Replicação de Dados no VCube Baseada em DHT de Salto Único

Alexandre B. Neto[1], **Luiz A. Rodrigues**[1] **e Elias P. Duarte Jr.**[2]

[1] Ciência da Computação – Universidade Estadual do Oeste do Paraná (UNIOESTE)
Cascavel – PR – Brasil

[2]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

alexandrebn01@gmail.com, luiz.rodrigues@unioeste.br, elias@inf.ufpr.br

Abstract. *This paper presents a strategy to replicate files on a VCube, a virtual topology based on a hypercube that presents several logarithmic properties. The proposed strategy is based on a single-hop Distributed Hash Table (DHT), a data structure that provides an efficient and scalable solution to locate information in a P2P (peer-to-peer) network. The system uses a replication technique that fragments file data across system participants in order to improve availability. Simulation results confirm the efficiency of the proposed solution in scenarios with and without faults.*

Resumo. *Este trabalho investiga a replicação de dados utilizando o VCube, uma topologia virtual baseada em hipercubo com diversas propriedades logarítmicas. A estratégia proposta é baseada em tabela hash distribuída (DHT* - Distributed Hash Table*) de salto único, estrutura de dados que oferece uma solução eficiente e escalável para a localização de informações em redes peer-to-peer. O sistema proposto utiliza uma técnica de replicação que fragmenta os arquivos entre os participantes do sistema, buscando garantir a disponibilidade dos dados. Resultados de simulação confirmam a eficiência da solução proposta em cenários com e sem falhas de nodos.*

1. Introdução

Atualmente uma variedade de organizações necessitam distribuir informações e proporcionam serviços para milhares de usuários. A necessidade de desenvolver tecnologias que garantam a escalabilidade dos serviços cresce constantemente. Para um sistema ser escalável, deseja-se que este não possua dependências de pontos centralizados e que seja auto-organizável, prevendo a alteração dinâmica da sua composição em tempo de execução. Outro ponto importante é a disponibilidade, especialmente levando em conta que a medida que o número de componentes aumenta, maior é a probabilidade da ocorrência de falhas.

Uma estrutura de dados que possui grande parte das propriedades citadas é a tabela hash distribuída (DHT - *Distributed Hash Table*). Esta estruturas são comuns em redes *peer-to-peer* (P2P), que são sistemas distribuídos que possuem a característica de se auto-organizar, com o objetivo de realizar o compartilhamento de recursos. As redes P2P surgiram originalmente como sendo uma alternativa ao modelo cliente/servidor. Em um

modelo de sistema P2P puro não há a noção de que um par (*peer*) seja apenas cliente ou servidor, um *peer* é, ao mesmo tempo, cliente e servidor. Além disso, uma rede P2P pode ser classificada em dois tipos, não-estruturada e estruturada. Na primeira, a disposição dos arquivos não possui vínculo com a topologia rede, na segunda, os arquivos são alocados em posições específicas da rede e não de forma aleatória. Desse modo, as buscas são realizadas de forma eficiente e escalável [Androutsellis-Theotokis e Spinellis 2004].

As DHTs são estruturas de dados que armazenam informações distribuídas sobre múltiplos *peers* do sistema. Para o armazenamento, elas utilizam o conceito de chave/valor, mais conhecido por sua utilização em tabelas *hash* tradicionais. Além disso, usam uma estrutura chamada tabela de roteamento para encaminhar uma consulta a partir do *peer* solicitante até o responsável pela chave procurada. Cada participante do sistema possui uma tabela. Ela é composta por um subconjunto de entradas que referenciam outros pares, associando um par à sua chave e a seu respectivo endereço na rede [Ghodsi 2006].

Algumas implementações utilizam tabelas de roteamento completas, onde cada *peer* conhece todos os demais participantes do sistema. As DHTs que usam essa abordagem são definidas como sistemas de salto único, pois dependem de um único salto para a consulta. Em outras implementações, cada *peer* faz uso de tabelas de roteamento parciais e necessitam que uma consulta seja roteada por diversos *peers* do sistema. As DHTs que fazem uso dessa abordagem são definidas como sistemas de múltiplos saltos.

Neste trabalho é apresentada uma estratégia de replicação de dados em uma topologia virtual baseada em hipercubo conhecida como VCube [Duarte et al. 2014] para uma solução de DHT de salto único denominada VCube-DHT, também proposta neste trabalho. O VCube possui diversas propriedades logarítmicas, escaláveis por definição. A estratégia de replicação proposta é baseada na fragmentação dos dados dos arquivos mantidos entre os participantes do sistema. Esta é uma diferença importante da técnica proposta para diversas outras existentes que fazem replicação total, isto é, os dados são replicados por completo entre os participantes da rede. Entre tais estratégias está a HyperDHT [Koppe et al. 2014], que também é uma DHT baseada em hipercubo virtual, mas prevê replicação total, além de não ser baseada no VCube. Este trabalho apresenta dados experimentais de comparação da estratégia proposta com a HyperDHT.

O restante do texto está organizado da seguinte forma. A Seção 2 apresenta uma abordagem de DHTs de único salto. A Seção 3 exibe as principais características referentes ao modelo do sistema. A Seção 4 aborda as principais características do VCubeDHT. A Seção 5 apresenta os resultados obtidos em relação as simulações realizadas. Por fim, a Seção 6 exibe as principais conclusões sobre o trabalho elaborado, bem como sugestões de trabalhos futuros a serem desenvolvidos.

2. Trabalhos Relacionados

Enquanto algumas DHTs mantêm tabelas de roteamento com informações sobre todos os nodos do sistema, outras optam pela utilização de tabelas parciais que são distribuídas entre os nodos da rede. As soluções que usam essa abordagem visam minimizar o tráfego de manutenção das tabelas de roteamento, com prejuízo para a latência das consultas.

CAN (*Content-Addressable Network*) [Ratnasamy et al. 2001] é um sistema completamente distribuído que não necessita de controle centralizado. Este possui uma estru-

tura baseada em um espaço cartesiano virtual multi-dimensional de coordenadas, sendo completamente lógico, não possuindo qualquer relação com a estrutura física do sistema. Além disso, este espaço é particionado de forma dinâmica entre os participantes do sistema. Estes mantêm informações acerca de um pequeno número de nodos adjacentes, armazenando o endereço IP e as coordenadas desses. O número de saltos necessários em uma consulta geralmente será maior que naqueles sistemas no qual os participantes possuem tabela de roteamento completa.

Chord [Ratnasamy et al. 2001] é um modelo de sistema que proporciona rápido mapeamento de chaves entre os nodos do sistema utilizando uma variante de *hashing* consistente. O identificador do nodo é selecionado através do *hash* de seu endereço IP, e o identificador da chave é escolhido baseado no *hash* da mesma. O uso de uma função *hash* distribuída espalha as chaves sobre os nodos de maneira ordenada e balanceada em um anel, conhecido como *Chord Ring* de tamanho 2^m, sendo *m* o número de bits dos identificadores. Além disso, mantém o equilíbrio do sistema, resolvendo os problemas relativos ao balanceamento de carga. O Chord é considerado de múltiplos saltos pois cada nodo possui uma tabela de roteamento parcial, denominada *finger table*, usada sobretudo, para reduzir a latência das consultas.

O HyperDHT [Koppe et al. 2014] propõe uma solução na qual o procedimento de localização de uma informação ou um objeto distribuído na rede seja realizado de forma rápida, eficiente e escalável. Esse sistema utiliza uma infra-estrutura baseada em um hipercubo virtual disponibilizada pelo algoritmo de diagnóstico distribuído DiVHA[Bona et al. 2006], que provém serviços na qual caracterizam-na como sistema de salto único. O sistema implementa mecanismos para o posicionamento e busca dos objetos na rede P2P, em que associa chaves aos nodos e objetos da rede. Além disso, realiza o particionamento e mapeamento das chaves da tabela *hash* entre os nodos utilizando técnicas de *hash* consistente. O sistema também implementa protocolos de entrada de novos integrantes da rede, que determina, de acordo com a topologia atual da rede, uma posição para inserção do novo nodo onde ele é mais necessário. Por fim, o HyperDHT implementa funções para replicar as informações ou objetos dispostos na rede.

3. Modelo do Sistema

Este trabalho considera um sistema distribuído como um conjunto finito Π com $n > 1$ processos independentes, onde $\Pi = \{p_0, p_1, ..., p_{n-1}\}$. Estes se comunicam e coordenam suas ações através de trocas de mensagens [Raynal 2013]. O modelo do presente sistema é associado a um grafo cuja topologia formada é baseada em um hipercubo virtual. Cada vértice representa uma posição na rede de sobreposição que pode ou não ser ocupada. Estes vértices estão estabelecidos em *clusters* e o número de vértices pertencentes a um *cluster* é definido de acordo com a dimensão *d* do hipercubo e determinado por 2^d.

Assim como no HyperDHT, o sistema proposto permite que os *peers* possam tanto sair como entrar da rede. Sendo assim, um vértice pode ser considerado *ocupado* se ele contém um *peer*, caso contrário é classificado como *vazio*. Assim como os vértices, os participantes da rede também podem assumir dois estados, sendo eles *disponíveis* ou *indisponíveis*. Um *peer* considerado disponível é aquele que realiza suas ações de forma esperada, respondendo corretamente às requisições dos demais participantes feitas a ele. De modo contrário, um *peer* indisponível é aquele que, por algum motivo, se comporta

de maneira inesperada, não possuindo comunicação com os demais participantes da rede.

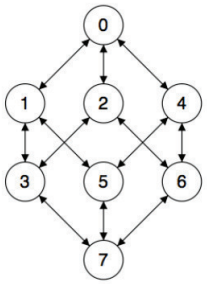
Os *peers* notificam-se uns aos outros sobre a ocorrência de um *evento* na rede por meio de trocas de mensagens. Um evento é uma mudança de estado, seja de um vértice ou de um *peer*. O processo que verifica a ocorrência de um evento é conhecido como teste. Os testes são realizados em rodadas por meio de mensagens trocadas entre os *peers* adjacentes de forma a classificá-los como disponíveis ou indisponíveis. A fim de realizar o diagnóstico do sistema, bem como a construção da rede de sobreposição, este trabalho utiliza o algoritmo de diagnóstico distribuído VCube [Ruoso 2013].

3.1. O Algoritmo VCube

O VCube é uma solução de diagnóstico distribuído que constrói e mantém os processos do sistema com base em uma topologia virtual baseada em um hipercubo. O hipercubo virtual é criado e mantido de acordo com as informações de diagnósticos obtidas por meio de um sistema de monitoramento de processos, descrito em [Duarte et al. 2014]. A cada execução do VCube, cada processo é capaz de testar outros processos no sistema para verificar se estão corretos ou falhos. Um processo é correto se a resposta ao teste realizado for recebida corretamente dentro de um intervalo de tempo.

Os processos são estabelecidos em *clusters*. Em cada rodada de testes um processo *i* testa o primeiro processo sem falha *j* na lista de processos de cada *cluster s* e obtém informação sobre os processos naquele *cluster*. Os membros de cada *cluster s*, bem como a ordem em que eles serão testados por um processo *i* são dadas pela função $C_{i,s}$. Ela determina que o processo *i* testa o processo $j \in C_{i,s}$, somente se o processo *i* é o primeiro correto em $C_{j,s}$. Logo, todo processo é testado uma única vez em cada rodada, garantindo uma latência de diagnóstico de $log_2 N$ rodadas no caso médio e $\log_2^2 N$ no pior caso.

A Figura 1 ilustra a organização lógica de um hipercubo de três dimensões. Em $C_{(0,s)}$, o processo 0 testa o processo 1 quando *s* é 1, depois testa o processo 2 quando *s* é 2, e por fim, quando *s* é 3, testa o processo 4 e é notificado com informações atualizadas sobre o estado dos demais processos daquele *cluster*.



s	$c_{0,s}$	$c_{1,s}$	$c_{2,s}$	$c_{3,s}$	$c_{4,s}$	$c_{5,s}$	$c_{6,s}$	$c_{7,s}$
1	1	0	3	2	5	4	7	6
2	2 3	3 2	0 1	1 0	6 7	7 6	4 5	5 4
3	4 5 6 7	5 4 7 6	6 7 4 5	7 6 5 4	0 1 2 3	1 0 3 2	2 3 0 1	3 2 1 0

Figura 1. Organização Hierárquica do VCube de $d = 3$ dimensões.

4. VCubeDHT

O sistema utilizado como base para a elaboração do VCubeDHT foi o HyperDHT [Koppe et al. 2014], descrito na Seção 2. A principal motivação para elaboração deste sistema foi a implementação de uma estratégia de replicação de dados mais eficiente que a usada no HyperDHT. Além disso, uma nova solução de diagnóstico foi utilizada, o VCube [Duarte et al. 2014].

As principais estratégias usadas no HyperDHT, como mecanismos para posicionamento e busca de objetos na rede, particionamento e distribuição consistente das chaves e também protocolos de entrada e saída, foram mantidas na elaboração do VCubeDHT. Para garantir a disponibilidade dos arquivos armazenados, foi implementada, no sistema proposto, uma abordagem de replicação ativa, explorando uma nova alternativa para propagação das réplicas entre os *peers* do sistema.

4.1. Determinação dos Vizinhos

A fim de determinar os vizinhos que irão receber as réplicas de um *peer*, a presente proposta utiliza a mesma estratégia adotada pelo HyperDHT. Ela define que cada participante do sistema realiza um cálculo baseado em três parâmetros, sendo eles, um identificador i do vértice no qual deseja-se verificar seus vizinhos, a dimensão d na qual determina que os vértices a serem testados são aqueles pertencentes ao *cluster* d, e um contador z, que representa a distância mínima entre dois vértices. O cálculo é feito com o uso da operação de $\oplus$ (ou exclusivo) entre o identificador i e o contador z, identificando os vértices mais próximos daquele representado pelo identificador i. A função utilizada é definida em [Koppe et al. 2014] e representada por: $C_{i,d} = i \oplus z,\ \forall z \in \mathbb{Z}_{\geq 0} : z \leq 2^d - 1$.

Na tabela da Figura 1 são exibidos os resultados após o término da execução da função $C_{i,d}$ feita por cada *peer* em um sistema de dimensão $d = 3$. Nela é possível notar, por exemplo, que o método identifica o primeiro vértice mais próximo ao vértice 0 sendo o vértice 1, o segundo mais próximo sendo o vértice 2, e o terceiro, sendo o vértice 3, e assim sucessivamente. No entanto, se o vértice analisado não estiver ocupado, ele não poderá fazer parte do grupo de replicação, dessa forma, o próximo da lista é examinado.

4.2. Replicação de Dados no VCubeDHT

O VCubeDHT é uma proposta de propagação de dados que replica os fragmentos em nível de *byte*. O pseudo-código responsável pelo processo de propagação das réplicas é apresentado no Algoritmo 1. A função MAKE_REPLICAS recebe como entrada dois parâmetros. O primeiro, denotado por k, caracteriza o fator de replicação escolhido, isto é, o número de vizinhos nos quais os dados serão replicados. Já o segundo, identificado como $frag$, é o fator de fragmentação, que define a quantidade de blocos nos quais cada arquivo será divido.

A replicação faz uso das informações de estado de cada *peer* (ocupado ou não-ocupado), armazenadas em $vertices$, e da lista de arquivos mantidos no *peer* i que está executando a replicação, referenciada por $files$. Na linha 3 os blocos de bytes de cada arquivo que devem ser enviados para cada um dos k *peers* vizinhos de i é calculada pela função CONFIGUREBLOCKS e armazenado em $blocks_k$. O cálculo consiste na criação de uma tabela verdade de tamanho 2^{frag} e na seleção das linhas com $frag - 1$ valores 1.

Em seguida, caso o vértice j sendo examinado esteja ocupado (linha 7), o *peer* responsável é considerado como pertencente à cadeia de replicação. Cada arquivo será fragmentado em $frag$ fragmentos. Após isso, os blocos serão enviados para os *peers* identificados como pertencentes à cadeia de replicação de i (linha 8).

A Figura 2 apresenta a execução do método de replicação feita pelo *peer* 0, cujos fatores de replicação e fragmentação são iguais a 3. Nesse exemplo, o arquivo *R.txt* será fragmentado em 3 blocos: BC, AC e AB. Assim, o vizinho mais próximo do *peer* 0 irá

Algoritmo 1 Replicação do arquivos do *peer* i (p_i)

Entrada:

$files$ ▷ Conjunto dos arquivos alocados ao *peer* i

$vertices[0, .., n-1]$ ▷ Informação sobre os vértices ocupados e não-ocupados

1: **procedure** MAKE_REPLICAS(k, $frag$)
2: $z = 0$
3: $\forall b = 1, .., k : blocks_b \leftarrow$CONFIGUREBLOCKS($b$, k, $frag$)
4: **while** $k > 0$ **and** $z < n$ **do** ▷ Envia os respectivos blocos para as k réplicas
5: $z++$
6: $j = i \oplus z$
7: **if** $vertices[j]$ **is** $occupied$ **then**
8: SEND($blocks_k$) **to** p_j
9: $k--$

10: **function** CONFIGUREBLOCKS(b, k, $frag$)
11: $blocks_b \leftarrow \emptyset$
12: **for all** $file \in files$ **do**
13: $blocks_b \leftarrow$ os fragmentos do arquivo para a réplica b com base em k e $frag$
14: **return** $blocks_b$

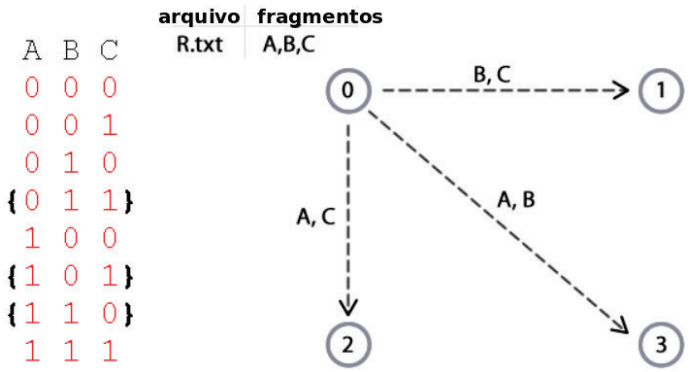


Figura 2. Exemplo de replicação a partir do método MAKE_REPLICAS**.**

receber os blocos B e C, o segundo mais próximo receberá os blocos A e C e, por fim, o terceiro mais próximo receberá os blocos A e B.

4.3. Balanceamento de Réplicas

Com o intuito de realizar o diagnóstico dos participantes da rede, após um período de tempo que pode ser ajustado pela necessidade da aplicação, o VCubeDHT faz com que todos os *peers* executem o algoritmo de diagnóstico distribuído VCube. Se após o diagnóstico for identificado um *peer* falho, além de identificar o novo *peer* responsável pelo vértice deixado pelo *peer* falho, há a necessidade da redistribuição dos blocos de dados entre os *peers* que possuem o *peer* falho como pertencente a seu grupo de replicação.

Após o término do diagnóstico, cada participante da rede determina, através de seu vetor de *timestamp* (que indica o seu conhecimento sobre o estado de cada *peer* do

sistema), a ocorrência de um *peer* falho. Depois dessa identificação, é verificado se o *peer* que está executando o algoritmo faz parte da cadeia de replicação daquele *peer* diagnosticado como falho. Na sequência, o mesmo *peer* replicará os blocos de dados referentes a seus arquivos ao seu novo vizinho, substituindo o *peer* falho em sua cadeia de replicação. Após esse procedimento, é identificado o novo *peer* responsável pelo vértice em que se encontrava o *peer* falho e, por fim, caso o *peer* que esteja executando o algoritmo não for responsável pelo vértice deixado pelo *peer* falho, este deverá enviar todos os blocos referentes ao *peer* falho ao novo responsável pelo vértice desocupado. A Figura 3 apresenta esse processo. Os vértices disponíveis são identificados pelos círculos com a cor branca, e, o indisponível, representado pelo círculo com maior destaque. Cada *peer* armazena os blocos de dados referentes a cada réplica dos arquivos que possui.

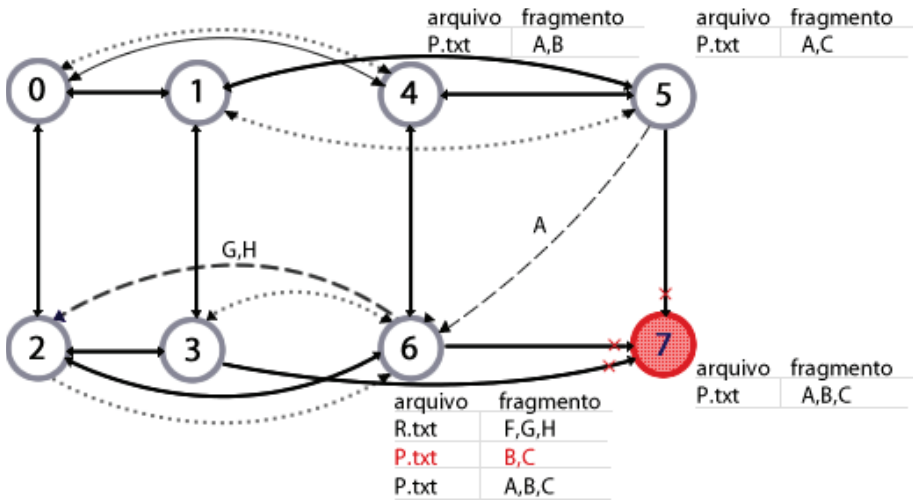


Figura 3. Exemplo da redistribuição dos blocos pelos *peers* do sistema.

Após o diagnóstico do sistema, todos os participantes possuem conhecimento do estado dos demais, porém, os que iniciam o procedimento de reorganização do sistema são aqueles que possuíam o *peer* falho como membro de seu grupo de replicação. Desse modo, o *peer* 6 busca um novo vizinho, substituindo o *peer* 7 e enviando a réplica de seu arquivo (em blocos) a ele. Nota-se que nesse processo, os blocos de dados do *peer* 6 (G e H) referentes ao arquivo R.txt são enviados ao *peer* 2. O *peer* 5 é responsável por enviar o bloco de dados referentes ao *peer* falho. Portanto, o *peer* 5 envia o bloco A ao novo responsável, sendo este o *peer* 6. Esse procedimento é representado pelas conexões tracejadas. Após esse processo, o *peer* 6 possuirá todos os blocos de dados do arquivo referente ao *peer* 7. Da mesma forma que o *peer* 6 identificou seu novo vizinho, os demais *peers* também terão de fazê-lo. Tal ação é representada pelas conexões pontilhadas.

5. Avaliação Experimental

Com o intuito de simular um cenário próximo a uma rede de computadores baseada na Internet, foram construídas diferentes redes constituídas de *peers*, *roteadores* e *links* de comunicação. Foram criados cenários com 8, 16, 32, 64, 128, 256 e 512 participantes.

O número de roteadores foi definido por um valor logarítmico, representado por $\log_2 N$, sendo N o número de participantes da rede. Consequentemente, o número de

roteadores presentes em uma rede com 8 participantes foi limitado a 3. Em um cenário composto por 16 participantes foram definidos 4 roteadores, e assim sucessivamente. As conexões entre os roteadores também foram definidas aleatoriamente, de modo que um roteador não pode se conectar a todos os demais, mas sim com $R - 2$ vizinhos, sendo R o número de roteadores. Foi definido que cada roteador terá no mínimo 1 e no máximo $N - (R - 1)$ participante(s) da rede conectado(s) a ele. A largura de banda de cada *link* de conexão também foi aleatorizada, possuindo valores entre 20 a 40 megabits por segundo. Além disso, a latência de cada *link* foi aleatorizada com valores entre 10 a 40 milissegundos.

Para todos os testes, os fatores de replicação e fragmentação foram ajustados em $log_2 N$, sendo N o número de *peers* do sistema. Portanto, em um sistema com 8 *peers*, cada *peer* terá 3 vizinhos e, no VCubeDHT, cada arquivo será fragmentado em 3 blocos. O tamanho do arquivo configurado no simulador SimGrid [Legrand et al. 2016], varia entre 1, 10, 100 megabytes a 1 gigabyte. O roteamento dos pacotes de dados entre os participantes da rede foi feito pelo algoritmo Dijkstra [Dijkstra 1959]. Por fim, o critério escolhido para comparação entre os sistemas foi o tempo total de simulação.

5.1. Avaliação do Protocolo de Entrada

A etapa de análise do Protocolo de Entrada (PE) tem como objetivo exibir os resultados obtidos em relação ao método de propagação de dados utilizado pelo HyperDHT e VCubeDHT, destacando o resultado obtido durante o Protocolo de Entrada (PE). Vale ressaltar que em todos os testes, não foram geradas falhas durante a entrada dos participantes na rede. Além disso, o critério escolhido para comparação entre os sistemas foi o tempo de simulação total para que os participantes da rede repliquem seus dados. A Tabela 1 exibe os resultados dos testes de acordo com os parâmetros pré-estabelecidos. Nota-se que as colunas identificadas por (H) representam o HyperDHT, já aquelas em (V), o VCubeDHT. Em todos os testes o VCubeDHT obteve melhor tempo em relação ao HyperDHT, isso ocorreu pois o algoritmo proposto replica um conjunto menor de dados.

No cenário onde a rede é composta por 8 *peers* e cada um possui um arquivo de 1 MB (megabyte), no HyperDHT, cada *peer* transfere 3 MB, totalizando 24 MB propagados em 11,67 segundos. Já no sistema VCubeDHT, cada *peer* transfere dois blocos de aprox. 350 KB, totalizando cerca de 5 MB transferidos em 9,39 segundos.

No cenário descrito, houve um número menor de *bytes* a ser transferido pelo VCubeDHT, sendo aprox. 20 MB. Nos cenários com 8 e 512 *peers*, nota-se um resultado próximo para ambos sistemas. Isso ocorre pois a latência impactada na transferência do arquivo completo pelo HyperDHT é bem próxima a latência inferida no envio dos blocos feitos pelo VCubeDHT. Em razão disso, o ganho de desempenho do VCubeDHT não é muito significativo, mesmo com um número menor de *bytes* enviados. Além disso, é possível observar que o mesmo ocorre para os diferentes cenários. As Figuras 4 a 7 apresentam quatro gráficos de comparação entre os sistemas VCubeDHT e HyperDHT, de acordo com os resultados obtidos na Tabela 1.

5.2. Avaliação da Reorganização do Sistema

Após a detecção de um participante falho na rede, tanto o HyperDHT como o VCubeDHT tem como objetivo inicial identificar o novo *peer* responsável pelo vértice onde estava o

Tabela 1. Tempos de execução do protocolo de entrada do HyperDHT (H) e do VCubeDHT (V).

Peers	Tempo (segundos)							
	H	V	H	V	H	V	H	V
8	11,67	9,39	61,41	39,15	581,18	366,83	5917,56	3731,54
16	20,29	17,63	108,22	81,76	1047,35	769,15	10656,76	7815,15
32	28,64	25,47	179,39	140,84	1783,46	1406,43	18233,48	14374,41
64	47,96	40,53	345,92	288,01	3303,90	2727,09	34475,60	28479,64
128	47,35	42,42	341,96	289,03	3211,84	2823,17	33355,79	28379,51
256	83,93	71,56	743,39	653,22	7475,21	6465,41	75886,69	66317,94
512	108,67	101,72	1055,13	935,04	10468,69	9339,63	107317,68	95121,49
	1 MB		**10 MB**		**100 MB**		**1 GB**	
	Tamanho do Arquivo							

Figura 4. PE - Arquivo de 1 MB

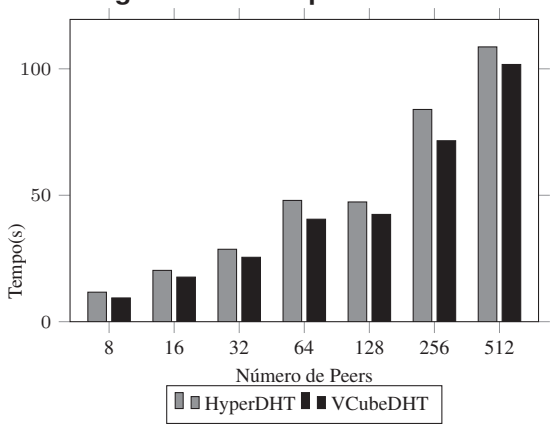
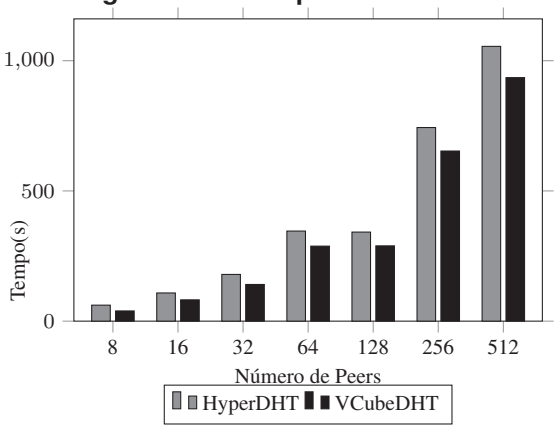


Figura 5. PE - Arquivo de 10 MB



participante falho, de modo a responder pelos dados armazenados por esse. Portanto, após essa etapa, inicia-se o processo de Reorganização do Sistema (RS), que verifica o comportamento do VCubeDHT após o diagnóstico de *peers* falhos na rede.

Nessa etapa foi considerado que nenhum novo *peer* solicita a entrada na rede. A escolha inicial do *peer* falho ocorreu de forma aleatória, porém, posteriormente foi pré-configurada a falha desse para coleta dos resultados em um sistema com maior número de participantes e com arquivos de tamanhos maiores. A Tabela 2 exibe os resultados dos testes realizados de acordo com os parâmetros pré-estabelecidos. O tempo de simulação foi novamente escolhido como critério de desempenho.

Baseado no cenário composto por 8 *peers*, onde o *peer* 7 encontra-se falho e o arquivo a ser replicado possui 1 *megabyte*, no sistema HyperDHT os *peers* 4, 5 e 6 (pertencentes ao grupo de replicação do *peer* 7), enviam uma réplica de seu arquivo aos *peers* 0, 1 e 2 nesta ordem, transferindo um total 3 MB. No VCubeDHT os *peers* 4, 5 e 6 enviam parte de seu arquivo, em dois blocos de 350 KB, aos *peers* 0, 1 e 2, nesta ordem. Em seguida, é identificado pelo *peer* 5, o único bloco de dado necessário para que o *peer* 6 possua o arquivo completo referente ao *peer* falho. Na sequência o *peer* 5 transfere

Figura 6. PE - Arquivo de 100 MB

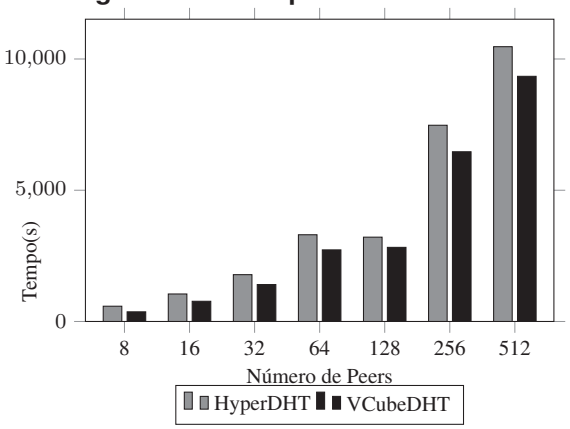
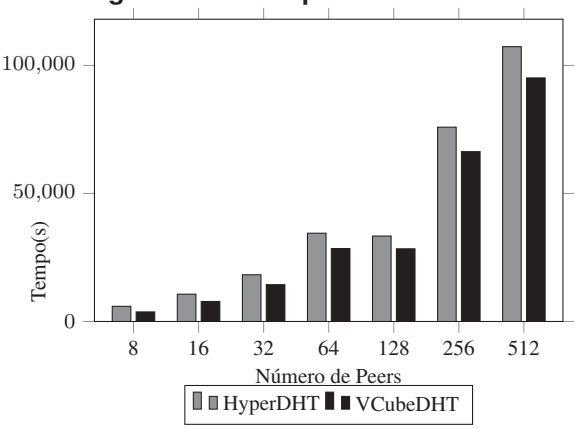


Figura 7. PE - Arquivo de 1 GB



apenas um bloco de dado cujo tamanho é 350 KB ao *peer* 6. Portanto, o número de *bytes* transferidos pelo VCubeDHT foi de aprox. 2,5 MB contra aprox. 3 MB no HyperDHT, ou seja, cerca de 700 MB a menos. Nota-se, através da Tabela 2, que o VCubeDHT obteve pior desempenho, transferindo as réplicas em 2,12 segundos, sendo que o HyperDHT as transferiu em 1,50 segundos. As Figuras 8 e 9 exibem os arquivos de *log* da execução dos sistemas de acordo com o cenário descrito.

Tabela 2. Tempo para a reorganização do HyperDHT (H) e do VCubeDHT (V).

Peers	Tempo (segundos)							
	H	V	H	V	H	V	H	V
8	1,50	2,12	8,25	7,73	76,91	66,48	781,81	669,59
16	1,89	2,56	6,70	7,10	54,77	52,44	548,32	517,94
32	3,12	2,99	9,26	9,05	77,36	70,34	776,56	699,62
64	3,83	4,81	9,67	10,19	72,14	67,35	717,55	658,98
128	1,64	3,79	6,67	6,48	58,27	50,08	588,07	504,19
256	2,88	3,79	9,69	10,13	77,79	73,50	776,99	704,14
512	2,94	3,64	10,72	11,02	88,55	86,45	887,63	861,12
Tamanho do Arquivo	**1 MB**		**10 MB**		**100 MB**		**1 GB**	

É possível observar que para cada dado enviado, o VCubeDHT obteve um pequeno ganho, porém, o tempo para transferência de cada dado em ambos sistemas é próximo, mesmo que o número de *bytes* enviado pelo HyperDHT seja superior ao VCubeDHT. Além disso, é possível notar que o *peer* 5, após enviar os blocos de dados 1 e 2 ao *peer* 1, aguarda até o mesmo recebe-lo (devido ao envio bloqueante) e, posteriormente realiza o envio do bloco de dado 1 ao *peer* 6. Nota-se que até o momento o VCubeDHT possui uma pequena vantagem, porém, ao emitir o último bloco, o tempo total se torna maior em relação ao obtido pelo HyperDHT.

Mesmo com grande parte dos resultados destacando um melhor desempenho do sistema HyperDHT, onde o tamanho do arquivo armazenado pelos *peers* é de 1 *megabyte*, é possível notar que o VCubeDHT obteve um melhor desempenho em uma rede composta

```
[peer-5: 160000.004] send: peer-4 | rec: peer-0 | size block: 699050.666 | data: 730750_R_1_2
[peer-6: 160000.005] send: peer-5 | rec: peer-1 | size block: 699050.666 | data: 913438_R_1_2
[peer-7: 160000.006] send: peer-6 | rec: peer-2 | size block: 699050.666 | data: 109612_R_1_2
[peer-1: 160001.043]    peer-0 receive piece of peer-4   | data: 730750_R_1_2
[peer-1: 160001.043]    size block: 699050.66 | time to transfer: 1.039

[peer-3: 160001.130]    peer-2 receive piece of peer-6 | data: 109612_R_1_2
[peer-3: 160001.130]    size block: 699050.66 | time to transfer: 1.124

[peer-2: 160001.397]    peer-1 receive piece of peer-5 | replicas: 3 | data: 913438_R_1_2
[peer-2: 160001.397]    size block: 699050.66 | time to transfer: 1.392

[peer-6: 160001.397] send: peer-5 | rec: peer-6 | size block: 349525.333 | data: 127881_R_1

[peer-7: 160002.127]    peer-6 receive piece of peer-5 | data: 127881_R_1
[peer-7: 160002.127]    size block: 349525.33 | time to transfer: 0.729

[peer-7: 165002.127] time send: 160000.004 | time receive: 160002.127 | time total = 2,123
```

Figura 8. Arquivo do *log* de execução do VCubeDHT composto por 8 *peers*.

```
[peer-5: 160000.004] send: peer-4 | rec: peer-0 | size block: 1048576.0 | data: 730750
[peer-6: 160000.005] send: peer-5 | rec: peer-1 | size block: 1048576.0 | data: 913438
[peer-7: 160000.006] send: peer-6 | rec: peer-2 | size block: 1048576.0 | data: 109612

[peer-1: 160001.281] peer-0 receive piece of peer-4 | replicas: 4 | data: 730750
[peer-1: 160001.281] size file: 1048576.0 | time to transfer: 1.277

[peer-3: 160001.385] peer-2 receive piece of peer-6 | replicas: 4 | data: 109612
[peer-3: 160001.385] size file: 1048576.0 | time to transfer: 1.379

[peer-2: 160001.502] peer-1 receive piece of peer-5 | replicas: 4 | data: 913438
[peer-2: 160001.502] size file: 1048576.0 | time to transfer: 1.497

[peer-2: 165001.502] time send: 160000.004 | time receive: 160001.502 | time total = 1,498
```

Figura 9. Arquivo do *log* de execução do HyperDHT composto por 8 *peers*.

de 32 *peers*. Tal desempenho se deve ao fato de que a rota percorrida pelo *peer* responsável por replicar o último bloco de dado possui menor latência, logo, o VCubeDHT manterá a pequena vantagem que possuía no decorrer do envio dos blocos de dados.

Nota-se que se a rota percorrida pelo *peer* responsável por replicar o último bloco de dado referente ao *peer* falho possuir consideravelmente uma menor latência em relação as demais, o resultado na replicação de um arquivo pequeno pelo VCubeDHT se mostrará mais eficiente em relação ao HyperDHT. Caso contrário, possuirá um pior desempenho.

A partir dos dados da Tabela 2, nota-se que o tempo de transferência de ambos sistemas diminui constantemente a medida com que o tamanho do arquivo é maximizado. No momento em que os sistemas trabalham com arquivos de 100 *megabytes* de tamanho, o VCubeDHT começa a obter um melhor desempenho em relação ao HyperDHT.

Através do método utilizado pelo simulador para efetuar o cálculo do tempo de comunicação entre os participantes do sistema, notou-se que a latência impacta fortemente no envio de dados com tamanhos menores, como na replicação de arquivos com 1 *megabyte*, onde o VCubeDHT obteve pior desempenho. A medida com que o tamanho do arquivo aumenta, torna-se visível a eficiência do VCubeDHT sobre o HyperDHT. As Figuras 10 a 13 apresentam quatro gráficos de comparação entre os sistemas VCubeDHT e HyperDHT, de acordo com os resultados obtidos na Tabela 2.

Vale ressaltar que não se torna possível, de acordo com o cenário estipulado, comparar as diferentes redes, isto é, a rede composta por 128 *peers* com a de 64 *peers*. Isso ocorre pois mesmo que o número de réplicas a serem enviadas no primeiro seja maior que segundo cenário, as características da rede, serão diferentes para ambos e impactarão de

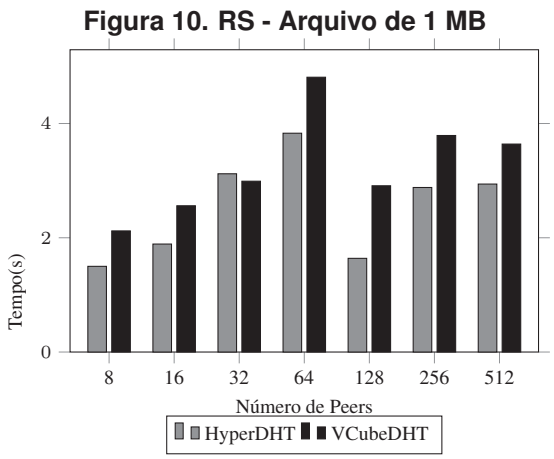


Figura 10. RS - Arquivo de 1 MB

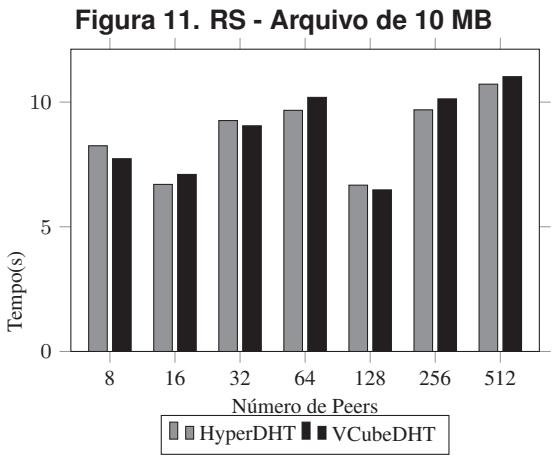


Figura 11. RS - Arquivo de 10 MB

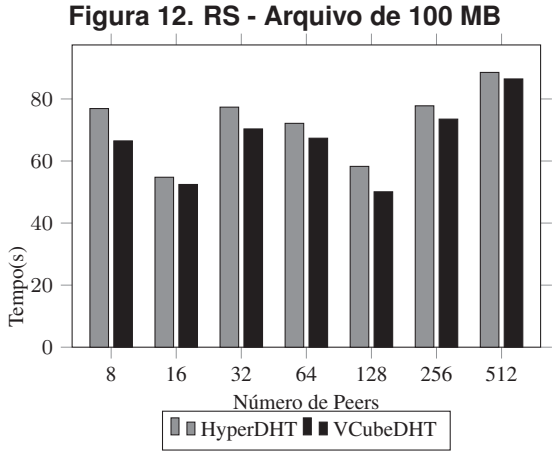


Figura 12. RS - Arquivo de 100 MB

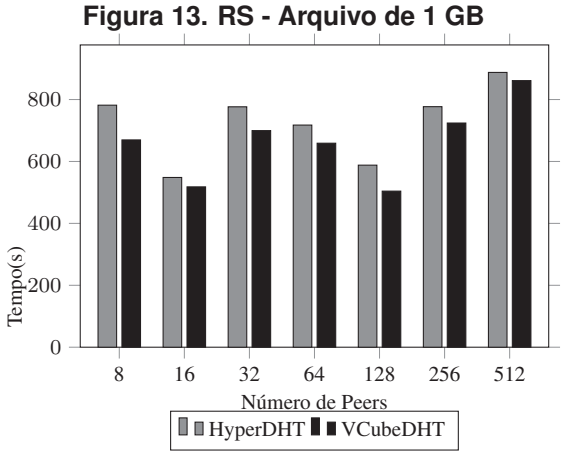


Figura 13. RS - Arquivo de 1 GB

diferentes formas. Além disso, de acordo com o algoritmo de Reorganização do Sistema, os novos vizinhos a serem identificados serão diferentes para ambas as redes.

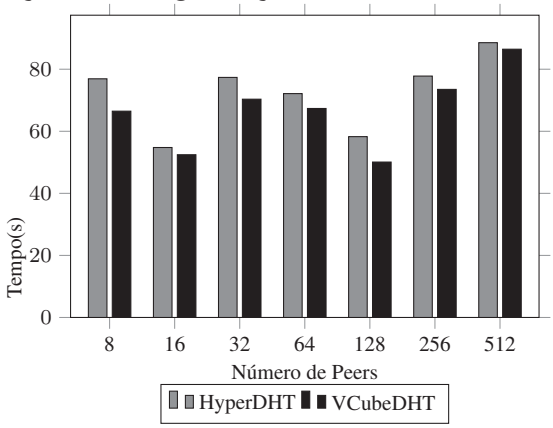
5.3. Disponibilidade dos Dados

Além de analisar o desempenho do algoritmo proposto na Reorganização do Sistema, há a necessidade de investigar a disponibilidade dos dados garantida por ele. Sendo assim, a partir de outro modo de visualização dos resultados da ultima análise, é possível definir cenários que destacam a disponibilidade dos sistemas HyperDHT e VCubeDHT.

A Figura 14 exibe os resultados obtidos na Reorganização do Sistema com arquivos de 100 *megabytes* de tamanho. Analisando a primeira coluna, que representa a rede composta por 8 *peers*, nota-se que, se por algum motivo, após 66,48 segundos do início do processo de reorganização do sistema, os participantes que estavam replicando seus dados se tornem falhos, o sistema HyperDHT não garantiria a disponibilidade total dos dados armazenados pelos *peers*. Já no VCubeDHT, observa-se que após esse instante, os *peers* já haveriam terminado a reorganização, garantindo uma maior disponibilidade. Porém, nos cenários onde o VCubeDHT replica arquivos de 10 *megabytes* de tamanho, nota-se que este não garante uma melhor disponibilidade sobre o HyperDHT, visto que o tempo para reorganizar o sistema, no primeiro, na maior parte dos casos, se torna superior.

No HyperDHT não há a redistribuição dos dados referentes ao *peer* falho. Isso

Figura 14. Replicação na Reorganização do Sistema com Arquivo de 100 MB



ocorre pois o novo responsável possuirá a réplica dos arquivos em questão. Porém, em um cenário onde os *peers* não falhem instantaneamente, após a falha de todos os *peers* pertencentes a um único grupo de replicação, nota-se que as réplicas do primeiro *peer* a se tornar falho não existirão mais no sistema. Já no VCubeDHT, como há essa etapa de redistribuição dos blocos, todas as réplicas estarão disponíveis no sistema. Sendo assim, o VCubeDHT garante, novamente, uma maior disponibilidade dos arquivos.

6. Considerações Finais

A principal contribuição deste trabalho é a investigação da replicação de dados a partir da fragmentação de arquivos em uma DHT de um salto sobre a topologia virtual VCube. O VCube tem várias propriedades logarítmicas, que garantem a escalabilidade da solução proposta. Na abordagem empregada pelo VCubeDHT, os *peers* do sistema replicam seus dados de forma parcial, buscando obter melhorias em relação ao tempo de transmissão e disponibilidade, garantindo a tolerância a falhas de participantes da rede responsáveis por determinados segmentos de dados. O sistema foi implementado através de simulação e foram feitos testes comparativos com outro sistema DHT, utilizado como base para o desenvolvimento deste trabalho.

Com base nos resultados obtidos, pode-se concluir que a alternativa de replicação baseada em fragmentação, implementada pelo VCubeDHT, se mostra mais eficiente que a alternativa de replicação total do HyperDHT, onde os arquivos a serem replicados possuem tamanhos iguais ou superiores a 100 *megabytes*. Na emissão de arquivos cujos tamanhos são inferiores a 100 *megabytes*, o algoritmo proposto obtém melhores resultados se a latência impactada na emissão do último bloco de dado referente ao *peer* falho for significantemente menor que as demais. Além disso, é importante ressaltar que a alternativa abordada pelo VCubeDHT garante a disponibilidade total dos dados caso os *peers* do sistema não se tornem falhos instantaneamente. Porém, no pior caso, o sistema mantém a disponibilidade dos arquivos, tolerando $frag - 1$ falhas, onde $frag$ representa o fator de fragmentação configurado no sistema.

Neste trabalho, as informações armazenadas pelos participantes do sistema não sofrem alterações. Dessa forma, propõe-se como trabalho futuro, a exploração do VCubeDHT para que este suporte trabalhar com arquivos modificados, garantindo a consistência dos dados. Além disso, propõe-se a implementação e execução do VCubeDHT em

um ambiente real, ou mesmo em um ambiente de testes como o PlanetLab.

Agradecimentos

O trabalho foi desenvolvido no âmbito do grupo PET-Comp, financiado pelo Ministério da Educação (MEC), com apoio da Fundação Araucária/SETI, proj. 144/2015-45112 e CNPq, proj. 309143/2012-8.

Referências

Androutsellis-Theotokis, S. e Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36:335–371.

Bona, L. C. E., Duarte Jr., E. P., Mello, S. L. V. e Fonseca, K. (2006). Hyperbone: Uma rede overlay baseada em hipercubo virtual sobre a internet. In: *Anais do XXIV Simpósio Brasileiro de Redes e Sistemas distribuídos*, SBRC'2006.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271.

Duarte, Jr., E. P., Bona, L. C. E. e Ruoso, V. K. (2014). VCube: A provably scalable distributed diagnosis algorithm. In: *5th Work. on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA'14, pp. 17–22, Piscataway, USA. IEEE Press.

Ghodsi, A. (2006). *Distributed k-ary system: Algorithms for distributed hash tables*. Tese de Doutorado, Department of Electronic, Computer, and Software Systems, KTH-Royal Institute of Technology.

Koppe, J. P., de Bona, L. C. E. e Jr., E. P. D. (2014). Hyperdht - dht de um salto baseada em hipercubo virtual distribuído. In: *Anais do IX Workshop de Redes P2P, Dinâmicas, Sociais e Orientadas a Conteúdo*, SBRC-Wp2p+.

Legrand, A., Marchal, L. e Casanova, H. (2016). Scheduling distributed applications: the simgrid simulation framework.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. e Shenker, S. (2001). A scalable content-addressable network. In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pp. 161–172, New York, NY, USA. ACM.

Raynal, M. (2013). *Distributed Algorithms for Message-Passing Systems*. Springer Publishing Company, Incorporated.

Ruoso, V. K. (2013). *Uma estratégia de testes logarítmica para o algoritmo Hi-ADSD*. Tese de mestrado, Universidade Federal do Paraná, Curitiba, PR.