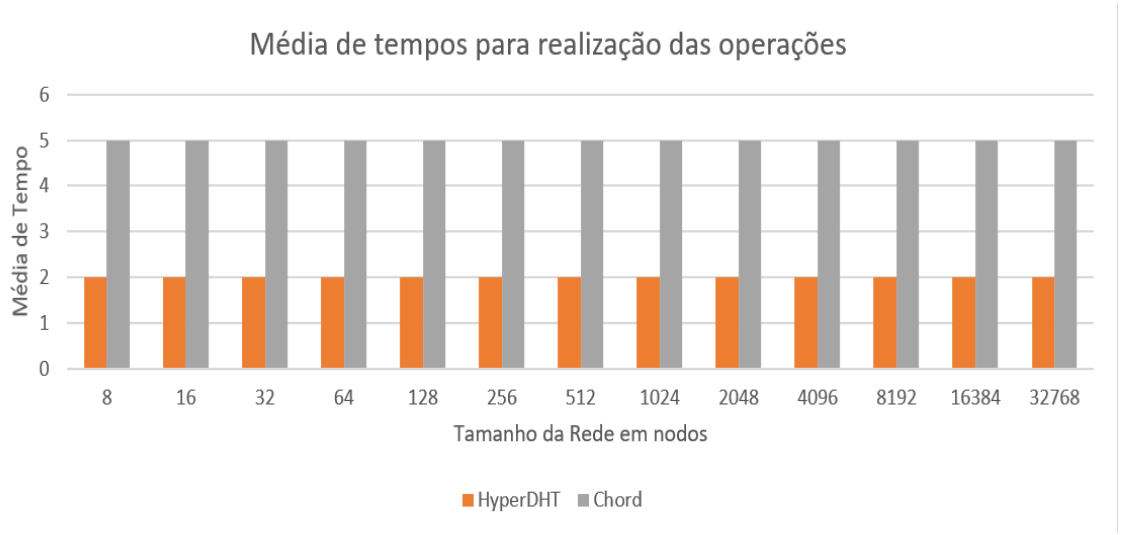

Figura 3.1: Gráfico de tempo médio para a realização das operações de *Lookup* e *Put* de dados. Fonte: O Autor

As Tabelas 3.1 e 3.2 apresentam o tempo total para a realização das operações de *put* e *lookup* em cada cenário. Nestas tabelas vemos que conforme a quantidade de nodos na rede aumenta o tempo total para a realização das operações diminui. Isso se dá pelo fato que conforme maior a rede, menos operações de *put* e *lookup* cada nodo deve fazer. As figuras e são a representação gráfica destes valores.

Tabela 3.1: HyperDHT - Tempo total para a realização das operações em cada cenário. Fonte: O Autor

	8	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
Cenário 1	1237510	618760	309380	154700	77360	38680	19340	9680	4850	2420	1100	660	260
Cenário 2	4125010	2062510	1031260	515630	257820	128920	64470	32240	16120	8070	2020	1320	600
Cenário 3	1237590	618800	309410	154730	77400	38720	19360	9900	5060	2640	1150	690	300
Cenário 4	4125230	2062560	1031300	515660	257870	129140	64680	32340	16280	8180	2200	1570	680
Cenário 5	1497100	748250	371370	187020	96670	48750	24760	14300	7670	3910	1980	1030	440
Cenário 6	6026800	2653360	1321290	650100	363030	171870	81700	46360	25330	12910	6520	3410	1260

Tabela 3.2: Chord - Tempo total para a realização das operações em cada cenário. Fonte: O Autor

	8	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
Cenário 1	6063790	3031920	1515960	758030	379060	189530	94760	47430	23760	11850	5390	3230	1270
Cenário 2	20212550	10106300	5053170	2526580	1263310	631700	315900	157970	78980	39540	9890	6460	2940
Cenário 3	6064190	3032120	1516100	758170	379260	189720	94860	48510	24790	12930	5630	3380	1470
Cenário 4	20213630	10106540	5053370	2526730	1263560	632780	316930	158460	79770	40080	10780	7690	3330
Cenário 5	7335790	3666420	1819710	916390	473680	238870	121320	70000	37580	19150	9700	5040	2150
Cenário 6	29531320	13001460	6474320	3185490	1778840	842160	400330	227160	124110	63250	31940	16700	6170

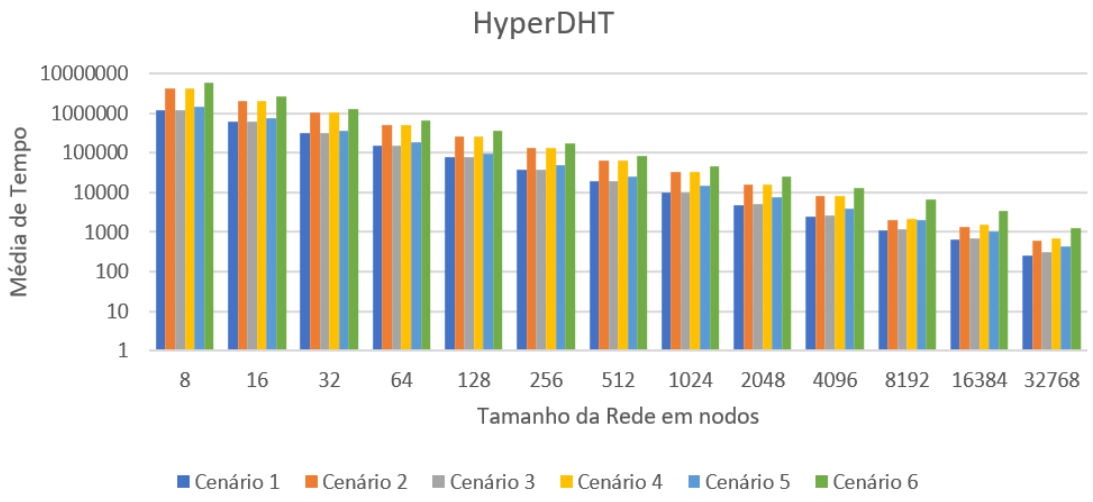

Figura 3.2: Gráfico de tempo total para a realização das operações em cada cenário no HyperDHT. Fonte: O Autor

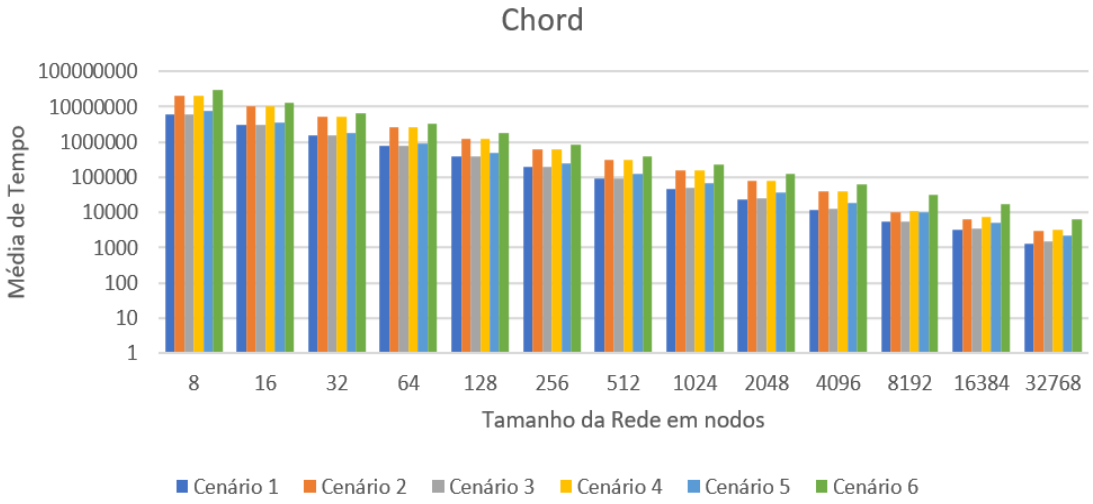

Figura 3.3: Gráfico de tempo total para a realização das operações em cada cenário no Chord. Fonte: O Autor

Os cenários 5 e 6 apresentam além dos tempos para a realização das operações o tempo médio para a reconfiguração da DHT após um evento. O cálculo para o tempo de reconfiguração da DHT após um evento inicia-se no momento em que o contato com o nodo é realizado pelo *Ping Pong* e o mesmo encontra-se indisponível, ou seja, seu valor no *timestamp* é um número ímpar, informando seu estado de falha. Neste momento é iniciada a contagem do tempo de

propagação de falhas levando em conta o valor deste nodo no *timestamp* do nodo que realizou o acesso. A finalização da contagem ocorre no momento em que todos os nodos da rede estão cientes do estado do nodo em questão. Nestes cenários ocorrem eventos de entrada e saída de nodos a cada 10 unidades de tempo, estes muito importantes para para a realização do cálculo do tempo necessário para a reconfiguração da DHT após um evento.

O cenário 5 trata-se da operação *put*, com o gráfico apresentado na Figura 3.4 vemos uma diferença considerável nos tempos gastos para a de reconfiguração da DHT após um evento conforme a quantidade de nodos na rede utilizados. Neste gráfico vemos que o HyperDHT mostra-se muito mais eficiente tratando disso. O Chord, que por conta de sua topologia em anel e regras da DHT necessitou de um tempo bem maior em relação ao HyperDHT, onde a diferença cresce consideravelmente a partir dos 64 nodos na rede, diferença essa que aumenta gradativamente conforme a evolução do tamanho da rede. A Tabela 3.3 representa em forma de tabela os mesmos valores, acrescentando a diferença de tempo entre cada DHT para a realização da operação.

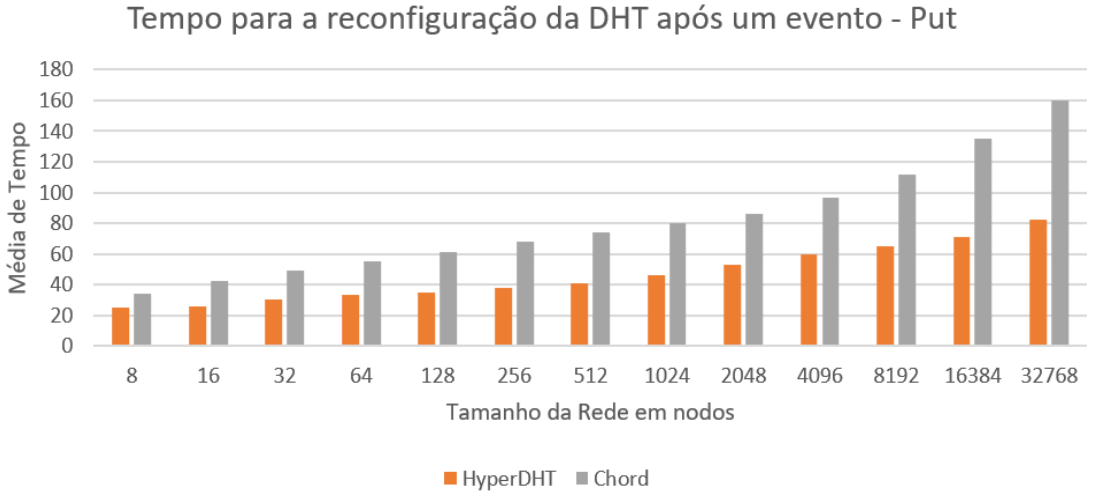

Figura 3.4: Gráfico de tempo de reconfiguração da DHT após um evento na operação de *Put*. Fonte: O Autor

Tabela 3.3: Tempo de reconfiguração da DHT após um evento na operação de *Put*. Fonte: O Autor

	8	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
HyperDHT	25	26	30	33	35	38	41	46	53	60	65	71	82
Chord	34	42	49	55	61	68	74	80	86	97	112	135	160
Diferença	26,4%	38,0%	38,7%	40,0%	42,6%	44,1%	44,5%	42,5%	38,3%	38,1%	41,9%	47,4%	48,7%

O cenário 6 trata-se da operação de *lookup*. Com o gráfico apresentado na Figura 3.5 vemos que os tempos gastos nesta operação se alteram de forma semelhante ao cenário 5. Isso se dá pelo fato que o fator diferencial para o acréscimo de tempo para a reconfiguração da DHT após um evento é a quantidade de nodos na rede e não a operação utilizada.

Neste gráfico novamente o HyperDHT mostra-se mais eficiente que o Chord. Neste caso, a diferença de tempo necessário para a reconfiguração da DHT fica bastante evidente novamente a partir dos 64 nodos na rede, onde o Chord necessita de muito mais tempo para a propagação das falhas. A Tabela 3.4 representa em forma de tabela os mesmos valores, acrescentando a diferença de tempo entre cada DHT para a realização da operação.

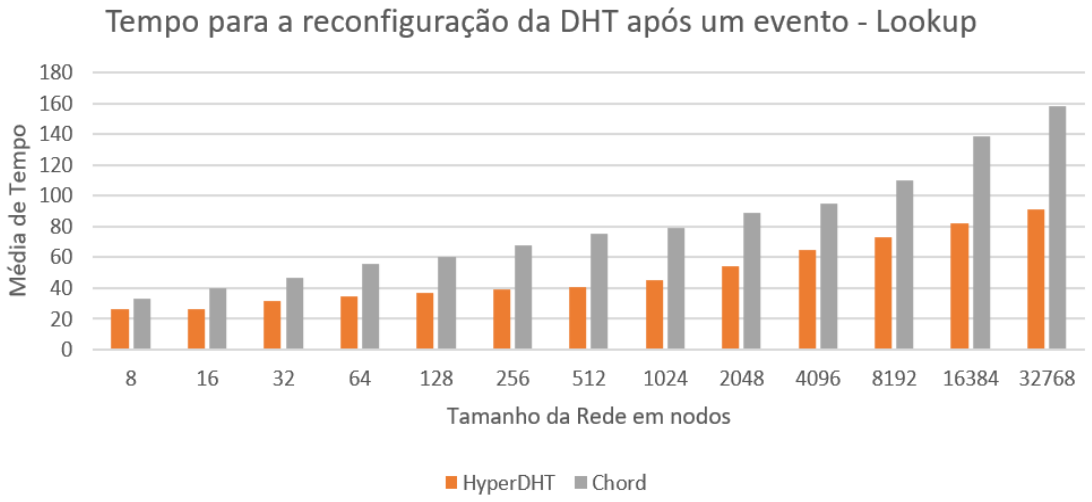

Figura 3.5: Gráfico de tempo de reconfiguração da DHT após um evento na operação de *Lookup*. Fonte: O Autor

Tabela 3.4: Tempo de reconfiguração da DHT após um evento na operação de *Lookup*. Fonte: O Autor

	8	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
HyperDHT	26	26	32	35	37	39	41	45	54	65	73	82	91
Chord	33	40	47	56	60	68	75	79	89	95	110	139	158
Diferença	21,2%	35,0%	31,9%	37,5%	38,3%	42,6%	45,3%	43,0%	39,3%	31,5%	33,6%	41,0%	42,4%

Com os gráficos apresentados concluímos que a DHT mais eficiente no quesito de reconfiguração da DHT após um evento é o HyperDHT e a mais custosa é o Chord. Para a realização das operações de *lookup* e *put* o HyperDHT novamente se sai superior que o Chord, onde o mesmo é consideravelmente mais custoso para a realização destas operações.

Com base nestes resultados podemos afirmar que o uso do HyperDHT no contexto de *Big Data* pode ser muito bem visto, levando em conta que o mesmo foi muito superior em todos os testes ao Chord, este último sendo uma DHT já estabelecida para seu uso neste contexto.

Capítulo 4

Conclusões e Trabalhos Futuros

DHTs são estruturas de dados que utilizando o conceito das tabelas *hash* usuais, compostas por pares do tipo <chave, valor> armazenam dados distribuídos em vários nodos do sistema, originalmente utilizadas em redes P2P as DHTs podem construir sistemas distribuídos garantindo a descentralização, escalabilidade e tolerância a falhas. As DHTs mostram-se uma boa chamada ao serem utilizadas no contexto do *Big Data*, visto que as mesmas são capazes de gerenciar grandes quantias de dados com certa eficiência.

A principal contribuição deste trabalho dá-se em analisar o desempenho do HyperDHT para as operações de *lookup* e *put* de dados no contexto *Big Data*. Esta análise sendo realizada por meio da comparação do desempenho do HyperDHT com outra DHT, Chord, esta já bem estabelecida neste caminho.

Para a realização dos testes foram analisados o desempenho das DHTs em 6 diferentes cenários cada qual testado em redes de 8 até 32768 participantes em potência de 2 (hipercubos de 2^3 até 2^{15} nodos). Os testes foram realizados no simulador *open source* para sistemas P2P PeerSim.

Com base nos resultados obtidos pode-se concluir que o HyperDHT mostra-se eficiente para uso em *Big Data* levando em conta que o mesmo mostrou-se mais eficiente que a DHT Chord já utilizada neste meio.

O HyperDHT para a realização da reconfiguração da DHT após um evento é superior ao Chord no tempo para a realização do mesmo. Para as operações de *lookup* e *put* a diferença é ainda maior, onde é visto que o custo para a realização das operações é menor que a metade gasta pelo Chord.

Neste trabalho vemos a eficiência do HyperDHT para as operações de *lookup* e *put* no con-

texto *Big Data*. Dessa forma, propõe-se como trabalho futuro a análise do HyperDHT para a replicação de dados no contexto *Big Data*, de modo a garantir a compatibilidade da DHT para seu uso real em *Big Data*, visto que apenas a *put* e *lookup* não são suficientes para a implementação da DHT em um ambiente real. Além disso, propõe-se a implementação e execução do HyperDHT em um ambiente real.

Referências Bibliográficas

BONA, L. C. et al. Hyperbone: Uma rede overlay baseada em hipercubo virtual sobre a internet. 01 1996.

COULOURIS, G. et al. *Distributed Systems: Concepts and Design*. 5th. ed. USA: Addison-Wesley Publishing Company, 2011. ISBN 0132143011, 9780132143011.

DUARTE, E. P.; NANYA, T. A hierarchical adaptive distributed system-level diagnosis algorithm. *IEEE Transactions on Computers*, v. 47, n. 1, p. 34–45, Jan 1998. ISSN 0018-9340.

GROVER, P.; JOHARI, R. Bcd: Bigdata, cloud computing and distributed computing. In: *2015 Global Conference on Communication Technologies (GCCT)*. [S.l.: s.n.], 2015. p. 772–776.

HADZILACOS, V.; TOUEG, S. *A Modular Approach to Fault-Tolerant Broadcasts and Related Problems*. Ithaca, NY, USA, 1994.

KOPPE, J. P. *HyperDHT - DHT de um salto baseada em hipercubo virtual distribuído*. Dissertação (Dissertação de Mestrado) — UFPR – Universidade Federal do Paraná, Curitiba - PR, Agosto 2013.

KRULL, J. W.; WU, J.; MOLINA, A. M. Evaluation of a fault tolerant distributed broadcast algorithm in hypercube multicomputers. In: *Proceedings of the 1992 ACM Annual Conference on Communications*. New York, NY, USA: ACM, 1992. (CSC '92), p. 459–466. ISBN 0-89791-472-4. Disponível em: <http://doi.acm.org/10.1145/131214.131272>.

LI, B.; LIAO, X. Peer-to-peer in big data management. *Peer-to-Peer Networking and Applications*, v. 6, n. 4, p. 361–362, Dec 2013. ISSN 1936-6450. Disponível em: <https://doi.org/10.1007/s12083-013-0243-1>.

MONNERAT, L. R. R. *Tabelas Hash Distribuídas com consultas de um salto com baixa carga de manutenção*. Tese (Tese de Doutorado) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Setembro 2010.

MONTRESOR, A.; JELASITY, M. PeerSim: A scalable P2P simulator. In: *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*. Seattle, WA: [s.n.], 2009. p. 99–100.

NETO, A. B. *VCubeDHT- Uma DHT de Salto Único Baseada em um Hipercubo Virtual com Replicação*. Dissertação (Trabalho de Conclusão de Curso) — UNIOESTE – Universidade Estadual do Oeste do Paraná, Cascavel - PR, Agosto 2016.

NETO, A. B.; RODRIGUES, L. A.; DUARTE, E. P. J. Vcubedht - uma dht de salto Único baseada em um hipercubo virtual com replicação. In: *Anais do Workshop de Testes e Tolerância a Falhas (SBRC-WTF)*. [S.l.]: SBC, 2017. (SBRC-WTF 2017), p. 42–55.

NOYES, D. *The Top 20 Valuable Facebook Statistics ? Updated July 2018*. 2018. Consultado na INTERNET: https://zephoria.com/top-15-valuable-facebook-statistics/, 2018.

RATNASAMY, S. et al. A scalable content-addressable network. In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM, 2001. (SIGCOMM '01), p. 161–172. ISBN 1-58113-411-8. Disponível em: <http://doi.acm.org/10.1145/383059.383072>.

RODRIGUES, L. A. *Uma solução autonnômica para k-exclusão mútua em sistemas distribuídos*. Tese (Tese de Doutorado) — Universidade Federal do Paraná, Curitiba, PR, Agosto 2014.

RUOSO, V. K. *Uma estratégia de testes logarítmica para o algoritmo HI-ADSD*. Dissertação (Dissertação de Mestrado) — UFPR – Universidade Federal do Paraná, Curitiba - PR, Agosto 2013.

SHARMA, S.; MANGAT, V. Technology and trends to handle big data: Survey. In: *2015 Fifth International Conference on Advanced Computing Communication Technologies*. [S.l.: s.n.], 2015. p. 266–271. ISSN 2327-0632.

STOICA, I. et al. Chord: A scalable peer-to-peer lookup service for internet applications. In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM, 2001. (SIGCOMM '01), p. 149–160. ISBN 1-58113-411-8. Disponível em: <http://doi.acm.org/10.1145/383059.383071>.

TANENBAUM, A. S.; STEEN, M. v. *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 0132392275.

WANG, X.; YIN, Y. L.; YU, H. Finding collisions in the full sha-1. In: *Proceedings of the 25th Annual International Conference on Advances in Cryptology*. Berlin, Heidelberg: Springer-Verlag, 2005. (CRYPTO'05), p. 17–36. ISBN 3-540-28114-2, 978-3-540-28114-6. Disponível em: <http://dx.doi.org/10.1007/11535218_2>.

ZHU, H. Big data and collaboration research: Creating real big data. *IEEE Systems, Man, and Cybernetics Magazine*, v. 4, n. 1, p. 16–19, Jan 2018.