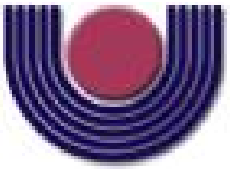

Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Uma análise do VCubeDHT para Armazenamento e Recuperação de Grandes Volumes de Dados

Elixandre Michael Baldi

CASCAVEL
2019

ELIXANDRE MICHAEL BALDI

Uma análise do VCubeDHT para Armazenamento e Recuperação de Grandes Volumes de Dados

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
2019

Elixandre Michael Baldi

Uma análise do VCubeDHT para Armazenamento e Recuperação de Grandes Volumes de Dados

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. Edson Tavares de Camargo
Colegiado de Ciência da Computação, UTFPR -
Toledo

Cascavel, 11 de dezembro de 2019

DEDICATÓRIA

Dedico este trabalho para meu pai, Maninho (in memorian)

AGRADECIMENTOS

A meu pai Alexandre Micael Baldi *(in memorian)*, a minha mãe Marisa Aparecida Amaral Baldi, e a meu irmão Eliseu Baldi Neto que estiveram ao meu lado nas horas mais difíceis e felizes da minha vida.

A meu orientador e mestre Prof. Dr. Luiz Antonio Rodrigues pela dedicação, compreensão, paciência e amizade.

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 Report Algorithm*
Hi-ADSD	*Hierarchical Adaptive Distributed System-level Diagnosis*
IP	*Internet Protocol*
P2P	*Peer to Peer*
RS	Revisão Sistemática
TTL	*Time-to-Live*

Lista de Símbolos

$C_{i,s}$	Função responsável por criar o hipercubo
N	Número de nodos pertencentes ao sistema
p	Participante qualquer do sistema
q	Participante qualquer do sistema
r	Participante qualquer do sistema

Sumário

Resumo

Um dos problemas presentes nos sistemas distribuídos com grande quantidade de recursos e nodos, e a localização dos mesmos por parte de um determinado participante. As DHTs (*Distributed Hash Table*) apresentam uma solução para este problema organizando os recursos baseado no conceito de *chave*/*valor*, ou seja, para cada recurso de valor *v* existe uma chave *k* correspondente. Dessa forma as DHTs contam com a função *lookup* para encontrar o responsável por uma determinada chave, baseada na visão local do nodo executante da função. As DHTs de múltiplos saltos permitem que cada nodo mantenha uma visão parcial do sistema, em que o *lookup* necessita de possíveis consultas nas tabelas *hashs* de outros participantes para a conclusão da busca. As DHTs de salto único apresentam a alternativa de evitar as consultas adicionais de forma que cada nodo mantém uma visão total do sistema. Para manter as tabelas *hashs* atualizadas, algoritmos de diagnóstico distribuídos são executadas nos sistemas com DHTs. Em DHTs de múltiplos saltos destacam-se o algoritmo Chord e Kademlia, nas de salto único, destacam-se o VCube, OneHope, D1HT e HyperDHT. O sistema VCubeDHT é um modelo de sistema distribuído que utiliza as DHTs de salto único para a organização das chaves e o algoritmo VCube para o diagnóstico dos nodos falhos. O presente trabalho apresenta melhorias nos protocolos do sistema VCubeDHT para sistemas com grande quantidade de recursos e participantes com foco no armazenamento e recuperação de recursos e entrada e saída de participantes na rede. Outra contribuição do trabalho foi o desenvolvimento de uma revisão sistemática sobre o assunto, que mostrou que as DHTs de múltiplos saltos são mais abordadas em trabalhos do que as DHTs de salto único. O VCubeDHT apresenta maior eficiência em todos os aspectos testados e comparados com uma DHT que utiliza o Chord como algoritmo de diagnóstico distribuído. Dentre os aspectos, destacam-se o tempo de diagnóstico de falha; duração do *lookup*; quantidade de saltos para realizar o *lookup*; duração de um processo de entrada de participantes na rede.

Palavras-chave: DHT, sistemas distribuídos, VCube, recuperação de dados.

Capítulo 1

Introdução

De acordo com Sagiroglu e Sinanc (2013), cinco *exabytes* (10^{18} bytes) de dados haviam sido gerados por seres humanos até 2003. Este valor passou a ser gerado a cada dois dias em 2012, alcançando cerca de 2,72 *zettabytes* (10^{21} bytes). Os autores ainda estimam que até 2020, com o avanço da Internet das Coisas (IoT), milhões de novos equipamentos serão conectados em redes de computadores, gerando grande quantidades de dados diariamente. A grande questão é: como extrair informação relevante e em tempo hábil de um conjunto de dados tão grande e em constante incremento? A resposta está ligada ao termo *big data*.

O termo *big data* baseia-se em seis características, das quais estão presentes em qualquer aplicação que envolva um conjunto muito grande ou complexo de dados, são elas: grande *volume* de dados; *variedade* de estruturas de armazenamento resultando em uma *complexa* comunicação entre estruturas diferentes; *velocidade* de geração de dados; *variabilidade* no fluxo de dados, em que um único recurso pode ser requisitado por diversos dispositivos ao mesmo tempo; dados que representam *valor* crítico ao usuário e que não podem ser perdidos (LANEY, 2001), (KATAL; WAZID; GOUDAR, 2013).

Para garantir o bom funcionamento de um sistema distribuído considerando as características de um cenário de *big data*, diversas técnicas são sugeridas na literatura, tais como P2P (*peer to peer*) (AMIR; SRINIVASAN; KHAN, 2015) e computação em nuvem (ZHANG et al., 2012). Seguindo o contexto da quarta revolução industrial (HERMANN; PENTEK; OTTO, 2016), em que um dos pilares diz respeito a aplicações que tomam decisões descentralizadas baseadas em informações localizadas em diversos locais de uma rede, este trabalho foca numa solução no contexto de sistemas distribuídos.

Coulouris, Dollimore e Kindberg (2007) descrevem um sistema distribuído como um con-

junto de componentes se comunicam e coordenam suas ações por meio de mensagens, de forma que cada componente pode prover um determinado recurso ou informação para os outros. Tanenbaum e Steen (2007) preferem definir um sistema distribuído como uma coleção de computadores independentes que se apresenta ao usuário na interface de apenas um dispositivo. Seguindo a segunda definição é possível armazenar e dados e distribuídos recursos em diversos dispositivos conectados por meio de uma rede, e acessar os mesmos por apenas um *hardware* participante deste sistema. Uma das formas para armazenar as informações, distribuir os recursos e localizar dados na rede é utilizar as chamadas DHTs (*Distributed Hash Table*), que organizam os componentes seguindo uma ordenação lógica em que cada nodo (participante da rede) fica responsável por um espaço de chaves relacionadas aos recursos. Dessa forma para cada recurso é armazenado uma tupla <*chave, valor*> e as pesquisas por estes recursos tem como parâmetro somente a chave em que o protocolo utilizado pelo sistema encarregará de localizar (KUROSE; ROSS, 2010).

Cada participante do sistema DHT armazena uma tabela de roteamento que relaciona os outros nodos com seus respectivos espaço de chaves. Esta tabela pode armazenar apenas alguns nodos, ou seja, aquele nodo tem visão parcial do sistema e consequentemente uma pesquisa necessitará de consultas em outros participantes, aumentando assim a visão do sistema e então localizando o responsável pela chave. Assim sendo, a tabela de roteamento pode armazenar informações sobre todos os indivíduos do sistema tendo assim uma visão global do sistema, de forma que uma consulta não necessita de consultas adicionais em outros nodos.

O termo *hop* que do inglês significa "salto" é empregado para cada consulta de um nodo a outro. Os sistemas DHTs em que os participantes tem visão parcial do sistema são chamado de *Multi Hop DHT*, pois necessitam de múltiplos saltos para um pesquisa. Os sistemas DHTs que mantêm uma tabela de roteamento com visão global do sistema são chamados de *Single Hop*, pois as pesquisas são realizadas com apenas um salto.

Visto que não tem como garantir que os componentes de um sistema distribuídos sempre estarão ativos e que as consultas de um sistema DHT seja realizadas com eficiência, é necessário que as tabelas de roteamento mantenham-se o mais atualizadas possível. Para isto, é essencial detectar eventuais falhas e disseminar esta informação para todo o sistema o quanto antes. Esta tarefa é responsabilidade de um algoritmo de diagnóstico distribuído, que por sua vez realiza tes-

tes de falhas em todos os participantes da rede, caso encontre um falho, reporta para os demais. O VCube (DUARTE Jr.; BONA; RUOSO, 2014) é um algoritmo de diagnóstico distribuído que organiza os nodos com base em uma topologia hierárquica de hipercubo virtual, garantindo diversas propriedades logarítmicas, como distância máxima entre nodos e quantidade máxima de testes por rodada.

Koppe (2013) propôs uma solução DHT *Single Hop* utilizando o algoritmo *DiVHA* (*Distributed Virtual Hypercube Algorithm*) (BONA et al., 1996), uma versão anterior ao *VCube*, como uma alternativa as DHTs existentes, como, por exemplo, o *Chord* (STOICA et al., 2001) e o *OneHop DHT* (GUPTA; LISKOV; RODRIGUES, 2004), que organizam os nodos do sistema em um anel virtual, e o *Can* (RATNASAMY et al., 2001) que utiliza uma arquitetura baseada num espaço cartesiano multidimensional de coordenadas.

O algoritmo DiVHA, assim como o VCube, organiza os nodos em um hipercubo virtual, mas diferentemente do VCube as arestas deste hipercubo são armazenadas em um grafo, em que cada evento (detecção de uma falha ou entrada de um participante), arestas são geradas ou eliminadas. O VCube, por sua vez, abstrai o grafo numa regra matemática envolvendo o nodo que irá realizar o teste, o possível nodo a ser testado, e o nível hierárquico em que ambos partilham na topologia da rede. Com esta regra, o VCube garante como número de testes máximo $Nlog2N$, e latência máxima de log_2^2N rodadas de testes para o algoritmo determinar o *status* de todos os nodos da rede, além de economizar recursos, de forma que não é necessário manter grafos, realizar operações em grafos, como por exemplo busca de menor caminho entre dois nodos, e gerar novas arestas (RUOSO, 2013).

Diante do apresentado, este trabalho tem como objetivo geral formalizar protocolos com ênfase no armazenamento e recuperação de grandes volumes de dados em sistemas distribuídos que utilizam DHTs de forma a basear-se no VCube. Em especial os protocolos serão aplicados na solução VCubeDHT, proposto por Barreiro Neto, Rodrigues e Duarte Jr. (2017). Além disso, sumarizou-se alguns objetivos específicos:

- A partir de uma revisão sistemática sobre o uso de DHT no contexto de *Big Data*, buscar soluções DHT na literatura e, se possível, relacioná-las no contexto de *big data*;
- Avaliar o uso do VCube como rede de sobreposição para o uso em DHTs;

- Complementar os protocolos do VCubeDHT (BARREIRO NETO; RODRIGUES; DUARTE Jr., 2017);

- Realizar testes em diversos cenários, comparando o VCubeDHT com outras soluções, a fim de averiguar as hipóteses apresentadas na introdução.

O primeiro objetivo específico foi motivado pela obra de Khan et al. (2003), em que afirma que é possível identificar obras relevantes a um determinado tema de uma pesquisa de forma a evitar a não consideração de eventuais textos relevantes presentes numa base de textos científicos, aplicando uma revisão sistemática. Para isto segue-se uma sequência de passos sistemáticos de acordo com um protocolo.

Com a obtenção do conjunto de obras relacionadas ao tema a partir da revisão sistemática, é possível averiguar a eficiência do uso de DHT em ambientes distribuídos com grande quantidade de dados e então realizar testes em ambiente simulado a fim de comparar a eficiência do VCube e suas propriedades logarítmicas com as soluções presentes. Os cenários seguirão os *benchmarks* presentes em (SMITH, 2017) e (SMITH, 2018).

O restante do trabalho organiza-se da seguinte forma: O Capítulo 2 apresenta a fundamentação teórica, detalhando os conceitos comentados e outros relacionados a tolerância a falhas e sistemas distribuídos. O Capítulo 3 descreve os passos e a execução da revisão sistemática. O Capítulos 4 apresenta o VCubeDHT e seus protocolos. O Capítulo 5 realiza descreve quatro cenários para realização de testes juntamente com discussões sobre cada cenário. Por fim, o Capítulo 6 apresenta as conclusões, considerações finais e trabalhos futuros sugeridos.

Capítulo 2

Referencial Teórico

Em toda a história do mundo, a humanidade passou por três mudanças que transformam a forma com que o homem vive. No século XVIII a primeira revolução industrial apresentou ao mundo as máquinas a vapor possibilitando assim as primeiras produções em larga escala. No fim do século XIX o uso da eletricidade e as novas organizações de trabalho (Fordismo e Taylorismo) passou a ser conhecido como a segunda revolução industrial. A terceira revolução industrial, no fim do século XX introduziu nos meios de produção a eletrônica avançada e a tecnologia da informação, conhecido assim como "A revolução digital" (HERMANN; PENTEK; OTTO, 2016).

Em 2011, uma iniciativa chamada *Indústria 4.0* ficou informalmente conhecida como a quarta revolução industrial (HERMANN; PENTEK; OTTO, 2016). Esta iniciativa reuniu representantes de empresas, da política e das universidades na Alemanha. Não cabe para este trabalho entrar muito em detalhes, mas o objetivo desta iniciativa é fornecer "melhorias nos processos industriais envolvidos na fabricação, engenharia, uso de material e cadeia de fornecimento e gestão do ciclo de vida" (KAGERMANN; WAHLSTER; HELBIG, 2013).

Segundo Hermann, Pentek e Otto (2016) os pilares da Indústria 4.0 são: interconexão, transparência da informação, decisões descentralizadas e assistência técnica. O primeiro pilar diz respeito a dispositivos conectados através da IoT (*Internet of Things*), em que principalmente tecnologias sem fio realizam comunicação entre máquinas, dispositivos, sensores e pessoas, possibilitando assim fábricas inteligentes adaptáveis as exigências que variam muito rapidamente no mercado. O segundo pilar diz respeito a possibilidade de informações sensíveis ao contexto serem capturadas e estejam disponíveis para pessoas e dispositivos a fim de influenciar em decisões em tempo real. O terceiro pilar baseia-se nos dois primeiros em que é possível

utilizar informações locais cruzando-as com informações globais (recuperadas de outros dispositivos ou processos) melhorando assim a tomada de decisões e aumento da produtividade geral. O último pilar descrito discute a complexidade dos processos distribuídos e consequentemente a dificuldade de resolver problemas decentralizados. Dessa forma é preciso ferramentas que auxiliem o ser humano a tomar as melhores decisões para evitar ou corrigir problemas no cenário distribuído.

Dado o contexto da quarta revolução industrial, entende-se que a humanidade passa por diversas mudanças nos meios de produção e organização. Além disso observa-se que as demandas humanas tendem a dependência de grande quantidade de informação. Neste sentido a organização Reinsel, Gantz e Rydning (2018) apresenta algumas previsões para o ano de 2025.

A Figura 2.1 apresenta onde os dados são armazenados atualmente e onde serão armazenados no futuro. Observe que atualmente os dados são armazenados principalmente em redes corporativas e de entretenimento e que até 2025, as redes corporativas terão uma quantidade mais significativa. Isto mostra uma grande criticidade dos dados, visto que dados corporativos estão relacionados aos serviços que os mesmos exercem sendo assim informações de risco.

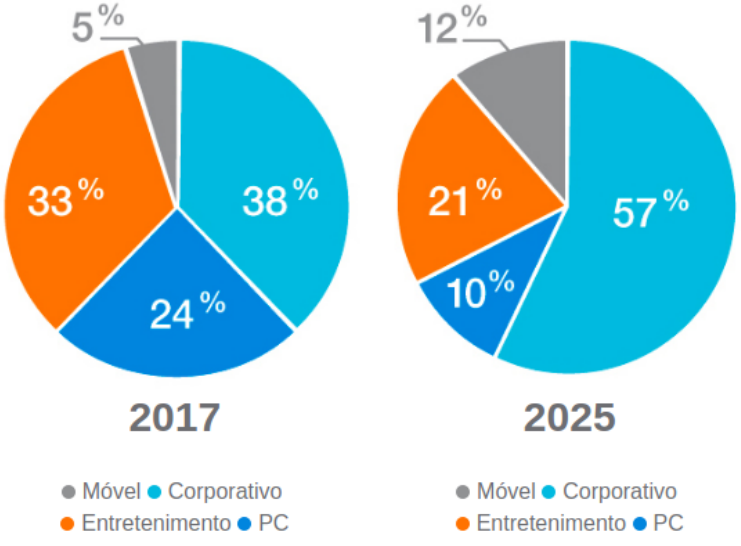

Figura 2.1: Onde os dados são/serão armazenados (REINSEL; GANTZ; RYDNING, 2018).

Além disso, Reinsel, Gantz e Rydning (2018) prevê que a taxa média per capita das iterações com dados por dia aumentará vinte vezes nos próximos vite anos, de forma que atualmente esta taxa está perto das 600 iterações e em 2025 chegará aos 4800 iterações. Este valor esta

principalmente ligado ao desenvolvimento de tecnologias de *IoT* presentes em casas, escritórios, veículos, implantes, móveis, dentre outros. Outra informação interessante que Reinsel, Gantz e Rydning (2018) apresenta é que usuários conectados a internet, devem somar 75% da população mundial, de forma que 95% das aplicações estarão no contexto móvel e de tempo real. Por último, aplicações do contexto de aprendizagem de máquina, em que dependem de milhões de bytes de informações para serem treinadas e posteriormente realizar suas atividades de forma mais eficiente, deverão ter um valor de mercado de quarenta bilhões de dólares.

Diante destes dados Jeff Fochtman, presidente da Seagate Technology, o grupo que patrocinou o desenvolvimento das informações apresentados anteriormente, em entrevista a Cave (2018) levantou alguns pontos: primeiramente, são armazenados menos de 1% dos dados que são criados, o que continuará a diminuir, pois a quantidade de armazenamento enviado não tem capacidade de acompanhar a quantidade de dados criados. Além disso, existe perguntas abertas sobre como administrar tantos dados. Estas perguntas e preocupações são relacionadas ao termo *big data*.

Na continuidade deste capítulo o termo *big data* será detalhado e posteriormente a discussão fluirá para cenário de sistemas distribuído bem como o impacto que grande quantidade de informação causa num sistema global.

2.1 *Big Data*

Laney (2001) elegeu as palavras *volume*, *variedade* e *velocidade* como as três dimensões dos desafios no gerenciamento de dados, chamando-os de "os Três Vs":

- **Volume:** grande quantidade de dados disponíveis, seja esse dado uma transação bancária, que ocorrem inúmeras por dia, ou informações obtidas por um sensor a cada segundo, ou ainda, trazendo para um contexto atual, o armazenamento de cada curtida que o *Facebook* registra diariamente.

- **Variedade:** Diz respeito à diferença de estruturas de armazenamento dos dados, como o registro de uma curtida de uma determinada publicação é feita de uma forma diferente das informações capturadas por um sensor, de forma que uma determinada aplicação dependa da união de ambos os registros.

- **Velocidade:** Diz respeito à velocidade que os dados são gerados. O autor dá o exemplo das transações bancárias feitas ao mesmo tempo ao redor do mundo, e a quantidade de requisições de um aplicativo para um banco de dados. Segundo Wiener e Bronson (2014), o *Facebook* gera cerca de 4 petabytes de dados e executa 600 mil consultas em 800 mil tabelas distribuídas. Para cada execução (armazenamento ou recuperação de dados) é necessário que a aplicação processe o dado e então realiza a execução, o que pode levar um alto tempo para ser concluído.

Posteriormente autores fizeram referência aos “Três” Vs para descrever o termo big data: Klein e Gorton (2015) destacam os sistemas heterogêneos (SQL, NoSQL e NewSQL) fazendo referência a “variedade”, sistemas escaláveis de sistemas distribuídos que armazenam dados em centenas ou milhares de computadores referenciando "volume", e as requisições de escrita e leitura de dados que são executados em paralelo, podendo gerar latência de resposta que diz respeito a "velocidade"; Katal, Wazid e Goudar (2013) além de descrever os “Três Vs” acrescentam as palavras *complexidade*, *variabilidade* e *valor* para definir *big data*:

- **Complexidade:** Diz respeito a complexidade da comunicação de sistemas provenientes de diferentes fontes com diferentes topologias.

- **Variabilidade:** Diz respeito a inconsistências no fluxo de dados, e a dificuldade de serem mantidas, principalmente em aplicações com muitos acessos simultâneos, causando carga de dados superior ao limite do hardware do sistema.

- **Valor:** Diz respeito aos dados armazenados terem algum valor para o usuário. O autor exemplifica uma empresa que requisita um relatório crítico e que irá influenciar alguma decisão de risco.

Diante das definições de *big data* apresentadas, neste trabalho o termo será relacionado aos “Três Vs” de Laney (2001) juntamente com as palavras apresentado por Katal, Wazid e Goudar (2013).

2.2 Sistemas Distribuídos

Em Coulouris, Dollimore e Kindberg (2007), um sistema distribuído é definido como um sistema com componentes localizados em computadores que fazem parte de uma mesma rede e

que comunicam-se e coordenam suas ações por meio de mensagens. Dessa forma, um sistema distribuído apresenta três características: concorrência de componentes, falta de um relógio global e falhas de componentes independentes. O presente trabalho foca nos desafios gerados a partir da última característica.

Quando um dispositivo falha, o resto da rede continua funcionando, mas um trecho da rede que dependa daquele dispositivo pode ficar isolado. É responsabilidade do projetista do sistema definir estratégias para driblar as consequências causadas por uma falha (COULOURIS; DOLLIMORE; KINDBERG, 2007). Neste caso, as estratégias descritas acrescentam ao sistema a característica de ser tolerante a falhas.

Existem dois atores principais nos cenários com sistemas distribuídos. O primeiro ator é o *cliente*, que sempre requisita um dado ou recurso para o *servidor*, que por sua vez fornece o que o *cliente* requisita. É possível que um mesmo dispositivo seja cliente ou servidor, o que será descrito a seguir, mas é importante deixar claro que num único processo, o dispositivo assume apenas um papel. Dependendo da organização do *cliente* e do *servidor*, o sistema assumirá um dos seguintes modelos: i) modelo cliente, servidor; ii) modelo P2P (*peer to peer*). iii) modelo híbrido. A seguir uma breve descrição de cada modelo baseado em Coulouris, Dollimore e Kindberg (2007).

- Cliente Servidor: Considerando um sistema de proposito geral, o sistema é organizado com inúmeros participantes, em que os recursos são concentrados nos *servidores* que tem papel apenas de fornecer recursos. Os processos de uma determinada aplicação, por exemplo, é responsabilidade dos *clientes*, que caso precisem de algum recurso, requisitam aos servidores presentes. A Figura 2.2 ilustra esta arquitetura.

  As setas representam as possíveis requisições de recurso na rede que vão dos participantes (clientes) para o servidor que neste modelo esta centralizado e armazenando todos os recursos. Como vantagem apresentado por Kurose e Ross (2010), os servidores mantém endereços fixos e conhecidos de forma que os clientes não gastam processamento para localizar um determinado recurso. Como desvantagem Kurose e Ross (2010) diz que é preciso garantir que o servidor estará livre de falhas para evitar comprometimento da rede, o que não é possível num cenário real. Desta forma o sistema fica sujeito a possíveis falhas em seu funcionamento. Outra desvantagem apresentada é que muitas vezes um

Figura 2.2: Modelo Cliente Servidor

único servidor é incapaz de atender requisições de inúmeros clientes ao mesmo tempo.

- *P2P*: É comum encontrar sistemas de compartilhamento de arquivos que utilizam esta arquitetura. Os recursos são distribuídos em todos os *peers* (participantes), evitando a centralização de recursos. Conforme algum cliente precise de algum recurso, a requisição será feita a outro *peer*, que neste processo assumirá papel de servidor. A forma como os recursos serão distribuídos, armazenados e localizados, depende da política que o sistema adotar. O objetivo deste modelo é "permitir o compartilhamento de dados e recursos em uma escala muito grande", eliminando a exigência de gerenciadores centralizadores. A principal vantagem da decentralização é que caso ocorra uma falha num dispositivo o sistema continua funcionando com recursos de dispositivos que não estão falhos. Além disso Kurose e Ross (2010) destaca a capacidade de ser escalável, visto que para cada novo participante, por mais que gere mais tráfego na rede, também aumenta a disponibilidade e capacidade de recursos da mesma. A Figura 2.3 apresenta o modelo P2P. Note que na figura, cada participantes mantém ligações para todos os outros, ou seja, é possível que um recurso esteja em vários nodos, caso um determinado nodo falhe, o requisitante pode realizar outras pesquisas pelo recurso.

  As aplicações P2P tem como criticidade o suporte de buscas e atualizações de informações. Visto que os dados estão distribuídos, e a quantidade de nodos na rede pode ser muito grande de forma que uma busca mais simples, de força bruta por exemplo, por determinado recurso na rede pode levar um tempo inviável para a aplicação, é preciso que operações de indexação e busca seja sofisticados em sistema deste modelo. Kurose

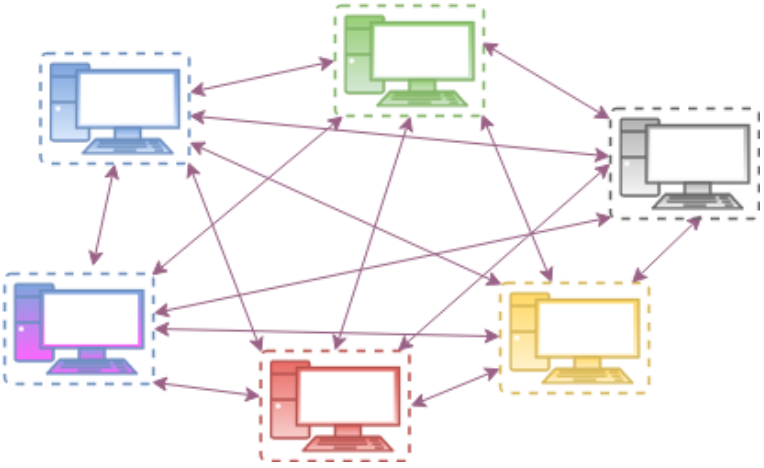

Figura 2.3: Modelo *Peer to Peer*

e Ross (2010) aponta as *Distributed Hash Table* (DHT) como uma técnica popular para estas operações, em que os nodos são organizados com base em uma ordenação lógica e cada participante responsabiliza-se por um espaço de chaves relacionadas aos dados armazenados (mais detalhes na Seção 2.4).

- Modelo Híbrido: como o nome diz, o modelo híbrido faz um compilado entre o modelo *cliente-servidor* e *peer to peer*, para sanar demandas específicas de uma aplicação como por exemplo, a utilização de vários servidores num modelo *peer to peer*, para aumentar o desempenho e resiliência dos recursos.

Dada a contextualização apresentado até o presente ponto, o resto desta seção apresentará os conceitos sobre redes com tolerância a falhas e posteriormente algumas soluções que utiliza-se *Distributed Hash Tables* (DHTs).

2.3 Tolerância a falhas

Os sistemas distribuídos tem uma particularidade de que todos os nodos (cada participante da rede) estão sujeitos a falhas, necessitando assim de um artifício para diagnosticar (detectar) falhas e reorganizar os nodos para que o sistema não fique corrompido. O diagnóstico ocorre por meio de testes de falhas enquanto o sistema estiver em execução, ou seja, um determinado algoritmo encarrega-se em testar todos os nodos averiguando quem está falho. A seguir uma breve descrição dos principais algoritmos de diagnóstico distribuído finalizando com o *VCube*,

que é o algoritmo de diagnóstico distribuído que este trabalho utilizará.

O primeiro algoritmo de diagnóstico foi o PMC (PREPARATA; METZE; CHIEN, 1967), nome que deriva das iniciais dos autores. Este modelo depende de um nó central livre de falhas chamado de *observador*. O observador realiza testes constantemente a todos os participantes da rede, e armazena o estado dos mesmos. Como é impossível garantir que o observador não irá falhar, este modelo apresenta restrições.

O próximo modelo, chamado *New-self* (KUHL; REDDY, 1980) foi o primeiro modelo a realizar diagnóstico de maneira distribuída, em que cada nó do sistema realiza uma certa quantidade de testes em seus vizinhos (pré-determinados) e compartilha aos nós não falhos a informação dos nós que estão falhos.

Posteriormente, Bianchini Jr. e Buskens (1991) propôs o *Adaptive-DSD*, um algoritmo que além de ser realizado de maneira distribuída, a quantidade de testes e os nodos que cada testador realizará são determinados em tempo de execução de forma adaptativa baseado no resultado obtidos nos testes anteriores. Para realizar esta abordagem, os nós são organizados em forma de anel e cada nodo *p* testa seu sucessor *s* apenas. Caso *p* verifique que *s* está falho, o mesmo teste é realizado no próximo nodo *r* do anel. Dessa forma o nodo *p* informará *r* que *s* está falho e todos os próximos nodos serão notificados quanto a falha de *s*. O problema deste algoritmo é a alta latência de disseminação de informação, visto que no contexto do exemplo anterior, o nó *a* antecessor de *p* será notificado sobre a falha de *s* depois de *n* testes, em que *n* é a quantidade de nodos na rede.

Com o objetivo de diminuir a latência de disseminação de falhas do algoritmo *Adaptive-DSD*, Duarte Jr. e Nanya (1998) propôs uma abordagem hierárquica do algoritmo, batizando-o como Hi-ADSD. Para este modelo, o conceito de *clusters* foi aplicado à organização da realização dos testes entre os nós. Dessa forma, a rede contém $log_2 N$ níveis de *clusters*, em que N é o tamanho da rede. A Figura 2.4 apresenta os *clusters* na visão do nodo 0 em uma rede com oito nodos. Observe que o nodo 0 mantém um enlace com um participante de cada nível. Cada nível contém 2^{s-1} participantes, em que s é o nível do *cluster*.

Um nodo realiza testes inicialmente em seu vizinho de *cluster* de nível um, posteriormente nos vizinhos de *cluster* de nível dois até testar todos os *clusters* da rede. Para listar os participantes de um *cluster* na visão de um nodo i utiliza da função recursiva $C(i, s)$ presente na

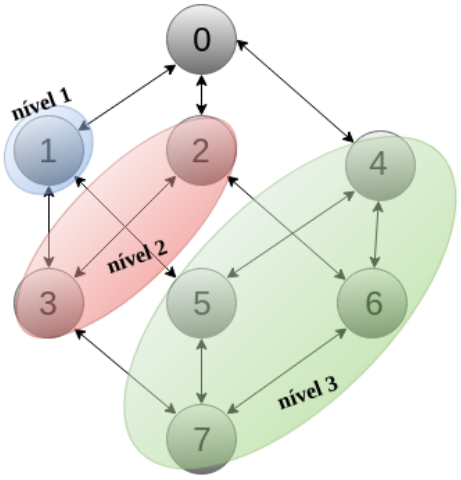

Figura 2.4: *Clusters* na visão do nodo 0, em uma rede de oito nodos

Tabela 2.1: $C_{i,s}$ para um sistema com oito participantes.

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	4,6,7,5	5,7,6,4	6,4,5,7	7,5,4,6	0,2,3,1	1,3,2,0	2,0,1,3	3,1,0,2

equação 2.1. A tabela 2.1 apresenta os possíveis resultados da Equação 2.1 para um sistema de oito nodos.

$$C_{i,s} = i \oplus 2^{s-1}, C_{i\oplus 2^{s-1},s-1}, ..., C_{i\oplus 2^{s-1},1} \tag{2.1}$$

Para diminuir a quantidade de testes para a execução do algoritmo, duas alternativas ao *Hi-ADSD* foram propostas: o *Hi-ADSD with Detours* e o *Hi-ADSD with Timestamp* (DUARTE Jr. et al., 2009). A primeira alternativa determina que o nó testador pare de testar os demais nodos de um *cluster* quando encontrar o primeiro falho. O nó testador obterá informações sobre os demais nós através de outros *clusters*, e caso não consiga, testes extras são realizados. A segunda proposta determina que cada nó manterá um vetor *timestamp* com *N* posições, em que *N* é o tamanho da rede. Cada índice representa o identificador do nó e inicializa em zero. Quando detectado um evento em um nó *i*, a posição *i* do vetor será incrementado. Dessa forma quando a posição apresentar um valor par, quer dizer que na visão local do *peer* que aloca o *timestamp* o nodo está ativo, caso contrário o nodo está apresentando falha. Esta abordagem permite que um nodo *p* obtenham informações sobre outros nodos *r* sem testar os mesmos, mas

por meio de testes que foram realizados por terceiros. Além disso, entende-se que o *timestamp* que obter o maior valor é o timestamp mais atual. Dessa forma, um nó testador pode limitar-se em testar apenas o primeiro nó não falho de um determinado *cluster*, diminuindo a quantidade de testes e consequentemente o tráfego de mensagens gerado pelo algoritmo.

O algoritmo DiVHA (BONA et al., 1996) utiliza-se do conceito de *clusters* em que o nodo testador testa apenas um nodo por nível, ou seja, é inspirado no algoritmo *Hi-ADSD with Timestamps* e tem como objetivo "minimizar o número de testes necessários a tornar o algoritmo determinístico" (DUARTE Jr. et al., 2005). Este algoritmo é modelado utilizando grafos. O grafo $H(S)$ representa um hipercubo completo, o que ocorre quando não há nodos falhos no sistema. Desta forma algumas propriedades logarítmicas são garantidas, como por exemplo, a distância máxima entre dois nodos é de $log_2 N$. Quando a rede apresenta nodos falhos, um grafo $T(S)$ é representado, onde são removidas as arestas que conectam os nodos falhos de $H(S)$. Considere que cada aresta define um teste que deve ser realizado. Desta forma um grafo $T_H(S)$ é construído a partir de $T(S)$ com arestas adicionais de forma a garantir que nenhum nó ficará isolado e consequentemente todos os nós será testado por pelo menos um nodo sem falha.

A construção de $T_H(S)$ é realizada em dois passos. No primeiro determina-se o conjunto dos nós falhos que não recebem arestas. Posteriormente, cada nó deste conjunto receberá um testador que é um nó não falho que tem a menor distância mínima para o nó falho. Caso exista concorrência o nodo testador será aquele que que pertence ao menor *cluster* em relação ao nó falho, de acordo com a função $C(i, s)$. O segundo passo tem como objetivo manter o grafo de forma que os nós não falhos tenham sempre caminhos mínimos entre eles. Neste momento, arestas são inseridas para reconstruir caminhos que se perderam com a presença de nós falhos.

Finalmente como último algoritmo de diagnóstico distribuído, o VCube (DUARTE Jr.; BONA; RUOSO, 2014) é uma pequena alteração no algoritmo *Hi-ADSD with timestamp* (RUOSO, 2013). Dessa forma, o VCube também utiliza do conceito de *clusters* e são organizados com base na função *C(i,s)*. No Algoritmo 1, é apresentado o pseudo código do algoritmo executado no nodo *i*. Considere o vetor t_i como o vetor *timestamp*.

O nodo realiza intervalo de testes para cada *cluster*, iniciando no de nível 1 e finalizando no *cluster* de maior nível do sistema. Visto que a função $C(i, s)$ retorna uma lista dos nodos vizinhos de *i* no *cluster s*, o VCube determina que *i* testara apenas o nodo $j \in C(i, s)$, se *i* for o

primeiro nodo não falho de $C(j, s)$. Esta verificação é realizada em todos os nodos pertencentes a $C(i, s)$.

Algoritmo 1: Algoritmo VCube (RUOSO, 2013)

```
início
    repita
        para s ← 1 até log2N faça
            para todo j ∈ C_{i,s} | i é o primeiro nodo sem falha ∈ C_{j,s} faça
                teste(j)
                se j está sem falha então
                    se t_i[j] mod 2 = 1 então
                        t_i[j] + +
                    fim
                    obtém informações de diagnóstico
                fim
                senão
                    se t_i[j] mod 2 = 0 então
                        t_i[j] + +
                    fim
                fim
            fim
        fim
    até para sempre;
    durma até o próximo intervalo de testes
fim
```

2.4 *Distributed Hash Table* - DHTs

Como introduzido anteriormente, DHT é um sistema distribuído que oferece mecanismos para a distribuição e localização de dados em redes de computadores (KUROSE; ROSS, 2010). Para realizar estes mecanismos, o sistema mapeia uma chave *k* para um *peer p* utilizando uma função *hash* (que neste trabalho será o SHA-256) (AKBARINIA; PACITTI; VALDURIEZ, 2007). Cada participante armazena uma tabela com uma tupla <*chave,valor*> em que chave corresponde ao resultado *hash* de algum índice daquele valor armazenado, que muitas vezes é o nome. Dessa forma, caso um nodo queira requisitar um determinado recurso, primeiramente aplicará o nome daquele recurso numa função *hash*, resultando numa chave *k*. Por meio de uma tabela de roteamento que é mantida através dos mecanismos citados anteriormente, aquele *peer* saberá qual o participante ou bloco de participantes responsáveis por um espaço de chaves que

compreende aquela chave *k* resultante da função *hash*.

Com a informação do nodo responsável a um recurso, o requisitante realizará uma busca de salto único ou múltiplos saltos, o que será determinado pelo modelo do DHT. De acordo com Koppe (2013), as DHTs fazem uso de tabelas de roteamento com uma visão parcial ou total do sistema. O sistema DHT que utiliza tabelas com visão parcial do sistema são chamados de *Multi-hop* DHT (DHT de múltiplos saltos), pois para qualquer busca, o peer deverá consultar a visão parcial de outros nodos até encontrar a informação referente a chave da busca (cada consulta é chamada de um salto). Sistemas DHT que utilizam tabelas de roteamento com visão total do sistema são chamadas de *Single Hop* DHT (DHT de salto único), pois, com as informações da tabela de roteamento local, é possível fazer a consulta de uma determinada chave localmente. Quanto mais saltos uma busca exigir, maior será o tempo para o nodo requisitante receber a sua resposta. Koppe (2013) diz que "o principal objetivo das DHTs de salto único é minimizar ao máximo a latência das consultas sem comprometer o sistema com o consumo excessivo de recursos de rede para a manutenção das tabelas".

A seguir apresenta-se algumas DHT de múltiplos saltos e posteriormente de salto único.

2.4.1 DHTs de Múltiplos Saltos

Nesta seção, são descritos três DHTs de múltiplos saltos. A escolha de descrever estas baseou-se na particularidade que cada um apresenta: A primeira solução chama-se CAN (*Content-Addressable Network*) e baseia sua arquitetura em um espaço cartesiano virtual e multi-dimensional de coordenadas. A segunda solução chama-se Chord e organiza os espaços de chaves em uma topologia anel. A terceira solução chama-se Kademlia e organiza suas chaves em uma árvore binária.

CAN

Proposto por Ratnasamy et al. (2001), e como descrito anteriormente, CAN organiza as faixas da função *hash* num plano cartesiano. Considere na Figura 2.5 um possível cenário de organização dos participantes. Cada zona com cor diferente é o espaço de um participante. Note que cada participante pode ocupar zonas com tamanhos diferentes mas sempre com formato quadrático o que permite visualizar vizinhos, que são aqueles participantes e que fazem divisa

com os quatro lados do *peer*. Observe que o participante presente no quanto superior esquerdo não é vizinho do participante em destaque pois estão dispostos em diagonal.

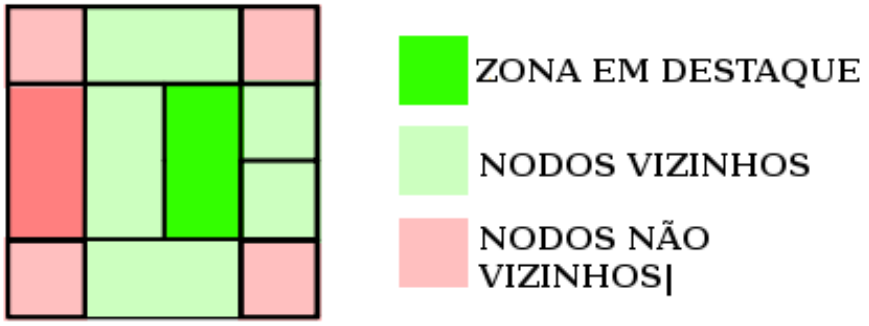

Figura 2.5: Zonas de um sistema CAN. Adaptado de Ratnasamy et al. (2001)

Diante deste contexto, cada participante mantém informações sobre faixas de *hashs* de seus vizinhos, ou seja, uma visão parcial de todo o sistema. No cenário de uma possível busca por um recurso, inicialmente o *peer* elege o vizinho que está mais próximo daquela chave, podendo aplicar a fórmula da distância euclidiana em todos os vizinhos (visto que são organizados em um plano cartesiano) e determinar qual vizinho é o mais próximo do recurso. Uma solicitação é enviada a esse vizinho que por sua vez realiza o mesmo procedimento até encontrar o participante responsável pelo recurso. Caso algum vizinho ou caminho esteja falho, é definido caminhos alternativos com tentativas de envios de solicitação para o segundo vizinho.

Para a entrada de um novo participante Z na rede, Z deve conhecer um participante da rede E. Desta forma, nodo Z escolhe um ponto aleatório do espaço de coordenadas e envia uma solicitação ao responsável X por intermédio do nodo E, ou seja, E define uma rota até o ponto escolhido por Z seguindo os passos da busca por um recurso. Encontrado esta rota, o novo participante Z divide o espaço de chaves e vizinhos com o participante X de forma que Z e X se transformarão em vizinhos como ilustra a Figura 2.6.

Dada as funcionalidades deste modelo, características importantes para um sistema *peer to peer* são garantidas, como: escalabilidade, recursos distribuídos, eficiência e tolerância a falhas e balanceamento de carga.

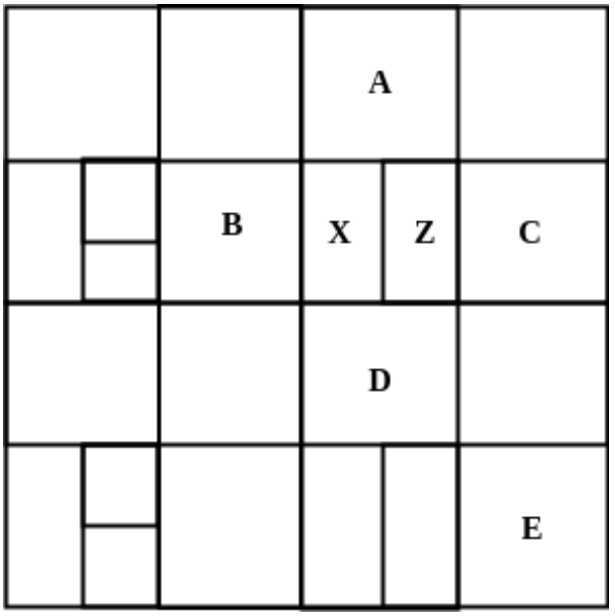

Figura 2.6: Exemplo de entrada de Z em um sistema CAN. Adaptado de Koppe (2013)

Chord

Proposto por Stoica et al. (2001), o protocolo Chord suporta uma única operação: dada uma chave, ela é mapeada num determinado nodo por meio da função *hash* SHA-1, e tem como objetivo executar os seguintes desafios encontrados em redes *peer-to-peer*: balanceamento de carga, decentralização, escalabilidade, disponibilidade e nomes flexíveis. Neste contexto, cada nodo recebe um identificador único que é uma operação *hash* de seu endereço IP, de forma que todos os resultados possíveis da função *hash* geram uma sequência de chaves, que neste protocolo é chamado de *Chord Ring*, pois são organizados numa topologia em anel.

Quando um participante *p* entra na rede, será responsável pelo espaço de chave que compreende o espaço que inicia na próxima chave do responsável anterior, até a chave que é igual ao identificador *p*. As chaves *k* que estão sob responsabilidade de *p*, o vêem como *sucessor(k)*. Para exemplificar, observe a Figura 2.7. Considere que um novo participante *p* tem como identificador o valor 15. Note que já existem alguns participantes na rede, em especial o nó 25 que é responsável pelo espaço de chave que entre as chaves 6 e 25. Observe que o início deste espaço de chave é exatamente a próxima chave do espaço de chaves do *peer* com identificador 5. Dessa forma o nó 15 deverá receber um espaço de chaves para ser responsável. Neste caso o sistema deverá realizar um processamento de transferir o espaço de chave que tem como início a chave

6 e como fim a chave 15. Além disso deve ser retirado este intervalo da responsabilidade do nó 25, que por sua vez será responsável apenas pelo intervalo de chaves de 16 a 25.

Com essas considerações é possível entender o funcionamento de uma requisição de recurso também chamado de *lookup*. Suponha que o *peer* 15 (neste momento já participando da rede), queira requisitar o recurso que tem a chave 70. Segundo o protocolo Chord, o *peer* requisitante deverá fazer um pedido para os próximos *peers* sucessores até que encontre o responsável por aquele recurso. Cada consulta realizada é chamada de um salto, e é ilustrada na Figura 2.7 como setas que passam de nó a nó. Note a quantidade de consultas são realizadas até que é encontrado o responsável pela chave 70, que no caso é o *peer* 71. A resposta para a requisição é realizada seguindo o caminho contrário das setas. Este processo de encontrar o responsável por determinado recurso denominada pela operação *findsucessor*, que tem como argumento a chave do recurso.

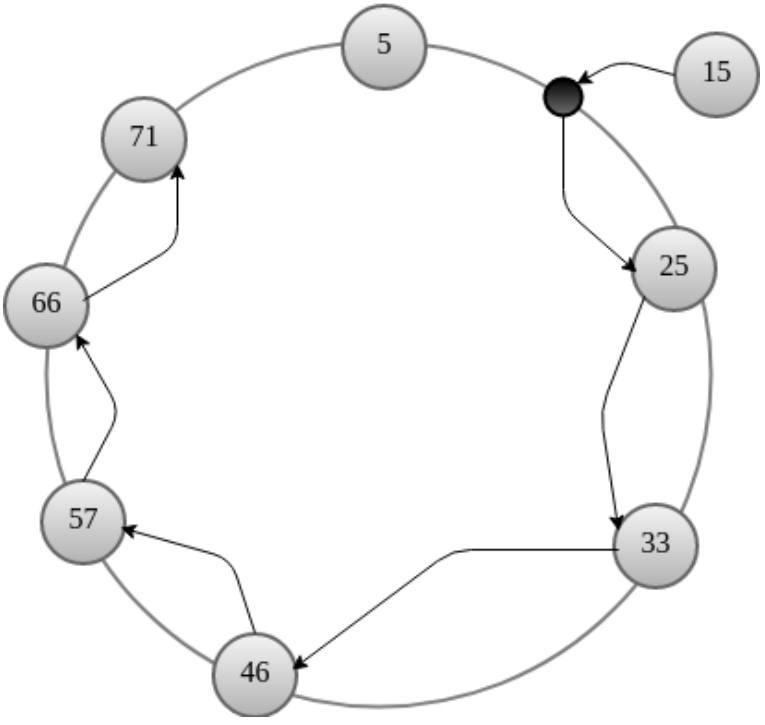

Figura 2.7: Representação de um sistema Chord.

Nota-se que até neste momento, o protocolo se mostra muito custoso. Com o objetivo de diminuir a quantidade de saltos necessários para realizar uma consulta, o protocolo utiliza de uma tabela de roteamento em cada *peer*, chamada de *finger table*. Esta tabela contém *m* entradas em que *m* é a quantidade de bits do resultado da função *hash*. Cada *i-ésima* entrada compreende

em um tupla com duas informações. A primeira informação é uma chave que é o resultado da soma $S = k + 2^{i-1}$, em que k é a chave do identificador do *peer*. A segunda informação é o sucessor da chave S encontrada, que encontra-se através da função *findsucessor(S)*. A Figura 2.8, retirada de Koppe (2013), ilustra como seria a *finger table* do *peer* $N8$ em que a função *hash* resulta em chaves de seis bits. As setas indicam o resultado da soma S para cada entrada da tabela. Note que neste cenário, o *peer* $N8$ armazenou três vezes uma chave para o sucessor $N14$.

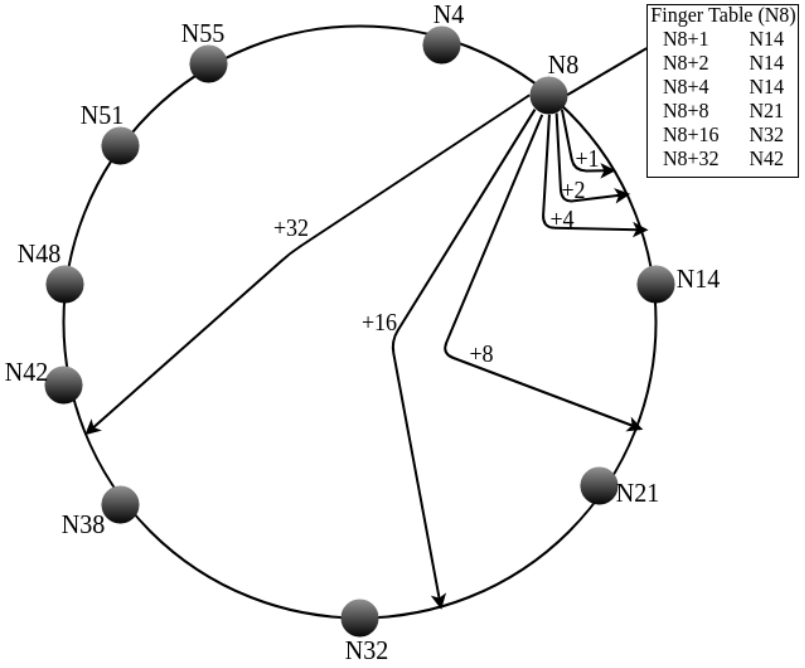

Figura 2.8: Exemplo de uma *Finger Table*. Adaptado de Koppe (2013)

Com este artifício, o nodo que requisitar um determinado recurso terá uma visão parcial do sistema, de forma a realizar saltos mais longos e consequentemente em menor quantidade para encontrar uma certa chave. É importante notar aqui, que o sistema necessita de uma ferramenta que verifique constantemente toda a rede para manter todas as *finger tables* atualizadas.

Kademlia

Proposto por Maymounkov e Mazières (2002), Kademlia teve uma grande contribuição ao aprimorar técnicas de outras soluções DHT, como o Chord, oferecendo uma tabela de roteamento flexível e minimizando o número de troca de mensagens de configurações, principalmente para atualizar as tabelas de roteamento. Esta solução foi uma das poucas DHTs imple-

mentadas em soluções para uso comercial. No caso, o sistema KAD (utilizado por aplicações como o eMule, Overnet e aMule) foi implementado baseado no Kademlia (STEINER; ENNAJJARY; BIERSACK, 2007a).

Os participantes da rede são identificados por IDs, que assim como no Chord, pode ser uma função *hash* do endereço IP do nodo. Além disso, os identificadores também são responsáveis por uma faixa de chave que são responsáveis por determinados recursos. Para isto ocorrer, o Kademlia trata os nós como folhas de uma árvore binária e a posição de cada nó é determinada pelo menor prefixo exclusivo de seu identificador. Os pares <*chave*, *valor*>, em que o valor seria o recurso armazenado e a chave a *hash* do valor, são armazenados no nó que a chave compartilha o maior prefixo com o identificador.

Kademlia armazena uma tabela de roteamento com m entradas, em que m é a quantidade de bits da chave gerada pela função *hash*, a mesma quantidade de entradas que a *finger table* presente no Chord. Geralmente, cada entrada contém uma tupla com duas informações principais: <*endereço IP*, Identificador do nó>. Esta tabela é chamada de *k-bucket* em que k é o mesmo valor de m. A lista inicia vazia e conforme o *peer* recebe mensagens de outros *peers* a tabela é atualizada com uma nova tupla ao fim da lista. Caso aquele endereço IP já exista na lista, o mesmo é enviado para o fim. Caso a lista esteja cheia, o destinatário realiza um *ping* no endereço IP menos recentemente visto que é o remetente do topo da lista. Caso obtenha resposta, o novo remetente é descartado, caso contrário o remetente antigo é descartado e o novo remetente é inserido no fim da lista. Está tem como política não armazenar dados de nós que tem mais probabilidade de estarem falhos, pois trocam mensagens com menos frequência. Além disso o protocolo armazena dados de nós que tem mais probabilidade de não estarem falhos, pois trocam mensagens com mais frequência.

São quatro tipos de mensagens que sistemas Kademlia executa: *ping*, *store*, *findNode* e *findValue*. O primeiro tipo mensagem (*ping*) tem como objetivo testar o estado de um determinado nodo que é informado como argumento. As mensagens do tipo *store* contém uma tupla com informações de chave e valor sobre um recurso que será armazenado na rede. As mensagens *findNode* tem como argumento um identificador e o destinatário retornará uma tupla <endereço IP, porta UDP, Identificador do nó> sobre o nó mais próximo conhecido daquela identificador. Mensagens do tipo *findValue* funcionam da mesma forma que as mensagens *findNode*, com a

diferença que o argumento será uma chave e não um identificador. Caso o destinatário estiver armazenando o recurso relacionado a aquela chave, o retorno será o próprio recurso.

Para localizar os *k* nodos mais próximos de um determinado identificador, o nodo interessado envia mensagens *findNode* assíncronas para *n peers* mais próximos da sua tabela *k-buchet*, em que *n* é geralmente três, mas depende do projeto do sistema. Quando os nós recebem a solicitação *findNode*, retornam os nós conhecidos mais próximos à chave informada. Dessa forma o nó que que iniciou a pesquisa atualiza sua lista de nodos mais próximos a aquela chave e realiza a mesma consulta, agora com os nós mais próximos. Este procedimento se repete até que o nó não atualize mais esta lista, de forma que serão os nós mais próximos da chave desejada.

Para um certo nodo *p* entrar na rede, inicialmente deverá ter contato com um participante e iniciar contato com o mesmo. Este participante armazenará o novo nodo em seus *k-buckets* atribuindo-o um identificador (que pode ser uma faixa da rede que tem poucos responsáveis). Para seguir com o processo, o novo nó realizara o chamado *self-lookup*, que seria um *node-Lookup* com seu próprio identificador ao participante conhecido que por sua vez retornará os nós com identificador mais próximo a aquele identificador. Dessa forma o novo *peer* seguirá o processo do *nodeLookup* até encontrar os *peers* mais próximo a ele e consequentemente definir seu lugar na árvore. Todo esse processo atualizará os nodos que foram requisitados.

2.4.2 DHTs de Salto Único

Nesta seção, será descrito as seguintes soluções DHTs de salto único: *OneHope* DHT, D1HT e HyperDHT. As duas primeiras de forma menos detalhadas, visto que utilizam topologia anel e são bastante semelhantes a solução de múltiplos saltos do Chord. A terceira solução será mais detalhada pelo fato de apresentar a particularidade de utilizar topologia de hipercubo. Uma informação preliminar é que soluções DHTs deste gênero dependem da disseminação de um novo evento para toda a rede no momento que for detectado para manter as tabelas de roteamento mais tempo atualizada, visto que desde o início da disseminação até o momento que o individuo recebe a mensagem e de fato atualiza sua tabela de roteamento, este indivíduo estará com a tabela desatualizada.

OneHope DHT

Proposto por Gupta, Liskov e Rodrigues (2004), foi a primeira solução DHT de salto único, ou seja, armazena uma tabela de roteamento que armazena informações sobre todos os participantes da rede. Como dito anteriormente, esta solução ordena os identificadores da rede em um anel, similar ao Chord, de forma que cada *peer* tem conhecimento de seu sucessor e antecessor na topologia.

Para as informações dos participantes estarem atualizadas em todos os *peers* da rede, mensagens de atualização de estado são propagadas pela rede constantemente seguindo uma estrutura hierárquica. Esta estrutura divide os participantes em grupos chamados *slices*. Cada *slice* possui vários grupos de tamanhos iguais chamados *unidade*. Cada *slice* e *unidade* contém um nodo líder. As atualizações podem ser a detecção de uma falha ou a entrada de um novo nodo na rede.

O líder de um *slice* é responsável em reunir e organizar todas as atualizações de suas unidades e notificar todos os outros líderes dos demais *slices* da rede. Quanto um líder de *slice* recebe a notificação de atualizações na rede, repassa para os líderes de suas unidades que repassam para todos os nós da unidade.

D1HT

Proposto por (MONNERAT; AMORIM, 2006), esta solução organiza os nós da rede em topologia anel, e assim como o Chord cada participante conhece apenas seu sucessor. Para manter as tabelas de roteamento atualizadas, o sistema utiliza do algoritmo de disseminação de eventos EDRA (*Event Detection and Report Algorithm*). Este algoritmo utiliza de uma tabela de referência construída da mesma forma que o *Finger Table*, utilizada pelo Chord e garante que a detecção de um novo evento seja disseminado a todos os nós em tempo logarítmico.

Quando um *peer* falha, seu sucessor detecta e dissemina para todos os nós presentes na tabela de referência juntamente com um parâmetro chamado TTL (*time to live*) que seria o índice da tabela de referência que armazena o nodo (iniciando em zero). Quando um nó recebe uma mensagem de disseminação de evento, ele atualiza sua tabela de roteamento e repassa para os próximos TTL nós de sua tabela de referência, juntamente com o índice TTL local. Caso o TTL recebido seja igual a zero, o nó não repassa a informação de evento.

Hyper DHT

Esta solução foi proposta por Koppe (2013), e utiliza a infra-estrutura fornecida pelo algoritmo DiVHA para detectar eventos em tempo logarítmico bem como disseminar atualizações na tabela de roteamento seguindo a mesma estrutura hierárquica. Cada vértice do grafo formado pelo DiVHA representa uma posição da rede de sobreposição que pode estar ocupado por um nodo (sendo assim sem falha), ou estar vazio.

Para manter o balanceamento de carga entre os nodos do sistema, esta solução utiliza o $SHA-1$ de 160 bits para atribuir aos identificadores. Dessa forma, a rede pode conter até 2^{160} identificadores diferentes. O protocolo de entrada garante alocar novos *peers* em regiões do grafo em que tem carência de nodos. Este local será na região onde existir o maior espaço de chaves sob responsabilidade de apenas um nodo, de forma que o espaço será dividido entre o nodo com maior carga e o novo participante da rede.

O processo de entrada de um novo nó é iniciado quando um nodo interessado contacta um participante conhecido, que por sua vez e através da sua visão global do sistema determina o local mais adequado para ser alocado o novo participante sem a necessidade de mensagens adicionais. Caso o nó participante determine que não existe nó vazio, um procedimento de aumento do sistema é realizado, de forma que a quantidade de vértices será duplicado e todos os nodos serão realocados no intuito de manter o balanceamento de carga.

VCubeDHT

Em Barreiro Neto, Rodrigues e Duarte Jr. (2017), uma nova solução é proposta chamada de VCubeDHT. O autor foca principalmente em funcionalidades de replicação de dados, o que garante disponibilidade de um certo recurso uma vez que vários nós o contém. Assim como no HyperDHT, Barreiro Neto, Rodrigues e Duarte Jr. (2017) baseia-se num algoritmo de diagnóstico distribuído, que no caso é o VCube, utilizando de suas propriedades logarítmicas e para detectar falhas e disseminar informações. O presente trabalho implementará o VCubeDHT, propondo funcionalidades, principalmente de recuperação e armazenamento de dados, com foco em manter uma rede no contexto de *big data*. A solução proposta será descrita no Capítulo 5.

Capítulo 3

Revisão Sistemática

Diante das vantagens da utilização de DHTs, principalmente relacionadas a características sobre inserção e localização de dados, que variam em diferentes soluções apresentadas no capitulo 2, juntamente com os desafios apresentados no contexto *big data*, propõem-se uma investigação sobre a eficiência do uso de DHT em ambiente distribuído com grande volumes de dados. Para isto, e seguindo a citação de Khan et al. (2003), em que aplicando-se uma revisão sistemática num problema bem formulado, e a partir de um conjunto de estudos e obras, é possível identificar obras relevantes ao tema e distingui-las de obras que podem não agregar-se ao contexto, realizou-se uma revisão sistemático descrita a seguir. Primeiramente será descrito os passos a serem realizados, e posteriormente a execução dos mesmos.

3.1 Metodologia

Revisão sistemática (RS) é um dos "principais meios para sumarizar evidências de pesquisas" e tem como objetivo "identificar, selecionar, avaliar, interpretar e sumarizar estudos considerados relevantes para um tópico de pesquisa ou fenômeno de interesse" (KITCHENHAM et al., 2009) (KITCHENHAM, 2004) (BIOLCHINI et al., 2005) (NAKAGAWA et al., 2017). Este artifício permite ao aplicador da RS conhecimento sobre o tema de forma a apoiar a identificação de tópicos para pesquisas futuras pois é composto por fases rigorosas e bem definidas para selecionar obras sobre o tema (BRERETON et al., 2007) (NAKAGAWA et al., 2017).

Uma RS apresenta uma alta confiabilidade de resultado visto que dois pesquisadores experientes realizando a mesma revisão chegam em resultados muito parecidos. Por outro lado, uma RS realizada por dois estudantes de graduação, por exemplo, pode apresentar resultados

divergentes não tendo garantia de confiabilidade. Dessa forma é recomendável que este tipo de pesquisa seja realizada em grupo, mesmo que seja uma tarefa desafiadora e demorada (NAKAGAWA et al., 2017). No caso deste trabalho, a RS foi realizada individualmente com duas justificativas: por ser um trabalho de conclusão de curso e claramente agregar um grande valor a obra, tendo assim mais possibilidades de embasar-se no maior número de textos sobre o tema aqui tratado; abrindo a possibilidade de um trabalho futuro ser realizado sobre o mesmo tema utilizando esta pesquisa em efeitos de comparações. Como vantagem de uma RS comparado com revisões informais, destaca-se a redução de vieses na seleção bem como a possibilidade de identificar e combinar as características das diversas obras selecionadas na revisão.

Será seguido nesta revisão sistemática, cinco passos propostos por Khan et al. (2003):

1. Definir perguntas para revisão: Especificar na forma de perguntas claras, inequívocas e bem estruturadas.

2. Selecionar obras literárias sobre o tema: A busca deve ser extensa e não deve ter restrição de idioma. A seleção deve ser criteriosa de acordo com as perguntas definidas no passo anterior.

3. Avaliar a qualidade das obras selecionadas: Uma avaliação mais detalhada deve ser aplicada nas obras selecionadas através de um guia de avaliação crítica geral e listas de verificação de qualidade baseada em padrões.

4. Resumir as evidências: O resumo será baseado numa classificação das características do estudo, qualidades e efeitos, explorando assim, diferenças entre estudos e combinar os efeitos.

5. Interpretar as evidências reunidas: Depois da realização de todos os passos anteriores, uma revisão deve ser feita nos resultados reunidos de forma a identificar se a qualidade do resumo está confiável, ou algum passo mal feito pode ter resultado numa informação duvidosa.

A seguir um detalhamento dos passos executados neste trabalho.

3.2 Execução

3.2.1 Definição de perguntas para revisão

A execução deste primeiro passo teve como objetivo criar consultas, respeitando padrões de pesquisa e idioma de bibliotecas onlines, relacionando as seguintes palavras chaves: "DHT", "*big data*", "*application*", "*storage*", "*lookup*", "*distributed*", "*hash table*", "*replication*" e "*availability*". As consultas são criadas através de operadores lógicos que relacionam os termos, como por exemplo:

- "*big data*" retornará todas as obras que contém o termo "*big data*", nos campos detalhas no motor de pesquisa;
- "*big data*" **AND** "DHT" retornará todos os textos que contenham necessariamente os dois termos;
- "*big data*" **OR** "DHT" retornará todos os textos que contenham pelo menos um dos dois termos;
- "*big data*" **+** "DHT": retornará obras que possam conter o termo "*big data*" ou não, mas obrigatoriamente obras que contenham o termo "DHT".
- **+** "*big data*" **+** "DHT": terá o mesmo efeito do operador AND, visto que obrigatoriamente as obras deverão conter os dois termos.

As consultas criadas são apresentadas na Tabela 3.1.

3.2.2 Seleção das obras literárias

Duas bibliotecas digitais foram escolhidas para aplicar as consultas: *ACM Digital Library* (LIBRARY, 2018a) e *IEEExplore Digital Library* (LIBRARY, 2018b). Essa escolha deu-se principalmente à alta popularidade que ambas apresentam na área da Ciência de Computação. As consultas presente na Tabela 3.1 foram realizadas em cada motor de pesquisa de forma a retornar obras que apresentam os termos em seus títulos, resumos e palavras-chaves. Isto limita o campo de busca diminuindo a probabilidade de selecionar obras de contextos totalmente

Tabela 3.1: Consultas definidas e seleção de obras nas bibliotecas digitais revisão sistemática *ACM Digital Library* e *IEEExplore Digital Library*.

ACM Digital Library		
Consultas	**Data de Acesso**	**Quantidade de obras resultadas**
"DHT"AND"*big data*"	23/05/2019	3
"applications"AND"*storage*"AND"*lookup*"AND "*distributed*"	23/05/2019	36
"*hash table*"AND"*big data*"	23/05/2019	13
"*lookup*"AND"*big data*"	23/05/2019	12
"*distributed hash table*"AND"*big data*"	23/05/2019	2
"*distributed hash table*"AND"*lookup*"AND"*storage*"	23/05/2019	12
"*availability*"AND"*distributed hash table*"	23/05/2019	43
"*replication*"AND"*big data*"AND"*distributed*"	23/05/2019	35
"*availability*"AND"*big data*"AND"*hash table*"	23/05/2019	4
IEEExplore Digital Library		
"DHT" AND "*big data*"	23/05/2019	11
"*big data*" AND "*lookup*" AND "*distributed*"	23/05/2019	17
"*big data*" AND "*distributed hash table*" AND "*lookup*" AND "*storage*"	23/05/2019	1
"*big data*" AND "*hash table*"	23/05/2019	31
"*distributed hash table*" AND "*lookup*" AND "*storage*"	23/05/2019	34
"*replication*" AND "*availability*" AND "*big data*" AND "*distributed*"	23/05/2019	32
"*big data*" AND "*storage*" AND "*lookup*" AND "*distributed*"	23/05/2019	13

divergente desta pesquisa. Note que algumas consultas são repetidas em diferentes bibliotecas digitais, mas outras não. Isto ocorreu por que existe diferença de disponibilidade de textos entre as bibliotecas e algumas pesquisas retornam uma quantidade significativa de obras em uma biblioteca, mas com resultado igual a zero em outra.

Nesta etapa foram reunidas 299 (duzentas e noventa e nove) obras, 160 (cento e sessenta) da *ACM*, e 139 (cento e trinta e nove) da *IEEExplore Digital Library*. Elas foram catalogadas, armazenando-se o título da obra, o(s) autor(es), palavras chaves, data da consulta, ano de publicação e biblioteca pesquisada. Posteriormente, as obras foram avaliadas, como descrito no próximo passo.

3.2.3 Avaliação das obras reunidas

Na etapa anterior a busca limitou-se em textos que continham aquelas consultas em seus resumos, títulos e palavras chave reduzindo assim o campo de busca. Mesmo assim, diversos textos fora do contexto de sistemas distribuídos foram selecionados, como textos sobre física, inteligência artificial, soluções de alto nível, ignorando assim os conceitos de DHT e topologias de redes. Dessa forma, nesta terceira etapa criou-se mais um atributo no catálogo das obras, chamado "relevância". Para atribuir valor "sim" ou "não" foram lidos os resumos e títulos das obras, caracterizando-os relevantes ou não para a busca.

Das duzentas e noventa e nove obras reunidas inicialmente, esta etapa eliminou um pouco mais de 70% das obras, de forma que sobraram 46 (quarenta e seis) textos da *ACM Digital Library* e apenas 25 (vinte e cinco) textos da *IEEExplore Digital Library*. Nota-se que neste tema, o motor de busca da *IEEExplore Digital Library* mostrou-se mais eficiente. Esta etapa reduziu a quantidade de textos da RS para 71 (setenta e uma) obras.

Visto que um texto pode estar presente nas duas bibliotecas e também ser resultado de mais do que uma consulta, constatou-se nesta etapa que diversos textos reunidos estavam repetidos. Dessa forma realizou-se uma etapa de retirada de textos replicados e, das 71 (setenta e uma) obras classificadas como relevantes, retaram 46 (quarenta e seis), sendo 32 (trinta e dois) textos oriundos da *ACM Digital Library* e 14 (quatorze) obras da *IEEExplore Digital Library*. Na Tabela 3.2 é apresentado a quantidade de obras que sobraram ao decorrer de cada etapa.

Tabela 3.2: Obras restantes em cada etapa.

Biblioteca	**Passo 2**	**Obras Relevantes**	**Obras não duplicadas**
ACM Digital Library	160	46	32
IEEExplore Digital Library	139	25	14
Total	299	71	46

3.2.4 Resumo das Evidências

Nesta etapa, as 46 (quarenta e seis) obras restantes na RS, foram de fato lidas, compreendidas e documentadas. Além disso constatou-se que diversas obras eram irrelevantes para o contexto da pesquisa. Neste caso, principalmente obras no contexto de virtualização, segurança, computação em nuvem, aplicações de alto nível, performance de hardware e até mesmo

de tolerância a falhas focado em replicabilidade foram descartadas por não tratarem de armazenamento, recuperação de dados e latência baseado em quantidade de saltos para realizar as funcionalidades da DHT. Obras que não eram artigos e obras não disponíveis foram eliminadas também. Dessa forma, restaram 21 (vinte e uma) obras, sendo apenas 11 (onze) obras da *ACM Digital Library* e 10 (dez) oriundas da *IEEExplore Digital Library*.

Para cada texto selecionado e lido, focou-se em identificar aspectos como: Se usa DHTs, se está no contexto de *big data*, de que forma, se trata de tolerância a falhas e se está no contexto de múltiplos saltos ou saltos únicos. Nas Tabelas 3.3 e 3.4 essas informações são apresentadas de acordo com cada texto, em que a primeira tabela apresenta os textos oriundos da *ACM Digital Library* e a segunda apresenta os textos oriundos da *IEEExplore Digital Library*. A relação entre as características dos textos será discutidos na próxima etapa. Quanto o resumo, que de fato é o resultado desta etapa, será apresentado respeitando os aspectos que foram identificados nas obras.

Uso das DHTs e Contexto de *big data*: Como o objetivo da RS é uma investigação da relação entre os temas *big data* e DHT, a maioria dos textos tratam de DHT, ou seja, utilizam uma solução DHT para executar todas ou algumas de suas funcionalidades que em geral apresentam-se como artifícios para tratar os problemas presentes no contexto de *big data*.

Neste contexto, os texto de Morselli et al. (2007) e de Sun et al. (2007) chamaram atenção por propor alternativas para redes desestruturadas, ou seja, que não utilizam de Tabelas *hashs* para roteamento mas que necessitam de *lookups* eficientes. Destacando a primeira citação, Morselli et al. (2007) explora o conceito de LMS (*Local Minima Search*), baseia-se em mapeamento de espaço de nomes e *hashing* consistente empregado por protocolos de topologia restrita, como a própria DHT. Além disso, os recursos são armazenados e replicados seguindo o mesmo princípio das soluções DHTs, em relacionar as *hashs* dos recursos com os identificadores de seus hospedeiros. Em contrapartida, não existe mecanismo de roteamento para localizar esses recursos. Ao invés disso, o conceito de *mínimo local* é utilizado, em que um identificador u é o mínimo local de um objeto, apenas se o identificador for o mais próximo da chave dentre os vizinhos do nó. Para definir rotas de mínimos locais para os objetos, é realizado envio de mensagens aleatórias durante a pesquisa. Como última consideração, os vizinhos não são dinâmicos.

Uma entidade externa aos nós é que define os links, que pode ser a própria infraestrutura, de forma que os enlaces são definidos pelos enlaces físicos. Ainda no mesmo contexto, Lloyd e Gokhale (2017) também não diz respeito às DHTs, mas apresenta uma topologia de hardware de forma a aproximar a memória do processador e então possibilitar aos nodos de uma rede terem maior poder de processamento (principalmente de busca), utilizando o FIFO (*first in first out*) para escalonar as requisições.

Cortés et al. (2015) e Cortés et al. (2016) propõe uma estratégia DHT juntamente com o conceito de LHT (*Location Hash Table*). Esta estratégia utiliza as coordenadas geográficas (latitude e longitude) de um nodo que está entrando na rede para determinado a faixa de chaves que o mesmo será responsável. Dessa forma os participantes ocupam faixas estratégicas, de acordo com sua posição geográfica, diminuindo a latência de eventuais pesquisas. Além disso, esta estratégia indexa informações de localização e dimensões temporais de cada meta-dado armazenado, permitindo assim um balanceamento de carga de forma a equilibrar as inserções e pesquisas na rede distribuída.

Topologia: No contexto de DHT é inevitável o aspecto de topologia não ser tratado. Temos alguns exemplos (Tabelas 3.3 e 3.4) de soluções que não estão amarradas em topologias e então foram classificadas como obras que não tratam da mesma. Fora estas obras, seis trabalhos selecionados utilizam anel e outras duas utilizam topologia de árvore. Este resultado deveu-se a popularidade de soluções como o *chord*, presente nos textos de Park et al. (2010), Zhang et al. (2011) e Amir, Srinivasan e Khan (2015) além de soluções como o *Kademlia*, presente no texto de Steiner, En-Najjary e Biersack (2007b).

A obra de Cheng et al. (2014) propõe uma solução hibridizada com o objetivo em diminuir a latência de consultas. Para isto, o sistema é dividido em dois níveis, em que o nível superior organiza os nodos com enlaces de alto desempenho, de forma que cada nodo é responsável por uma subrede organizada seguindo o protocolo do Chord, por este motivo o autor chama a topologia de *Mult-ring* que significa “Múltiplos anéis”.

Semelhante à solução *Kademlia*, Mohammed, Sharaf e Omara (2014) baseia a topologia em árvores, mas neste artigo o conceito de árvores radix é utilizado, em que o custo de atualização da árvore não é tão caro quanto as árvores B, além de maior rapidez nas consultas, inclusive ao

determinar a não correspondência de um certo valor. Sun et al. (2016) também propõe soluções utilizando árvores para indexar os espaços de chaves.

Tolerância a Falhas: Neste aspecto, a obra de Picconi e Sens (2006) chama a atenção por ter como objetivo aumentar a disponibilidade dos nós na rede diminuindo assim o chamados *churns*, que seria o evento de um nodo falhar. Ao reduzir os *churns*, a rede poupa tempo e processamento para proliferar a informação da falha e redistribuição dos recursos. Esta proposta armazena uma reputação para cada nodo da rede baseado no histórico de falhas, ou seja, nodos que costumam falhar recebem uma cota baixa para armazenamento de recursos, enquanto nodos que tem maiores reputação, além de terem cota de armazenamento maior, também recebem qualidade de serviços melhores como maiores larguras de banda e de download.

Também com o objetivo de aumentar a disponibilidade dos recursos na rede, Puthusseri et al. (2019) propõem um posicionamento inteligente dos fragmentos na rede, baseado no *P2P* e utilizando a topologia que o Kademlia oferece. Além desta regra, o protocolo para entrada de novos nodos é mais rígido. Os novos participantes são monitorados durante um determinado tempo e então recusados na rede, caso seu desempenho seja ruim, ou aceitos. Caso aceito, o monitoramento resultará num nível de reputação e o nó funcionará de acordo com este nível. Caso recusado, o sistema economizará tempo e recurso que gastaria com um nodo instável. A obra de Godfrey, Shenker e Stoica (2006) assemelha-se bastante com a solução de Picconi e Sens (2006).

Política de Saltos: Aspecto diretamente ligado à topologia utilizada. Dessa forma todas as obras tratam de DHTs de múltiplos-saltos, e a maioria, inclusive utiliza estas DHTs. Algumas obras não trataram de políticas de saltos, como é o exemplo de Godfrey, Shenker e Stoica (2006) e Picconi e Sens (2006), que se encaixam em diferentes soluções de DHTs. Este certo desinteresse das obras em utilizarem DHTs de salto único deve-se aos problemas em manter uma tabela de roteamento com milhões de linhas, de forma que deve ser atualizada constantemente. Diante deste contexto, Tang et al. (2005) propõem uma otimização no tamanho das tabelas de roteamento, em que alguns nós menos requisitados possam ser omitidos, baseado em uma função que envolve o tempo médio em que o nó mantem-se sem falhas e o número

médio de pesquisas que o nó processa por segundo. Diante desta função é possível que no decorrer da execução do sistema, o tamanho da tabela de roteamento altere várias vezes.

3.2.5 Interpretar as evidências reunidas

Observa-se na Tabela 3.3 o título dos textos obtidos através da biblioteca *ACM Digital Library* e algumas características que apresentam. Os mesmos dados são encontrados na Tabela 3.4 quanto às obras obtidas através da biblioteca *IEEExplore Digital Library*.

Tabela 3.3: Características das obras obtidas da biblioteca *ACM Digital Library* classificadas como relevantes à pesquisa.

Referência	**Uso de DHT**	**Contexto de *big data***	**Topologia**	**Tolerância a Falhas**	**Múltiplos Saltos / Salto Único**
(PARK et al., 2010)	sim	sim	anel	sim	Múltiplos Saltos
(GODFREY; SHENKER; STOICA, 2006)	sim	sim	não trata	sim	não trata
(TANG et al., 2005)	sim	sim	não trata	sim	não trata
(ZHANG; GOEL; GOVINDAN, 2005)	sim	sim	anel	não	Múltiplos Saltos
(YALAGANDULA; DAHLIN, 2004)	sim	sim	árvore	não	Múltiplos Saltos
(AMIR; SRINIVASAN; KHAN, 2015)	sim	sim	anel	não	Múltiplos Saltos
(ZHANG et al., 2011)	sim	não	anel	sim	Múltiplos Saltos
(PICCONI; SENS, 2006)	sim	não	anel	sim	não trata
(STEINER; ENNAJJARY; BIERSACK, 2007b)	sim	não	árvore	sim	Múltiplos Saltos
(MORSELLI et al., 2007)	não	sim	não trata	sim	Múltiplos Saltos
(LLOYD; GOKHALE, 2017)	não	sim	não trata	não	não trata

Tabela 3.4: Características das obras obtidas da biblioteca *IEEExplore Digital Library* classificadas como relevantes à pesquisa.

Referência	**Uso de DHT**	**Contexto de *big data***	**Topologia**	**Tolerância a Falhas**	**Múltiplos Saltos / Salto Único**
(PANDEY; AHMED; CHAUDHARY, 2009)	sim	sim	multianel	sim	Múltiplos Saltos
(PUTHUSSERI et al., 2019)	sim	sim	árvore	sim	Múltiplos Saltos
(SUN et al., 2016)	sim	sim	árvore	não	Múltiplos Saltos
(CORTÉS et al., 2015)	sim	sim	árvore	não	não trata
(CORTÉS et al., 2016)	sim	sim	árvore	não	não trata
(MOHAMMED; SHARAF; OMARA, 2014)	sim	sim	árvore	não	não trata
(CHENG et al., 2014)	sim	sim	híbrida - multianel	não	Múltiplos Saltos
(SHRIVASTAVA; KHATANIAR; GOSWAMI, 2007)	sim	sim	multianel	não	Múltiplos Saltos
(GARCES-ERICE et al., 2004)	sim	não	não trata	não	não trata
(SUN et al., 2007)	não	sim	não trata	não	não trata

Como citado na etapa anterior, as características das obras selecionadas mostram informações interessantes que serão utilizadas na continuidade deste trabalho:

A primeira informação obtida das Tabelas 3.3 e 3.4 é quanto as DHTs de salto único, que não são amplamente populares como as de múltiplos saltos, devido a dificuldade de manter as tabelas de roteamento. Outra informação obtida é quanto a popularidade de topologias em anel e árvore, em que não foi registrado soluções que utilizam topologia de hipercubo, por exemplo, que é a topologia que será utilizada neste trabalho. Diante destas informações será possível realizar comparações num cenário de simulação de um sistema distribuído no contexto de *big data* de soluções que utilizam topologia em anel e árvore juntamente com DHTs de múltiplos saltos com a solução proposta que utiliza DHTs de salto único e topologia de hipercubo.

Esta etapa também prevê possíveis falhas nos resultados sugerindo correções nas etapas anteriores. Pelo fato desta pesquisa ter durado mais de seis meses e o executor ser estudante de graduação, diversas correções nas etapas anteriores foram feitas, no decorrer da trabalho. A

questão dos textos duplicados foi uma destas correções e além disso vários textos selecionados foram submetidos novamente a terceira etapa de forma a serem eliminados por terem temas muito amplos que a princípio, no nível de criticidade do executador da RS, foram classificados relevantes, mas com o decorrer da pesquisa e o amadurecimento da criticidade do executador, estes textos foram considerados irrelevantes.

Com o fim desta pesquisa constatou-se que não aborda-se o contexto de sistemas de único salto com a mesma intensidade que os sistemas de múltiplos saltos. Isso pode estar ligado com a dificuldade dos dispositivos de uma rede manter uma tabela de roteamento com n posições, em que n é a quantidade de participantes no sistema, e além disso, entende-se dificuldades em manter esta tabela atualizada num contexto não simulado. Desta indagação entende-se que uma pesquisa, sendo ela uma outra revisão sistemática ou qualquer outra metodologia, seria pertinente para responder perguntas do tipo: Por que não se discute com a mesma intensidade sistemas DHT de múltiplos saltos e sistemas de saltos únicos? Será que pelo fato de ser custoso a manutenção de tabelas de roteamento gigantescas presentes nas DHTs de salto único? O quão viável seria manter uma tabela de roteamento em um sistema distribuído de salto único, levando como métrica tempo de propagação de dados e quantidade de troca de mensagens?

Capítulo 4

VCube DHT

4.1 Modelo do Sistema

Com o objetivo de criar cenários controlados de testes, o sistema em que o VCubeDHT é de fato simulado é composto por *n* nodos conectados num hipercubo virtual, em que os vértices do hipercubo representam os nodos e suas arestas os *links* em que os participantes poderão realizar os testes de detecção de falha. No entanto, todos os nodos podem comunicar com qualquer outro nodo, de forma que este sistema representa um sistema de salto único.

Um vértice é dito como *ocupado* quando conter um *peer* em execução, e falho quando não conter um *peer*, ou seja, não existe vértices ativos sem um *peer* alocado. Desta forma, um nodo *p* ocupa um espaço em que um vértice falho ocupava antes de estar nesse estado. Quando um vértice está ativo, e portanto com um *peer* alocado, ele comunica-se com os outros participantes da rede, respondendo de forma imediata as requisições realizadas a ele, podendo realizar múltiplas tarefas em paralelo caso receba requisições ao mesmo tempo ou quando está atendendo outras. Se o vértice está inativo e, portanto, falho, qualquer requisição feita a ele será ignorada, de forma que o sistema mantem acordado um *timeout* para todas as requisições de no mínimo 3 (três) unidades de tempo. Caso uma requisição exceda este tempo, se for uma mensagem de teste de falha, é detectado um evento (mudança de estado de um vértice) e caso seja outra requisição, o *peer* autor da mesma espera a rodada do algoritmo de detecção de falha terminar para então realizar a requisição novamente.

Um vértice pode falhar em momentos aleatórios, de forma que todas as informações contidas neste vértice são perdidas. Da mesma forma, novos participantes podem entrar na rede em qualquer momento, podendo assim ocupar o espaço dos nodos falhos mas com uma cópia das

informações do *peer* que determinou a posição para o novo participante.

Os testes de detecção de falha são realizados de forma periódica e consiste no envio de mensagens dos participantes ativos para outros nodos, a fim de detectar eventos. Uma rodada de testes consiste no período em que todos os nodos ativos completam suas verificações. Quando dois nodos ativos trocam mensagem de testes, é realizado também troca de informações, controladas por um *timestamp*, de forma a atualizarem suas informações com a de outros participantes da rede. O *timestamp* evita de um nodo considerar uma informação desatualizada. Desta forma os participantes podem detectar um nodo não ativo por meio do teste do mesmo e consequentemente não receber respostas no *timeout* determinado, ou ainda receber as informações de outros nodos que detectaram este evento. Da mesma forma um nodo pode detectar a entrada de um novo nodo se ele for o *peer* que determinou o espaço para o novo participante entrar, ou recebendo informações do nodo que executou esta ação, ou ainda recebendo uma mensagem de teste do novo participante da rede.

O VCubeDHT é dito como completo quando não existe nodos falhos, de forma que a quantidade de arestas são as mesmas de um hipercubo de dimensão $log_2 N$, para *N* processos, arestas adicionais são criadas de forma a descaraterizar um hipercubo completo.

4.2 Particionamento de Chaves

No VCubeDHT as chaves são calculadas através da função SHA-2 de 256 bits, permitindo 2^{256} possíveis chaves. Desta forma, cada nodo de índice *i* recebe uma faixa de chaves para ser responsável, determinado por:

$$[i * 2^{256-d}, ..., (i+1) * 2^{256-d}) \tag{4.1}$$

em que *d* é a dimensão do hipercubo. Neste contexto, em um sistema em que existem 2^{256} vértices alocados, cada vértice será responsável por uma única chave que consiste em seu índice. No cenário em que existam vértices faltantes, cada chave estará sob responsabilidade do nodo com o índice mais próximo.

O algoritmo que determina o *peer* responsável por um vértice *v* é dada pelo Algoritmo 2, retirado de Koppe (2013).

Algoritmo 2: Algoritmo para encontrar *peer* responsável por um vértice *v* (KOPPE, 2013).

```
Entrada: v
Saída: Índice i que corresponde o vértice responsável por v
início
    z = 0
    i = v
    repita
        z++
        i = v XOR z
    até vértice i esteja alocado no sistema;
fim
```

A função do Algoritmo 2 tem como entrada o vértice *v* que pretende-se encontrar o responsável. Desta forma, é inicializada uma variável *z* em zero, e outra *i* com o valor de *v*. O laço presente do algoritmo se repetirá até que *i* represente um vértice ocupado que será o responsável pelo vértice vazio *v*. Para cada repetição do laço, a variável *z* será incrementada representando assim, a distância entre dois vértices. O próximo vértice a ser testado pela condição será o resultado do *ou-exclusivo* entre o identificador *v* e a distância *z*. O resultado desta função será o participante que tem o índice mais próximo do vértice *v*. A condição para determinar se o vértice *i* está alocado no sistema será baseado na visão local que o nodo executante da função tem do sistema, mesmo sendo informações desatualizadas.

Com a formalização do particionamento de chaves no VCubeDHT, é possível descrever os protocolos de busca de chaves, inserções de valores e de entrada de novos participantes na rede.

A Figura 4.1 exemplifica o particionamento de chaves no VCubeDHT. Note que na figura as chaves são divididas em quantidades iguais para cada nodo presente no sistema. Note também que o nodo 3 encontra-se falho enquanto os demais encontram-se ativos. Dessa forma todas as chaves que estaria sob responsabilidade do nodo 3 passam a estar sob responsabilidade do nodo ativo mais próximo que é o nodo 2.

4.3 Protocolo de busca e inserção de valores em salto único

Como descrito anteriormente, todos os nodos mantém uma tabela de roteamento para todos os outros nodos, de forma a manter informações dos estado de cada participante. Esta tabela é atualizada pelo algorítimo de detecção de falhas, e pela troca de informações dos nodos ativos,

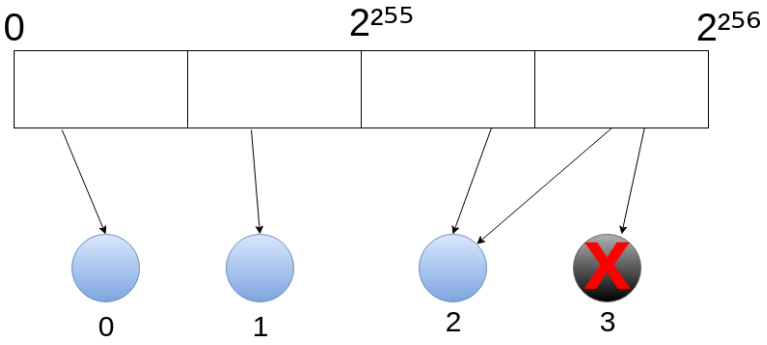

Figura 4.1: Representação do particionamento de chaves do VCubeDHT.

detectando eventos. A precisão desta tabela depende da frequência em que o algoritmo é executado, de forma a influenciar no Algoritmo 2 e, consequentemente, no protocolo de busca e inserção de valores.

Tanto para busca quanto para inserção de valores no sistema, o nodo executante iniciará o processo em busca do responsável de uma determinada chave. O Algoritmo 3, encontrado em Koppe (2013), apresenta os passos que um nodo executa para buscar o responsável desta chave.

Algoritmo 3: Algoritmo *lookup* do VCubeDHT (KOPPE, 2013).

Entrada: k, d
Saída: Índice i que corresponde o vértice que mantém a chave k
início
    v = floor(k / 2 $^{256-d}$)
    i = responsável pelo vértice v
fim

Como entrada o algoritmo *lookup* tem como uma chave k e a dimensão d do sistema. Desta forma, o algoritmo encontra o vértice v responsável pela chave k informada na primeira linha e então o algoritmo retorna o responsável pelo vértice v, operação executada pelo Algoritmo 2.

Para o nodo que estiver executando uma busca por valores, enviará, ao fim do processo, uma mensagem para o nodo i, juntamente com a chave k, requisitando o valor correspondente a essa chave. Caso a resposta desta busca não retornar em um determinado tempo, todo o processo é repetido quando a rodada do algoritmo de detecção de falha finalizar e consequentemente a tabela de roteamento esteja mais atualizada. O mesmo ocorre quando um nodo estiver executando uma inserção de valor na rede, com a diferença de que a mensagem será enviada juntamente com o valor. Além disso ao início do processo, o nodo aplica ao valor a ser inserido a função

SHA-2 de forma a ter como resultado a chave k de 256 bits, podendo assim dar continuidade ao processo.

4.4 Protocolo de Entrada

O objetivo do protocolo de entrada de um participante p no sistema VCubeDHT é posiciona-lo de forma tal a melhorar a distribuição de chaves do sistema, ou seja, o p será responsável por uma faixa de chaves de algum nodo s sobrecarregado frente aos demais nodos. Para este processo, é necessário que p tenha conhecimento de pelo menos um participante q. Além disso, o protocolo deve evitar com que colisões aconteça, ou seja, dois novos participantes ocupem o mesmo posicionamento. Por fim, a informação de que o novo nodo está ativo será propagado a partir da próxima troca de mensagens de q com seus vizinhos, baseado no protocolo do VCube.

Para tanto, o sistema seguirá os seguintes passos no protocolo de entrada:

1. Nodo p requisita entrada para o nodo q.
2. Nodo q verifica qual o nodo r com a maior faixa de chaves do sistema, baseado na visão local do sistema, informado assim para p, juntamente com o índice de um vértice v que é o melhor posição não ocupada para receber a nova alocação.
3. Nodo p requisita um espaço para o nodo r;
4. Nodo r verifica se o vértice v está sem nenhum ocupante, segundo a visão local do sistema.

   se sim, o nodo r divide sua faixa de chaves ao meio, de forma a deixar o nodo p responsável, atualizando assim sua visão do sistema e passando uma cópia para o nodo p, que neste momento passa a estar alocado no vértice v.

   se não, o nodo r se torna o nodo q e então o protocolo retorna ao segundo passo.

Para o nodo q determinar o nodo r com a faixa de chaves do sistema, é executado o Algoritmo 3 para cada nodo não ativo no sistema, ou seja, ao seguir o algoritmo, k receberá o índice do nodo não ativo selecionado. Dessa forma, é possível fazer a contagem de qual o nodo ativo é responsável por mais índices de nodos não ativos e, consequentemente, por suas chaves no sistema.

No passo 4, garante-se que dois nodos não ocuparão o mesmo posicionamento, pois apenas o nodo responsável pela faixa a ser dividida é que irá de fato, realizar a divisão. Caso a visão local do sistema do nodo q esteja desatualizada, ou diferente da visão do nodo r, o protocolo será realizado novamente depois do fim da rodada de teste do protocolo de detecção de falhas.

A Figura 4.2 a representa o protocolo de entrada no VCubeDHT. Note que um nodo p faz uma requisição de entrada para um nodo conhecido, que no caso é o nodo 0. Baseado na sua visão local do sistema o nodo 0 responde o endereço do nodo mais carregado de chaves do sistema. Este exemplo baseia-se na Figura 4.1, na qual o nodo 3 está livre e o espaço de chaves correspondente está sob responsabilidade do nodo 2. Continuando com o protocolo, o nodo p faz uma nova requisição, agora para o nodo 2 que por sua vez confirma que é o nodo mais carregado de chaves do sistema e responde para p o endereço que o mesmo deve se alocar, que é no vértice 3. Além disso, no passo 4 da figura o nodo 2 envia uma cópia da visão local que tem do sistema.

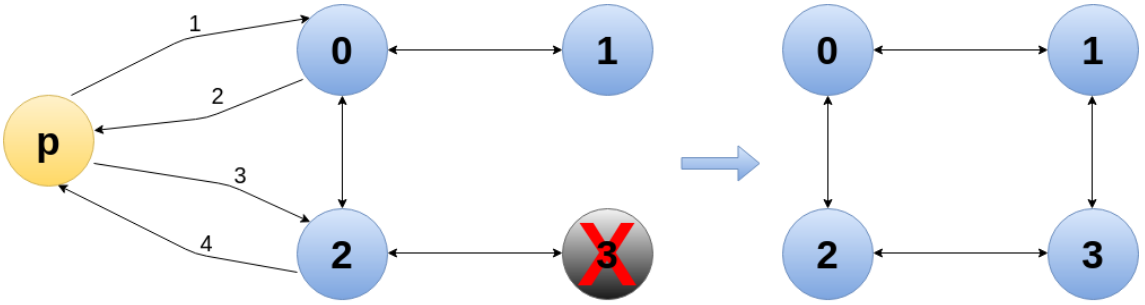

Figura 4.2: Representação do Protocolo de Entrada no VCubeDHT.

4.5 Tratamento de Saída

Como dito anteriormente, um nodo pode sair do sistema a qualquer momento no contexto do VCubeDHT. Esta saída pode ser espontânea ou por motivo de falha, mas os protocolos do sistema encararão o evento como uma falha, ou seja, não existe aviso prévio de saída de participante do sistema mesmo que seja espontânea. A propagação da informação para sistema dependerá do protocolo de detecção de falha, ou seja, um nodo p tem como visão local que q está ativo, mas quando testar o mesmo e não obtiver resposta em uma determinada quantidade de unidade de tempo será detectado a falha e nos próximo testes de falha com nodos ativos a informação será repassada.

Além disso, quando o nodo q deixa o sistema e um nodo r ocupa seu espaço, posteriormente, todas as informações de q serão perdidas de forma que r receberá uma cópia da visão local do sistema do nodo que ativo r. Por fim, no contexto do VCubeDHT, um nodo ou está ativo com um *peer*, ou está falho, não existe nodos ativos mas sem *peer* alocado.

Capítulo 5

Resultado e Discussões

No presente capítulo são descritos os testes realizados com o sistema VCubeDHT, bem como ao ChordDHT de forma a apresentar as vantagens e desvantagens da utilização do sistema abordado pelo presente trabalho em relação a quantidade de saltos e tempo necessário para realizar processos simples como o protocolo de entrada e saída de dispositivos e o protocolo de recuperação e armazenamento de dados no sistema distribuído. Para a implementação utilizou-se a linguagem de programação Java e a ferramenta de simulação de sistemas P2P *Peersim* (MONTRESOR; JELASITY, 2009).

O PeerSim é um simulador de sistemas distribuídos de código aberto escrito em Java. Ele é composto por um mecanismo simples baseado em ciclos e outro mecanismo acionado por eventos, permitindo realizar testes simplificados e, por parte do usuário, dar mais foco na implementação do algoritmo do que em protocolos de comunicação numa rede genérica. O PeerSim foi desenvolvido pela Biology-Inspired Techniques (2006) e pela Evolving (2004).

Para cada sistema testado foram construídos 4 cenários, de forma que cada um foi executado em redes de 8 a 16384 nodos variando em potência de 2. O objetivo de cada cenário varia entre apresentar tempo de execução do protocolo de entrada e do protocolo de recuperação e armazenamento de dados, quantidade de testes do algoritmo de diagnóstico distribuído e quantidade de saltos refeitos decorrente da visão incorreta do sistema por parte de um determinado participante. Para manter um grau de confiança dos testes, os números de cada tamanho de rede para cada cenário corresponde à média de dez execuções. Além disso, aplicou-se uma fórmula de grau de confiança de 95% em todas as variáveis de forma a identificar o intervalo de erro das mesmas.

O primeiro cenário apresenta o comportamento de uma rede ao iniciar com apenas um par-

ticipante e de forma periódica outros participantes passam pelo protocolo de entrada até que todos os nodos do sistema estejam ocupados. O segundo cenário objetiva-se em observar o comportamento de um sistema sem dispositivos inativos em que todos os participantes enviam um total de 1 milhão de mensagens. O terceiro cenário tem o mesmo objetivo do segundo com a diferença de que um evento de saída é simulada no primeiro instante da simulação. No quarto cenário, o sistema inicia com metade dos nodos inativos de forma a todos os participantes ativos realizam um total de 1 milhão de mensagens sendo que protocolos de entrada e eventos de saída são simulados de forma aleatória no decorrer da simulação.

Todos os nodos tem um *link* de salto único para qualquer outro participante da rede. Dessa forma, qualquer salto leva uma unidade de tempo, não existindo variação de velocidade de comunicação. Para um salto para um dispositivo falho, o *timeout* é finalizado na terceira unidade de tempo.

Uma última consideração antes do detalhamento dos cenários, é sobre o sistema simulado ChordDHT. Optou-se em adaptá-lo para um sistema de salto único de forma que a *fingertable* (que aqui pode ser chamado de *timestamp* para igualar o termo com o VCubeDHT) tem o tamanho n, em que n é a quantidade de participantes na rede. Isto quer dizer que todos os nodos armazenam uma visão local de todo o sistema. Desta forma, os protocolos de *Lookup*, *Put* e entrada de novo participante da rede são idênticos em ambos os sistemas comparados, diferenciando-se apenas no protocolo de detecção de falhas. Esta decisão teve como objetivo deixar ambos os sistemas mais similares, evitando resultados muito discrepantes em que uma comparação entre um sistema de salto único e um sistema de múltiplos saltos apresentariam ao utilizar métricas como tempo de execução, tempo de detecção de falhas e quantidade de saltos necessário para realização dos protocolo.

No decorrer deste capítulo quatro seções são organizadas para apresentar os cenários, apresentando seus detalhes, desenvolvimento e comparações entre o VCubeDHT e ChordDHT, juntamente com suas discussões. No fim, uma quinta seção apresenta considerações finais sobre a execução dos cenários.

5.1 Cenário 1

O Cenário 1 foi descrito com o objetivo de comparar a quantidade de tempo que ambos os sistemas levam para preencher todo o sistema a partir de apenas um nodo ativo. Dessa forma, determinou-se que cada rodada do algoritmo de diagnóstico distribuído inicia a cada 100 unidades de tempo e cada entrada de dispositivo a cada 150 unidade de tempos. Estes valores foram escolhidos para que o sistema tenha um comportamento mais regular, ou seja, para que haja tempo dos algoritmos detectarem os eventos antes que um novo protocolo de entrada seja iniciado, evitando assim a realização do processo com visões muito desatualizadas do sistema e consequentemente o alto número de saltos refeitos ou tentativa de ativação de um vértice já ocupado.

Um salto refeito no protocolo de entrada ocorre quando as requisições do possível novo nodo p tem respostas diferentes entre os nodos já participantes da rede, de forma que p deverá realizar mais consultas do que os passos descritos no protocolo de entrada no capítulo anterior. Para exemplificar, considere que p requisite a entrada para um nodo q, e q determina que o melhor vértice para a entrada de p é o vértice v e o mesmo está sob responsabilidade de r. Dessa forma p irá realizar uma requisição de entrada para r. Caso r considere que v já está ocupado, o protocolo de entrada deverá ser realizado novamente, de forma que r passa a ser o q. Neste instante a variável de "salto refeito"é incrementado nas simulações com o protocolo de entrada. Esse fenômeno ocorre pois o *timestamp* de p não estava atualizado, ou seja, não tinha a informação de que v já estava ocupado.

Seguindo no exemplo anterior, considere que r concorde com q em que o vértice v é ideal para ocupar o nodo p. Neste caso p irá se alocar na posição de v. Caso neste instante p detecte que v já esta ativado, ou seja, já exista um nodo s nesta posição, a variável de "ativação refeita"é incrementada. Segundo os protocolos do sistema deste caso, o nodo p deverá reiniciar o protocolo de entrada, de forma que os nodos do sistema tendem a estarem com seus *timestamp* mais atualizados, resultando em menos saltos perdidos, decorrente tanto de tentativa de ativações de vértices já ocupados, quanto de saltos refeitos.

Cada teste de falha realizado por ambos os algoritmos determina todos os nodos que serão testados na rodada de forma a escalonar cada teste a cada unidade de tempo seguinte. O nodo p que inicia o cenário com status ativo, tem seu *timestamp* atualizado com a realidade, de forma a

se atualizar a cada detecção de eventos. Note que na primeira rodada p é responsável por todos os vértices do sistema, ou seja, a primeira rodada de testes dura n unidades de tempo, em que n é o tamanho da rede.

A Figura 5.1 apresenta uma comparação da média do tempo de execução entre os sistemas VCubeDHT e ChordDHT. Em paralelo, a Figura 5.2 mostra a quantidade de rodadas de testes do algoritmo de detecção de falha de ambos os sistemas. Note que existe uma proporção entre a quantidade de testes do algoritmo de detecção de falhas e o tempo de execução, visto que cada rodada de teste é realizada a cada 100 unidades de tempo.

Nota-se que tanto o tempo de execução quanto a quantidade de teste do algoritmo de diagnóstico distribuído crescem em função do tamanho da rede e são bastante parecidos entre os sistemas, embora o VCubeDHT tenha uma ligeira economia de tempo e rodadas de testes. Esta economia não apresenta maior vantagem pelo fato da distância de tempo entre a entrada de cada dispositivo ser suficiente para o algoritmo de diagnóstico distribuído atualizar a visão parcial de todos os participantes do sistema.

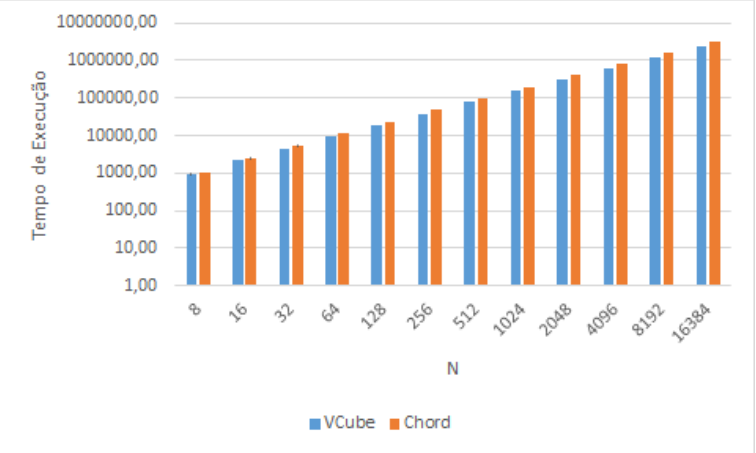

Figura 5.1: Tempo de execução no Cenário 1.

Diferentemente das variáveis da Figura 5.1 e 5.2, a variável de quantidade de saltos refeitos apresenta bastante discrepância entre os sistemas. A Figura 5.3 apresenta a quantidade de saltos refeitos em ambos os sistemas. Note que somente o VCubeDHT apresenta valores diferentes de 0. A variável de saltos refeito é incrementada sempre quando existe discrepância de visões locais entre os nodos participantes de um processo do protocolo de entrada, descrito com mais detalhes anteriormente.

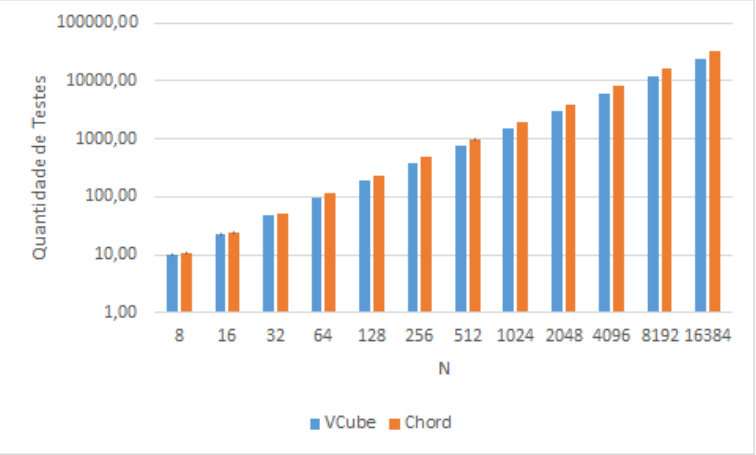

Figura 5.2: Quantidade de testes no Cenário 1.

Este fenômeno combinado com os números do gráfico da Figura 5.4, em que indica que o VCubeDHT realiza quase 1000 vezes menos tentativa de ativações de vértices já ocupados por algum dispositivo do que o ChordDHT, mostra que, embora o VCubeDHT tenha mais discrepâncias de visões locais entre os participantes, o ChordDHT apresenta mais uniformidades de visões incorretas. Além disso, se somar a quantidade de saltos refeitos para encontrar uma melhor posição para alocar um novo nodo com a quantidade de tentativa de ativações de vértices já ocupadas, que consequentemente gera mais um salto, o VCubeDHT continuaria com um número muito inferior de saltos adicionais dos saltos gastos pelo ChordDHT.

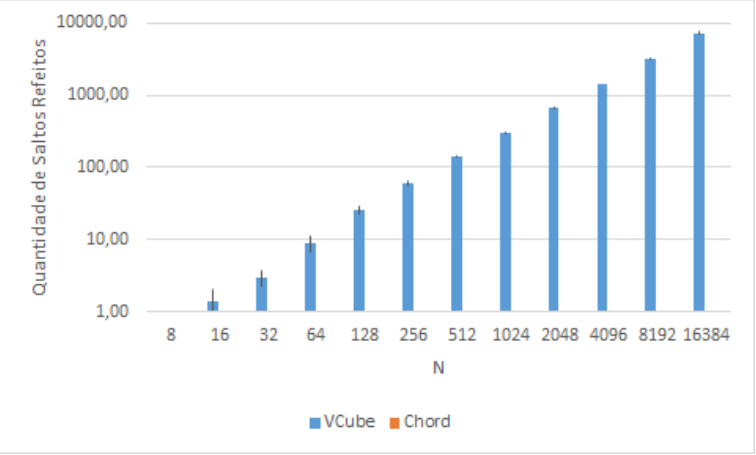

Figura 5.3: Quantidade de Saltos refeitos no VCubeDHT no Cenário 1.

Ao aplicar o grau de confiança de 95% nas 4 variáveis apresentadas no Cenário 1, nota-se que o intervalo de erro apresenta valores desprezíveis para as escalas logarítmicas.

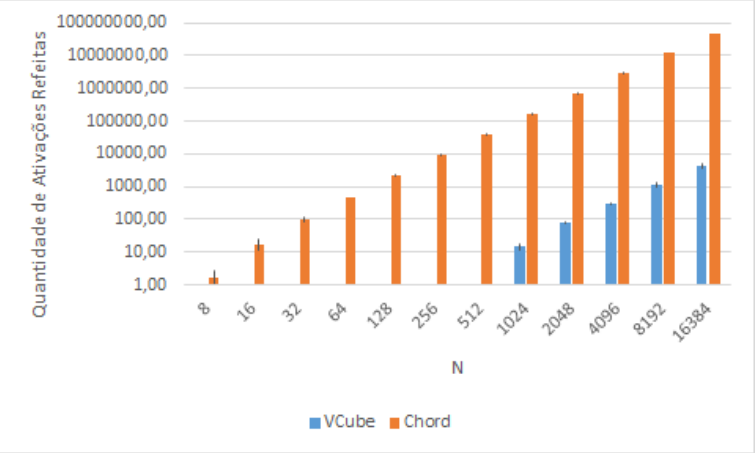

Figura 5.4: Comparação de quantidade de ativações de vértices refeitas no Cenário 1.

5.2 Cenário 2

O Cenário 2 tem como objetivo analisar quanto tempo leva para um milhão de mensagens sejam entregues. Estas mensagens são distribuídas igualmente a todos os participantes da rede e escalonadas a cada uma unidade de tempo. O cenário também determina que todos os dispositivos estarão ativos de forma a não existir eventos de entrada, nem saída e nem consultas que demandarão de saltos adicionais. Além disso todos os nodos iniciam com o *timestamp* atualizado e a cada cem unidades de tempo uma rodada de teste do algoritmo de diagnóstico distribuído é realizado. Cada mensagem corresponde a um evento de *Put* ou *Lookup*, visto que o tempo para a realização de ambas as operações é o mesmo, não tendo assim, a necessidade de diferenciá-los.

Visto que o presente cenário não apresenta nodos não ativos ou saltos refeitos decorrentes a visões locais desatualizadas do sistema, a média de tempo para realizar um evento de *Put* ou *Lookup* sempre será duas unidades de tempo, ou seja, se a mensagem é enviada no tempo 0, por exemplo, o destinatário recebe a mesma no tempo 1, retornando a requisição ao mesmo tempo e chegando a origem no tempo 2.

A Figura 5.5 apresenta, portanto, o tempo médio de execução (dado como empate) de ambos os sistemas em todos os tamanhos de rede testados. Da mesma forma, a Figura 5.6 mostra a quantidade de testes de ambos os sistemas. Percebe-se que o tempo de execução (medido em unidades de tempo) tende a ser cem vezes maior do que o valor da quantidade de rodadas de testes (medido em unidades) do algoritmo de diagnóstico distribuído, visto que o cenário

determina uma rodada a cada cem unidades de tempo. Além disso, como são um milhão de mensagens a serem distribuídas para todos os participantes, em redes menores o tempo necessário para concluir o envio de todas as mensagens é maior, visto que são mais mensagens para cada dispositivo. Consequentemente, para redes maiores este tempo tende a ser menor.

Para ilustrar este fenômeno, na rede com oito participantes em que existe 1 milhão de mensagens para serem entregues, cada nodo fica responsável por quantidades iguais de mensagens, que neste caso é 125 mil mensagens. Assim, visto que o tempo médio de entrega da mensagem é 2 unidades de tempo, será necessário um poco mais do que 125 mil unidades de tempo para todas as mensagens serem entregues, ou seja, no instante 125 mil, a última mensagem de cada nodo finalizará o envio, de forma a aguardar a próxima unidade de tempo para receber a confirmação e então finalizar o processo. No caso de uma rede com 16 mil e 384 participantes foram distribuídos entre 61 e 62 mensagens para cada dispositivo, necessitando de 63 unidades de tempo para concluir todas as entregas.

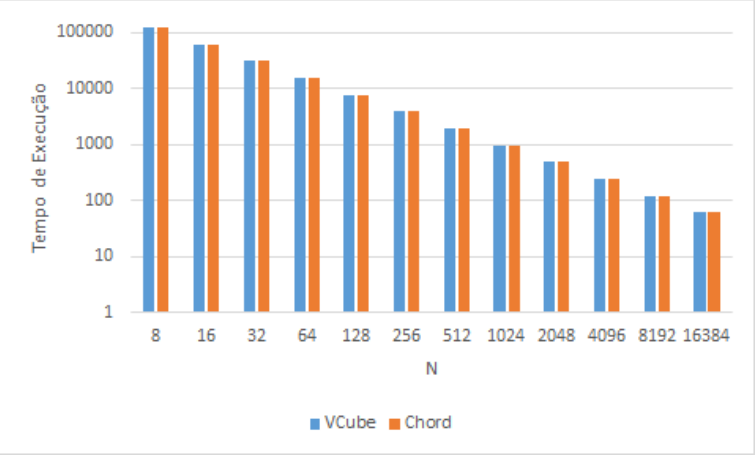

Figura 5.5: Tempo de execução do Cenário 2.

Ao analisar o grau de confiança de 95% do tempo de execução e quantidade de testes do algoritmo de diagnóstico distribuído no Cenário 2 tem-se que o intervalo de erro é desprezível para todos os tamanhos de rede.

5.3 Cenário 3

O Cenário 3 é uma cópia do segundo cenário com uma única diferença: um participante torna-se inativo no primeiro instante da simulação. Dessa forma é possível capturar o tempo

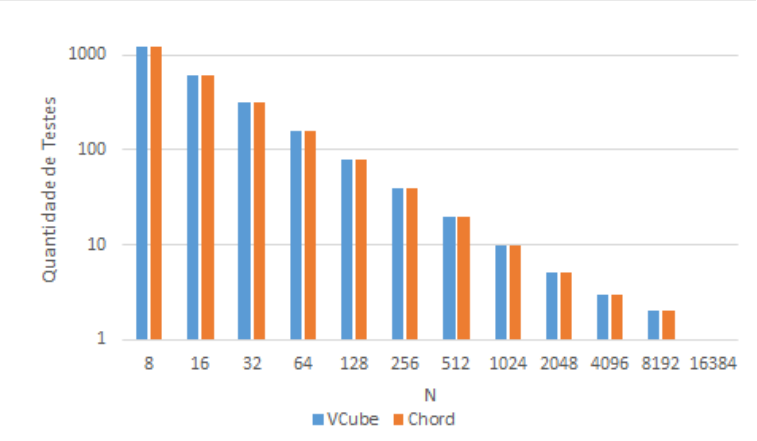

Figura 5.6: Quantidade de testes do Cenário 2.

que o sistema leva para que todos os nodos detectem este evento. Além disso, as mensagens que foram atribuídas ao nodo falho são redistribuídas para os demais nodos ativos.

Nota-se na Figura 5.7 e Figura 5.8 a mesma proporcionalidade entre as variáveis de tempo de execução e quantidade de testes do algoritmo de diagnóstico distribuídos presentes nos demais cenários. Além disso, o cenário executado pelo VCubeDHT apresenta exatamente o mesmo comportamento do Cenário 2, ou seja, o tempo de execução está em função das 1 milhões de mensagens distribuídas entre os dispositivos ativos.

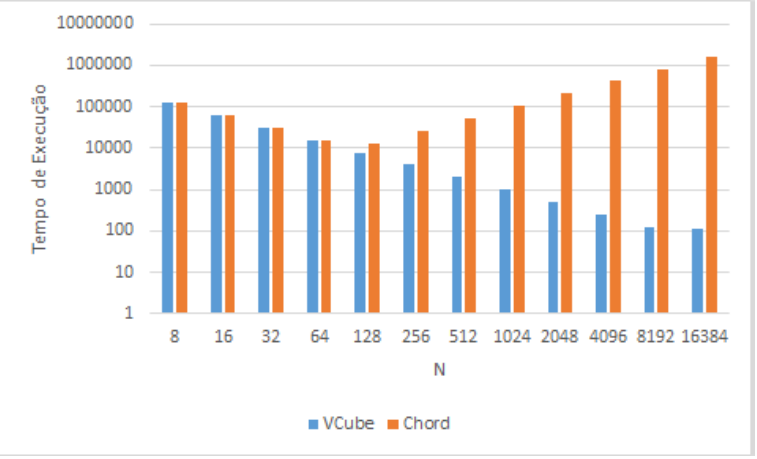

Figura 5.7: Tempo de execução do Cenário 3.

A particularidade deste cenário encontra-se nas redes a partir de 128 nodos executado pelo ChordDHT, em que apresenta um comportamento inesperado. Em vez do tempo de execução diminuir exponencialmente, segundo o padrão iniciado em redes de 8 nodos, o mesmo aumenta de forma exponencial. Este comportamento ocorre pelo fato de que para redes de até 64 nodos

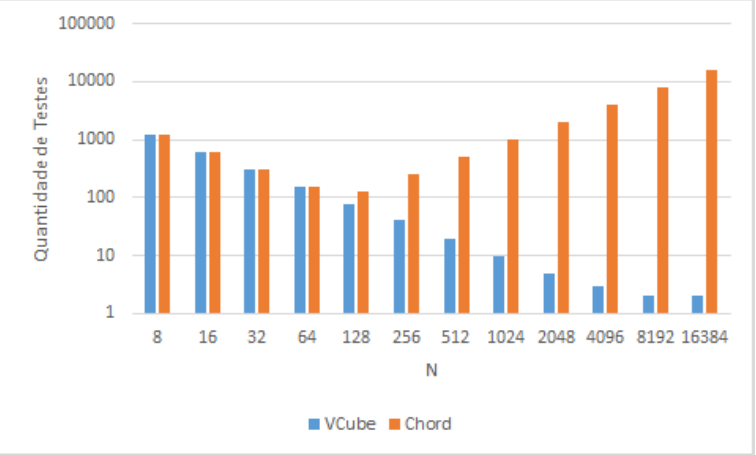

Figura 5.8: Quantidade de testes do Cenário 3.

no ChordDHT, o variável que mantem a simulação em execução é o envio das 1 milhão de mensagens, ou seja, a condição de parada da simulação é a conclusão do envio das mensagens. Para redes com 128 ou mais nodos, o tempo de diagnóstico do nodo inativo tem maior latência do que o envio das mensagens, ou seja, neste caso a condição de parada acaba sendo o tempo de diagnóstico por toda a rede do evento ocorrido no primeiro instante de tempo. Para reforçar este argumento note que o tempo de execução para redes superiores a 128 nodos no ChordDHT tem o mesmo valor do que o tempo de diagnóstico do mesmo tamanho de rede apresentado na Figura 5.9.

Igualmente ao Cenário 2, o intervalo de erro do grau de confiança de 95% do tempo de execução e a quantidade de testes do Cenário 3 é desprezível.

Ao analisar o tempo de diagnóstico da falha em ambos os sistemas, apresentado na Figura 5.9, observa-se que o VCubeDHT tende a manter uma constante próxima a cem unidades de tempo, ou seja, o diagnóstico da falha é propagado para todo o sistema entre o primeiro e segundo intervalo de teste do VCube, visto que o VCube realiza $log_2 N$ testes em cada intervalo. Note que o intervalo de erro, para o grau de confiança de 95%, varia, em todos os tamanhos de rede, no máximo em 40 unidades de tempo para mais ou para menos da constante próxima a 100 unidades de tempo. Já o ChordDHT, como esperado de um algoritmo de diagnóstico distribuído em anel, apresenta um crescimento exponencial em função do tamanho da rede. Quanto ao intervalo de confiança, na visão de um gráfico em escala logarítmica de base 10, torna-se desprezível.

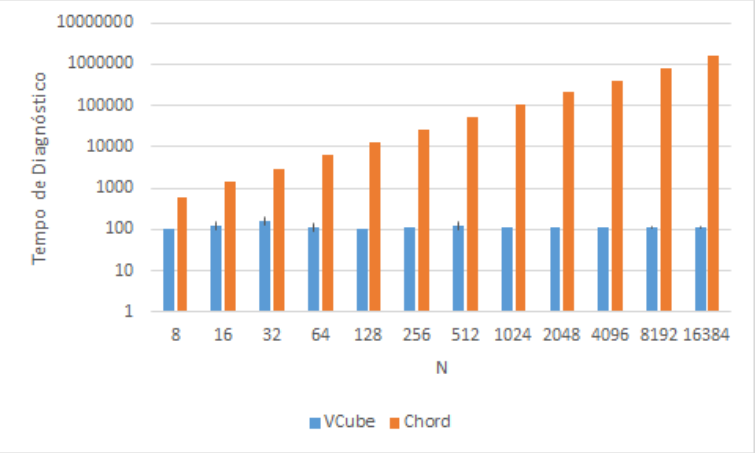

Figura 5.9: Tempo para diagnóstico de falha no Cenário 3.

O gráfico da Figura 5.10, apresenta o total de saltos refeitos para realizar de todos os *puts* e *lookups* da simulação. Diferentemente do Cenário 0, a variável de saltos refeitos tanto no cenário 3 quanto no cenário 4, é incrementada sempre quando uma requisição a uma chave é feita para um nodo que não é responsável pelo mesmo. Note que a linha de crescimento dos saltos refeitos no ChordDHT, presente na Figura 5.10, continua sendo exponencial. Já o VCubeDHT apresenta-se bem comportado, com uma linha sem crescimento, com valores próximos a 100 unidades de tempo e com o intervalo de no máximo 17 unidades de tempo.

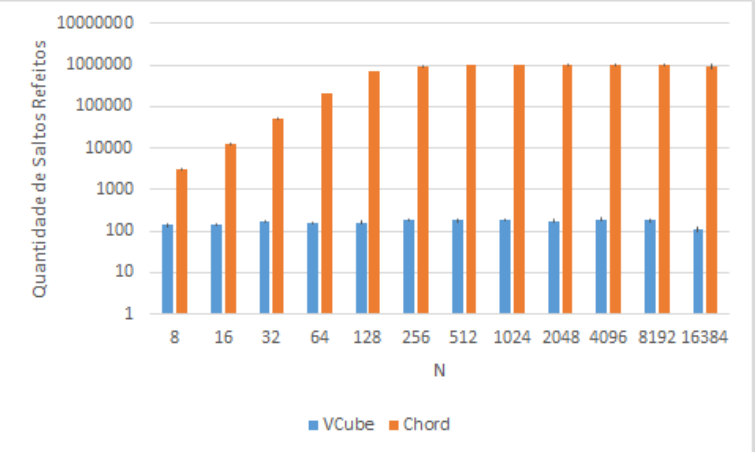

Figura 5.10: Quantidade de saltos refeitos no Cenário 3.

5.4 Cenário 4

O Cenário 4 tem como objetivo simular uma situação mais próxima da realidade, ou seja, pretende-se observar o comportamento de nodos saindo e entrando na rede de forma aleatória. Para isto, o sistema inicia com metade dos nodos falhos de forma que os nodos não falhos tem todas as posições do *timestamp* informando que todos os demais nodos estão ativos. Isto irá resultar num cenário inicialmente cheio de saltos perdidos, até a detecção de todos os nodos não ativos e então a estabilização do sistema. Além disso, os testes do algoritmo de diagnóstico continuam a cada cem unidades de tempo, e um evento é realizado a cada cem unidades de tempo, ou seja, aleatoriza-se um evento de entrada ou saída do sistema. Por fim, um milhão mensagens são executadas, distribuídas igualmente aos nodos ativos. Caso um nodo acabe tornando-se inativo antes de terminar todas as mensagens, a mesma é redistribuída para outro nodo ativo.

O gráfico da Figura 5.11 apresenta o tempo de execução do cenário e a Figura 5.12 a quantidade de testes realizados em ambos os sistemas. Note que o comportamento é semelhante ao do Cenário 3, apresentando a mesma anomalia com redes maiores do que 128 nodos. O que não mantém-se igual ao Cenário 3 é o intervalo de erro de ambas as variáveis das duas soluções, que aqui apresentam valores mais expressivos. Mesmo assim, a curvatura dos gráficos mostram-se confiáveis pelo fato que os intervalos de erro de um determinado tamanho de rede não ultrapassa os limites dos intervalos de erro dos tamanhos de redes vizinhos.

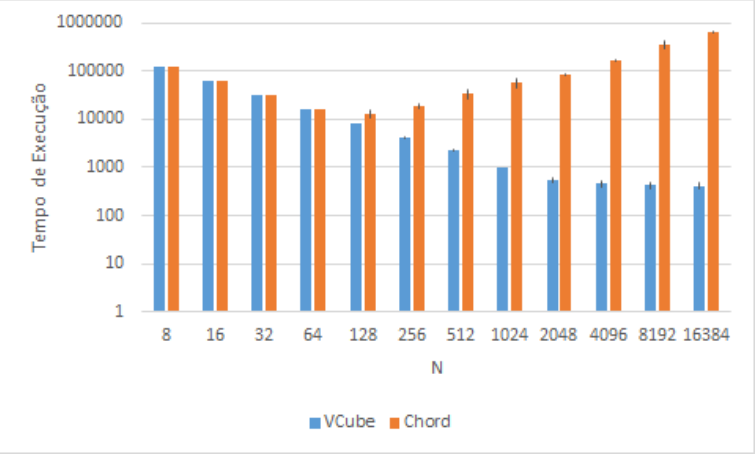

Figura 5.11: Tempo de execução no Cenário 4.

Observa-se no gráfico da Figura 5.13 o tempo médio de entrega das mensagens no Cenário

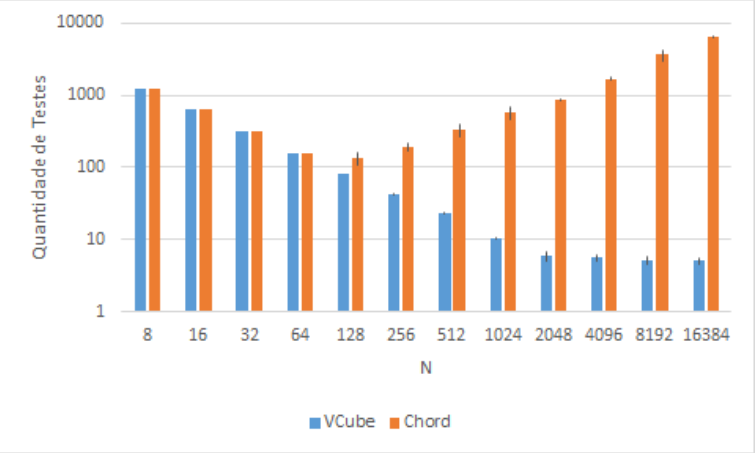

Figura 5.12: Quantidade de testes no Cenário 4.

4. Este é o único cenário em que a média de entrega de mensagens não é igual a 2 unidades de tempo, visto que, em ambos os cenários os vetores *timestamps* tendem a estarem mais desatualizados, e tendo como consequência, maior tempo para entrega das mensagens. Esta informação é reforçada com o comportamento do gráfico da Figura 5.14, em que apresenta os saltos refeitos em ambos os sistemas durante a execução do Cenário 4. Note que o gráfico cresce de forma exponencial, menos agressivamente no VCubeDHT, ou seja, com os *timestamps* tendendo a estarem mais desatualizados o número de saltos presentes em um processo de *lookup* ou *put* aumenta, e em consequência a quantidade de unidades de tempo para completar o processo também aumenta.

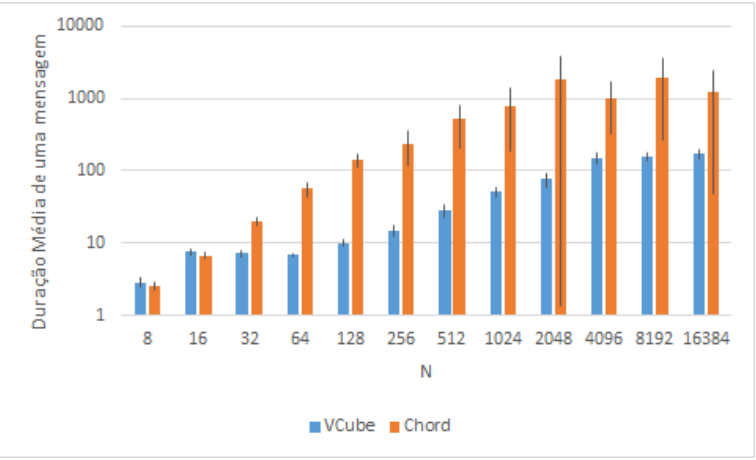

Figura 5.13: Duração média de uma mensagem no Cenário 4.

Quanto ao intervalo de erro presente no gráfico da Figura 5.14, nota-se o mesmo fenômeno

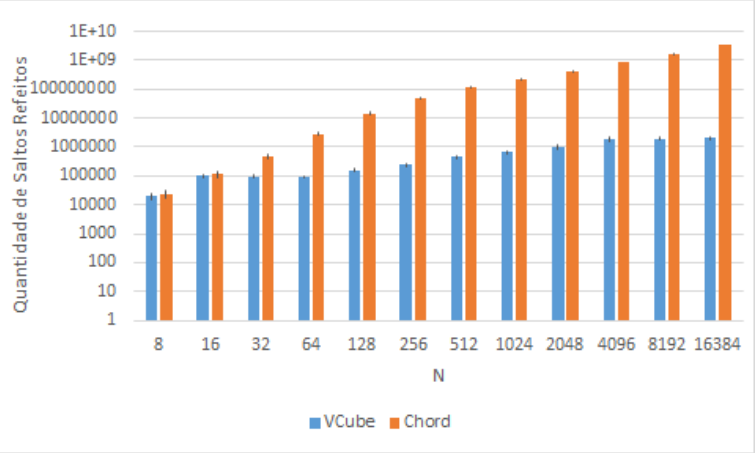

Figura 5.14: Quantidade de saltos refeitos no Cenário 4.

das variáveis de tempo de execução e quantidade de testes do Cenário 4, em que o intervalo acaba atingindo valores maiores, até pelo fato das unidades atingirem números bastante expressivos. Ao analisar o intervalo de erro presente na duração média de cada mensagem, no ChordDHT, nota-se intervalos bastante expressivos, principalmente em redes maiores que 256 nodos. Note que o intervalo de erro para redes com 2048 nodos ultrapassa a própria média da duração média da mensagem. Além disso a curvatura do gráfico mostra fenômenos como o que ocorre com redes de 4096 nodos, em que a duração da mensagem é menor do que redes de 2048 nodos.

Esta série de fenômenos são consequência de um cenário com entradas bastante aleatórias, ou seja, cada repetição de testes para cada tamanho de rede podem ter quantidade de nodos falhos diferentes e quantidade de entrada de participantes diferentes. Isto gera resultados resultados muito divergentes influenciando diretamente na duração das mensagens, visto que a eficiência dos algoritmos de diagnósticos distribuídos tendem a serem diferentes para cada teste.

5.5 Considerações Finais

Com base nas discussões levantadas neste capítulo é possível observar vantagens do sistema VCubeDHT em relação ao ChordDHT levando em consideração quantidade de saltos e tempo de execução dos diversos protocolos que compunham cada sistema. É possível que o leitor note empates técnicos, principalmente nos dois primeiros cenários, entre os sistemas em que as variáveis de tempo de execução e quantidade de rodadas de testes são apresentadas. Este

fenômeno tende a repetir nos dois últimos cenários até o momento em que a duração média de uma mensagem no ChordDHT passa a influenciar no tempo de execução dos cenários, e então a diferença entre os sistemas é perceptível. Quanto as demais variáveis discutidas não resta dúvidas ou fenômenos que possam confundir a diferença de desempenho dos sistemas.

Capítulo 6

Conclusões e Trabalhos Futuros

Tabelas Hash Distribuídas (DHT) são aplicações que permitem armazenar, distribuir e localizar recursos em um ambiente distribuído. Os recursos são organizados a partir de uma tupla <*chave, valor*>, de forma a indexar os recursos disponíveis e o nodo responsável pelo mesmo, e assim permitindo economia de processamento em cenário de sistemas com grande quantidade de recursos.

Considerando que os nodos dos sistemas distribuídos estão sujeitos a falhas, os recursos podem passar a ter novos responsáveis. Para evitar buscas por recursos em endereços desatualizados, as tabelas *hash* devem manter-se o mais atualizadas possível. Dessa forma, utiliza-se de algoritmos de diagnóstico distribuído, ou seja, um atuador faz determinadas varreduras no sistema a fim de detectar possíveis nodos falhos.

Existem dois tipos de DHTs, as de salto único e a de múltiplos saltos. A de salto único, cada participante da rede mantem uma tabela indexando todos os recursos e todos os participantes da rede, de forma que cada pesquisa de recurso é executada em apenas uma mensagem. Os nodos das DHTs de múltiplos saltos mantem tabelas menores com índices de poucos responsáveis. Dessa forma, uma pesquisa por recurso necessita de sub-pesquisas nas tabelas de nodos conhecidos caracterizando o múltiplo salto.

A primeira contribuição do presente trabalho foi a elaboração de uma revisão sistemática a fim de investigar a eficiência do uso de DHTs em ambientes distribuídos com grande quantidade de dados. Como conclusão, constatou-se a popularidade de DHTs de múltiplos saltos, possivelmente pelo fato de ser custoso a manutenção de uma tabelas gigantescas. Outra informação que a revisão sistemática trouxe foi a popularidade de algoritmos com topologia de anel e árvore, em especial os algoritmos Chord (STOICA et al., 2001) e Kademlia (MAYMOUNKOV;

MAZIÈRES, 2002).

A segunda etapa do trabalho, deu-se pela complementação do sistema VCubeDHT, de forma a formalizar protocolos para entrada e saída de nodos na rede e protocolo de busca e inserção de recursos. Com o objetivo de comparar a eficiência em diferentes cenários da solução, implementou-se o ChordDHT. Assim quatro cenários foram avaliados.

No cenário 1 o VCubeDHT se destacou em apresentar menos tentativas de ativações de nodos já ativos, o que mostrou o quão o algoritmo de diagnóstico distribuído mantem as tabelas hashs mais atualizadas. Isso por que o VCube apresenta latência de diagnóstico de $log_2^2 N$, em quanto o Chord é de N^2, em que *N* é o tamanho da rede.

O Cenário 2 apresentou valores idênticos em ambas as soluções, visto que o cenário não contém falhas e então a latência das soluções em cenários de falha não puderam ser capturadas.

O Cenário 3 mostrou que a alta latência de diagnóstico de uma falha no ChordDHT influencia diretamente no tempo de execução do mesmo. Neste cenário o VCubeDHT mostrou ter latências muito parecidas no diagnóstico de uma falha independentemente do tamanho da rede, em quanto o ChordDHT tem um crescimento em função do tamanho da rede.

O cenário 4 mostrou que ambas as soluções apresentam um crescimento na latência de envio de uma mensagem e da quantidade de saltos refeitos pelas mesmas. A diferença é que o VCubeDHT mostra-se um crescimento muito mais lento e o ChordDHT apresenta um crescimento exponencial.

Com estas constatações, conclui-se que o VCubeDHT é mais eficiente em vários aspectos que o ChordDHT. Além disso entende-se que a complementação dos protocolos presentes no VCubeDHT apresentam contribuições importantes para a comunidade científica. Sugere-se, para trabalhos futuros, a realização dos cenários novamente manipulando os parâmetros de entradas a fim de identificar novos fenômenos da solução. Sugere-se também encontrar alternativas para o VCubeDHT trabalhar como uma DHT de múltiplos saltos, de acordo com as conclusões da revisão sistemática, e então comparar a eficiência em diferentes cenários com a solução de salto único. Por fim, propõem-se uma implementação do VCubeDHT em sua totalidade de forma a executar testes analisando quantidade de saltos e aspectos de disponibilidade e replicação de dados, podendo assim até a realizar testes em cenários reais, como a implantação em bancos de dados distribuídos e outros serviços.

Referências Bibliográficas

AKBARINIA, R.; PACITTI, E.; VALDURIEZ, P. Data currency in replicated dhts. In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2007. (SIGMOD '07), p. 211–222. ISBN 978-1-59593-686-8. Disponível em: <http://doi.acm.org/10.1145/1247480.1247506>.

AMIR, A.; SRINIVASAN, B.; KHAN, A. I. A communication-efficient distributed algorithm for large-scale classification within p2p networks. In: ACM. *Proceedings of the Sixth International Symposium on Information and Communication Technology*. [S.l.], 2015. p. 75–82.

BARREIRO NETO, A.; RODRIGUES, L. A.; DUARTE Jr., E. P. Replicação de dados no vcube baseada em dht de salto único. In: *Anais do XVIII Workshop de Testes e Tolerancia a Falhas (WTF - SBRC 2018)*. Porto Alegre, RS, Brasil: SBC, 2017. v. 18. ISSN 2595-2684. Disponível em: <https://portaldeconteudo.sbc.org.br/index.php/wtf/article/view/2563>.

BIANCHINI Jr., R.; BUSKENS, R. An adaptive distributed system-level diagnosis algorithm and its implementation. p. 222 – 229, 07 1991.

BIOLCHINI, J. et al. Systematic review in software engineering. *System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES*, v. 679, n. 05, p. 45, 2005.

BIOLOGY-INSPIRED TECHNIQUES, f. S.-O. i. d. N. 2006. Disponível em: <http://www.cs.unibo.it/bison/>.

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

BRERETON, P. et al. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, Elsevier, v. 80, n. 4, p. 571–583, 2007.

CAVE, A. *What Will We Do When The World's Data Hits 163 Zettabytes In 2025?* 2018. Disponível em: <https://www.forbes.com/sites/andrewcave/2017/04/13/what-will-we-do-when-the-worlds-data-hits-163-zettabytes-in-2025>.

CHENG, W. et al. Stratified multi-ring distributed search model for big data. In: IEEE. *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*. [S.l.], 2014. p. 356–360.

CORTÉS, R. et al. Geotrie: A scalable architecture for location-temporal range queries over massive geotagged data sets. In: IEEE. *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*. [S.l.], 2016. p. 10–17.

CORTÉS, R. et al. A scalable architecture for spatio-temporal range queries over big location data. In: IEEE. *Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on*. [S.l.], 2015. p. 159–166.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Sistemas Distribuídos - 4ed: Conceitos e Projeto*. BOOKMAN COMPANHIA ED, 2007. ISBN 9788560031498. Disponível em: <https://books.google.com.br/books?id=KSZ1rIRWmUoC>.

DUARTE Jr., E. et al. Hypergrid: Arquitetura de grade baseada em hipercubo virtual. 05 2005.

DUARTE Jr., E. P. et al. A hierarquical distributed fault diagnosis algorithm based on clusters with detours. In: *2009 Latin American Network Operations and Management Symposium*. [S.l.: s.n.], 2009. p. 1–6.

DUARTE Jr., E. P.; BONA, L. C.; RUOSO, V. K. Vcube: A provably scalable distributed diagnosis algorithm. In: IEEE PRESS. *Proceedings of the 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. [S.l.], 2014. p. 17–22.

DUARTE Jr., E. P.; NANYA, T. A hierarchical adaptive distributed system-level diagnosis algorithm. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 47, n. 1, p. 34–45, jan. 1998. ISSN 0018-9340. Disponível em: <https://doi.org/10.1109/12.656078>.

EVOLVING, L.-s. I. S. D. D. 2004. Disponível em: <http://delis.upb.de/>.

GARCES-ERICE, L. et al. Data indexing in peer-to-peer dht networks. In: IEEE. *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*. [S.l.], 2004. p. 200–208.

GODFREY, P.; SHENKER, S.; STOICA, I. *Minimizing churn in distributed systems*. [S.l.]: ACM, 2006.

GUPTA, A.; LISKOV, B.; RODRIGUES, R. Efficient routing for peer-to-peer overlays. In: *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*. Berkeley, CA, USA: USENIX Association, 2004. (NSDI'04), p. 9–9. Disponível em: <http://dl.acm.org/citation.cfm?id=1251175.1251184>.

HERMANN, M.; PENTEK, T.; OTTO, B. Design principles for industrie 4.0 scenarios. In: *2016 49th Hawaii International Conference on System Sciences (HICSS)*. [S.l.: s.n.], 2016. p. 3928–3937. ISSN 1530-1605.

KAGERMANN, H.; WAHLSTER, W.; HELBIG, J. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0 – Securing the Future of German Manufacturing Industry*. München, apr 2013. Disponível em: <http://forschungsunion.de/pdf/industrie_4_0_final_report.pdf>.

KATAL, A.; WAZID, M.; GOUDAR, R. Big data: issues, challenges, tools and good practices. In: IEEE. *Contemporary Computing (IC3), 2013 Sixth International Conference on*. [S.l.], 2013. p. 404–409.

KHAN, K. S. et al. Five steps to conducting a systematic review. *Journal of the royal society of medicine*, SAGE Publications Sage UK: London, England, v. 96, n. 3, p. 118–121, 2003.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004.

KITCHENHAM, B. et al. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, Elsevier, v. 51, n. 1, p. 7–15, 2009.

KLEIN, J.; GORTON, I. Runtime performance challenges in big data systems. In: ACM. *Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development*. [S.l.], 2015. p. 17–22.

KOPPE, J. P. Hyperdht-dht de um salto baseada em hipercubo virtual distribuido. 2013.

KUHL, J. G.; REDDY, S. M. Distributed fault-tolerance for large multiprocessor systems. In: *Proceedings of the 7th Annual Symposium on Computer Architecture*. New York, NY, USA: ACM, 1980. (ISCA '80), p. 23–30. Disponível em: <http://doi.acm.org/10.1145/800053.801905>.

KUROSE, J.; ROSS, K. *Computer Networking: A Top-down Approach*. Addison-Wesley, 2010. (Pearson International edition). ISBN 9780136079675. Disponível em: <https://books.google.com.br/books?id=gxLePQAACAAJ>.

LANEY, D. 3d data management: Controlling data volume, velocity and variety. *META group research note*, v. 6, n. 70, p. 1, 2001.

LIBRARY, A. D. 2018. Disponível em: <https://dl.acm.org/>.

LIBRARY, I. X. D. 2018. Disponível em: <https://ieeexplore.ieee.org/>.

LLOYD, S.; GOKHALE, M. Near memory key/value lookup acceleration. In: *Proceedings of the International Symposium on Memory Systems*. New York, NY, USA: ACM, 2017. (MEMSYS '17), p. 26–33. ISBN 978-1-4503-5335-9. Disponível em: <http://doi.acm.org/10.1145/3132402.3132434>.

MAYMOUNKOV, P.; MAZIÈRES, D. Kademlia: A peer-to-peer information system based on the xor metric. In: *IPTPS*. [S.l.: s.n.], 2002.

MOHAMMED, S. I.; SHARAF, H. M.; OMARA, F. A. Information retrieval using dynamic indexing. *Proceedings of INFOS2014*, 2014.

MONNERAT, L. R.; AMORIM, C. L. D1ht: a distributed one hop hash table. In: *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*. [S.l.: s.n.], 2006. p. 10 pp.–. ISSN 1530-2075.

MONTRESOR, A.; JELASITY, M. Peersim: A scalable p2p simulator. In: *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. [S.l.: s.n.], 2009. p. 99–100. ISSN 2161-3567.

MORSELLI, R. et al. Efficient lookup on unstructured topologies. *IEEE Journal on selected areas in communications*, IEEE, v. 25, n. 1, 2007.

NAKAGAWA, E. Y. et al. *Revisão Sistemática da Literatura em Engenharia de Software: Teoria e Prática*. [S.l.]: Elsevier Brasil, 2017.

PANDEY, M.; AHMED, S. M.; CHAUDHARY, B. D. 2t-dht: A two tier dht for implementing publish/subscribe. In: IEEE. *Computational Science and Engineering, 2009. CSE'09. International Conference on*. [S.l.], 2009. v. 2, p. 158–165.

PARK, G. et al. Chordet: an efficient and transparent replication for improving availability of peer-to-peer networked systems. In: ACM. *Proceedings of the 2010 ACM Symposium on Applied Computing*. [S.l.], 2010. p. 221–225.

PICCONI, F.; SENS, P. Using incentives to increase availability in a dht. In: IEEE. *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. [S.l.], 2006. p. 8–pp.

PREPARATA, F. P.; METZE, G.; CHIEN, R. T. On the connection assignment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, EC-16, n. 6, p. 848–854, Dec 1967. ISSN 0367-7508.

PUTHUSSERI, K. S. et al. Storage and availability aware fragment placement for p2p storage systems. In: IEEE. *2019 International Conference on Communication and Signal Processing (ICCSP)*. [S.l.], 2019. p. 0867–0871.

RATNASAMY, S. et al. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 31, n. 4, p. 161–172, ago. 2001. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/964723.383072>.

REINSEL, D.; GANTZ, J.; RYDNING, J. The digitization of the world: From edge to core. 2018. Disponível em: <https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf>.

RUOSO, V. K. Uma estratégia de testes logarítmica para o algoritmo hi-adsd. 2013.

SAGIROGLU, S.; SINANC, D. Big data: A review. In: *2013 International Conference on Collaboration Technologies and Systems (CTS)*. [S.l.: s.n.], 2013. p. 42–47.

SHRIVASTAVA, B. K.; KHATANIAR, G.; GOSWAMI, D. Performance enhancement in hierarchical peer-to-peer systems. In: IEEE. *Communication Systems Software and Middleware, 2007. COMSWARE 2007. 2nd International Conference on*. [S.l.], 2007. p. 1–7.

SMITH, K. *45 Incredible and Interesting Twitter Statistics*. 2017. Disponível em: <https://www.brandwatch.com/blog/44-twitter-stats/>.

SMITH, K. *47 Incredible Facebook Statistics and Facts*. 2018. Disponível em: <https://www.brandwatch.com/blog/47-facebook-statistics/>.

STEINER, M.; EN-NAJJARY, T.; BIERSACK, E. W. A global view of kad. In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. New York, NY, USA: ACM, 2007. (IMC '07), p. 117–122. ISBN 978-1-59593-908-1. Disponível em: <http://doi.acm.org/10.1145/1298306.1298323>.

STEINER, M.; EN-NAJJARY, T.; BIERSACK, E. W. A global view of kad. In: ACM. *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. [S.l.], 2007. p. 117–122.

STOICA, I. et al. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, ACM, v. 31, n. 4, p. 149–160, 2001.

SUN, M.-T. et al. Attribute-based overlay network for non-dht structured peer-to-peer lookup. In: IEEE. *Parallel Processing, 2007. ICPP 2007. International Conference on*. [S.l.], 2007. p. 62–62.

SUN, P. et al. Metaflow: a scalable metadata lookup service for distributed file systems in data centers. *IEEE Transactions on Big Data*, IEEE, 2016.

TANENBAUM, A.; STEEN, M. V. *Sistemas distribuídos: princípios e paradgmas*. Pearson Educación, 2007. ISBN 9788576051428. Disponível em: <https://books.google.com.br/books?id=r2SGPgAACAAJ>.

TANG, C. et al. Low traffic overlay networks with large routing tables. In: ACM. *ACM SIGMETRICS Performance Evaluation Review*. [S.l.], 2005. v. 33, n. 1, p. 14–25.

YALAGANDULA, P.; DAHLIN, M. *A scalable distributed information management system*. [S.l.]: ACM, 2004.

ZHANG, H.; GOEL, A.; GOVINDAN, R. Improving lookup latency in distributed hash table systems using random sampling. *IEEE/ACM Transactions on Networking (TON)*, IEEE Press, v. 13, n. 5, p. 1121–1134, 2005.

ZHANG, Q.-f. et al. A novel scalable architecture of cloud storage system for small files based on p2p. In: IEEE. *2012 IEEE International Conference on Cluster Computing Workshops*. [S.l.], 2012. p. 41–47.

ZHANG, Z. et al. Seerdis: a dht-based resource indexing and discovery scheme for the data center. In: SOCIETY FOR COMPUTER SIMULATION INTERNATIONAL. *Proceedings of the 19th High Performance Computing Symposia*. [S.l.], 2011. p. 26–32.