Uma Estratégia de Testes Logarítmica para o Algoritmo Hi-ADSD

Vinicius K. Ruoso, Luis Carlos E. Bona, Elias P. Duarte Jr.

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brasil

{vkruoso,bona,elias}@inf.ufpr.br

Abstract. *The* Hierarchical Adaptive Distributed System-level Diagnosis (Hi-ADSD) *is a distributed diagnosis algorithm that creates a virtual topology based on a hypercube. A hypercube is a scalable structure by definition, presenting important topological features like: symmetry, logarithmic diameter and good fault tolerance properties. The latency of the algorithm executed on a system with N nodes is at most $\log_2^2 N$ However, the number of executed tests in the worst case is quadratic. This work presents a new testing strategy for the* Hi-ADSD *algorithm, which guarantees that at most $N \log_2 N$ tests are executed at each $\log_2 N$ testing rounds. The maximum latency is mantained in $\log_2^2 N$ testing rounds. The algorithm is adapted to the new testing strategy. Furthermore, the use of* timestamps *is adopted to allow each node to retrieve diagnosis information from several other nodes, thus reducing the average latency. The new algorithm is specified, formal proofs are given, and experimental results obtained by simulations are presented and compared with the* Hi-ADSD *algorithm.*

Resumo. *O algoritmo* Hierarchical Adaptive Distributed System-level Diagnosis (Hi-ADSD) *é um algoritmo de diagnóstico distribuído que cria uma topologia virtual baseada em um hipercubo. O hipercubo é uma estrutura escalável por definição, apresentando características topológicas importantes como: simetria, diâmetro logarítmico e boas propriedades para tolerância a falhas. A latência do algoritmo quando executado em um sistema com N nodos é de no máximo $\log_2^2 N$ rodadas de teste. Entretanto, o número de testes executados no pior caso é quadrático. Este trabalho apresenta uma nova estratégia de testes para o algoritmo* Hi-ADSD *que garante que são executados no máximo $Nlog_2N$ testes a cada log_2N rodadas. A latência máxima é mantida em $\log_2^2 N$ rodadas de teste. O algoritmo é adaptado para a nova estratégia de testes. Além disso, foi adotado o uso de* timestamps *para permitir que cada nodo obtenha informações de diagnóstico a partir de diversos outros nodos, consequentemente reduzindo a latência média. O novo algoritmo é especificado, suas provas formais são demonstradas e resultados experimentais obtidos por simulações são apresentados e comparados com o* Hi-ADSD.

1. Introdução

Considere um sistema composto de N nodos, cada um destes nodos pode assumir um de dois estados: falho ou sem-falha. Um algoritmo de diagnóstico distribuído em nível de sistema [Duarte Jr. et al. 2011, Masson et al. 1996] permite que os nodos sem-falha do

sistema determinem o estado de todos os outros nodos. Nodos desse sistema devem ser capazes de executar testes em outros nodos, podendo assim determinar em qual estado o nodo testado está. Assume-se que um testador sem-falha consegue determinar corretamente o estado do nodo testado.

Esta asserção referente à capacidade de um nodo sem-falha realizar testes com precisão faz parte do primeiro modelo proposto na área de diagnóstico em nível de sistema, apresentado por Preparata, Metze e Chien [Preparata et al. 1967], chamado modelo PMC. Segundo o modelo, o diagnóstico clássico consiste na escolha de um conjunto de testes a serem realizados e na obtenção dos resultados destes testes para o diagnóstico do sistema.

O diagnóstico adaptativo [Hakimi and Nakajima 1984], por sua vez, sugere que uma unidade determine quais testes vai executar baseando-se nos resultados dos testes executados previamente. Desta forma, cada unidade adapta o conjunto de testes que irá realizar de acordo com o resultado de testes anteriores. No diagnóstico adaptativo uma rodada de testes é definida como o tempo necessário para que as unidades sem-falha do sistema executem seus testes assinalados.

O modelo PMC considera a existência de um observador central, que é livre de falhas, para a realização do diagnóstico do sistema. Unidades realizam testes e enviam seus resultados a este observador, que os analisa e conclui qual o estado de todas as unidades do sistema. O diagnóstico distribuído em nível de sistema [Hosseini et al. 1984] é uma abordagem onde este observador central não existe, e cada unidade é responsável por realizar o diagnóstico do sistema como um todo.

Para que seja possível comparar o desempenho de diferentes algoritmos de diagnóstico são utilizadas duas métricas: latência e quantidade de testes. Em um sistema diagnosticável a ocorrência de um evento em uma unidade leva um certo tempo para ser detectada pelas outras unidades. A latência é definida como o número de rodadas de testes necessárias para que todos os nodos do sistema realizem o diagnóstico de um evento ocorrido. Define-se um evento como sendo a troca de estado de uma unidade do sistema, tanto de falho para sem-falha quanto de sem-falha para falho. Outra métrica importante para a avaliação de desempenho de estratégias de diagnóstico é a quantidade máxima de testes que o sistema realiza em uma rodada de testes.

Em [Duarte Jr. and Nanya 1998] o algoritmo *Hierarchical Adaptive Distributed System-Level Diagnosis (Hi-ADSD)* é introduzido. Este algoritmo apresenta uma latência de diagnóstico de no máximo $\log_2^2 N$ rodadas de teste para um sistema de N nodos. Os nodos são organizados em *clusters* progressivamente maiores. Os testes são executados de uma maneira hierárquica, inicialmente no *cluster* com dois nodos, indo para o *cluster* com quatro nodos, e assim sucessivamente, até que o *cluster* com $N/2$ nodos seja testado. Para que o nodo receba informações sobre um dado *cluster*, o nodo executa testes até que ele encontre um nodo não falho ou teste todos os nodos do *cluster* falhos. Considerando a rede como um todo, o maior número de testes executados em uma rodada de testes é $N^2/4$. Este cenário acontece quando $N/2$ nodos de um mesmo *cluster* estão falhos, e os outros $N/2$ nodos testam este *cluster* na mesma rodada de testes.

Neste trabalho apresentamos uma nova estratégia de testes para o algoritmo *Hi-ADSD*. Esta estratégia requer menos testes por rodadas de teste no pior caso, e mantém a

latência de $\log_2^2 N$ rodadas. Na estratégia original utilizada pelo *Hi-ADSD* para determinar seus testes, um nodo pode ser testado por vários outros nodos. A ideia principal da nova estratégia de testes é evitar que estes testes extras sejam executados. Para isso, uma ordem de testes é estabelecida para cada nodo em cada *cluster*. Assim, para um nodo i fazer um teste no nodo j na rodada de testes s, o nodo i deve ser o primeiro nodo sem-falha na lista de nodos testadores de j na rodada s. Desta maneira conseguimos evitar múltiplos testes em um mesmo nodo. Como cada nodo recebe no máximo um teste por rodada de testes, a nova estratégia garante um número de testes executados no sistema de no máximo $N \log_2 N$ a cada $\log_2 N$ rodadas de teste, fato que é provado no trabalho. Resultados de simulação são apresentados comparando o algoritmo *Hi-ADSD* original com a nova estratégia, tanto em termos de latência como do número de testes executados.

O restante deste trabalho está organizado da seguinte forma. A seção 2 apresenta conceitos e trabalhos relacionados. A seção 3 descreve a nova estratégia de testes. Por fim, a seção 4 conclui o trabalho.

2. Diagnóstico Distribuído Hierárquico

Considere um sistema S que consiste em um conjunto de N nodos, $n_0, n_1, ..., n_{N-1}$, alternativamente o nodo n_i pode ser chamado de nodo i. Assumimos que o sistema é totalmente conectado, ou seja, existe um canal de comunicação entre qualquer par de nodos e estes canais nunca falham. Cada nodo n_i pode estar falho ou sem-falha. Os nodos do sistema são capazes de testar outros nodos e determinar seu estado corretamente. O objetivo do diagnóstico é determinar o estado de todos os nodos do sistema. Neste trabalho assumimos que falhas são apenas do tipo *crash*, i.e. nodos falhos não interagem com outros nodos do sistema. Este capítulo apresenta conceitos e algoritmos de diagnóstico, focando em particular nos algoritmos hierárquicos.

2.1. O Algoritmo *Hi-ADSD*

O algoritmo *Hierarchical Adaptive Distributed System-Level Diagnosis (Hi-ADSD)* [Duarte Jr. and Nanya 1998] é hierárquico, além de ser distribuído e adaptativo, e realiza testes baseados em uma estratégia de dividir para conquistar. Esta hierarquia é baseada em um hipercubo. Quando todos os nodos do sistema estão sem-falha o grafo de testes é um hipercubo completo.

O hipercubo [LaForge et al. 2003] é uma estrutura escalável por definição, apresentando características topológicas importantes como: simetria, diâmetro logarítmico e boas propriedades para tolerância a falhas. Em um hipercubo, um nodo i é vizinho de um nodo j se, e somente se, os identificadores de i e j têm apenas um *bit* diferente em suas representações binárias. Dizemos que um hipercubo com 2^d elementos tem dimensão d. Em um hipercubo de dimensão d, um caminho entre quaisquer dois nodos tem no máximo d arestas. A Figura 1 mostra um 3-hipercubo.

No *Hi-ADSD* os nodos são organizados em *clusters* para execução de testes e obtenção de informações de diagnóstico. O tamanho dos *clusters* é sempre uma potência de dois, e baseia-se na estrutura do hipercubo. No primeiro intervalo de testes um nodo testa outro nodo e vice-versa, formando um *cluster* de dois nodos. No segundo intervalo de testes o testador testa um *"cluster"* de dois nodos, isto é, testa um nodo e obtém informação sobre o outro. No terceiro intervalo de testes o testador testa um cluster de

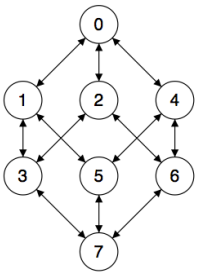

Figura 1. Um 3-hipercubo.

quatro nodos, e assim sucessivamente até que no *d*-ésimo intervalo de testes o testador testa um *cluster* com $N/2$ nodos, e todos formam um *cluster* de N nodos, ou seja, o sistema todo. A Figura 2 mostra os *clusters* formados em um sistema com 8 nodos.

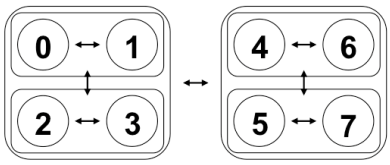

Figura 2. Clusters formados pelo algoritmo *Hi-ADSD* em um sistema de 8 nodos.

A lista ordenada de nodos testados por um determinado nodo i em um *cluster* de tamanho 2^{s-1} é dada pela função $c_{i,s}$. A Equação 1 mostra a especificação do algoritmo para esta função. Definimos o símbolo $\oplus$ como a operação *XOR*. Um exemplo de uso desta função em um sistema com 8 nodos é exposto na Tabela 1.

$$c_{i,s} = i \oplus 2^{s-1}, c_{i\oplus 2^{s-1},s-1}, ..., c_{i\oplus 2^{s-1},1} \quad (1)$$

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

Tabela 1. $c_{i,s}$ para um sistema com 8 nodos.

Quando um testador testa um nodo sem-falha, ele obtém informações de diagnóstico sobre os demais nodos do *cluster* do nodo testado. Quando o testador testa um nodo falho, ele continua testando nodos daquele *cluster* até encontrar um nodo não falho ou testar todos os nodos do *cluster* como falhos. Por exemplo, em um sistema com 8 nodos inicialmente o nodo 0 testa o nodo 1 do *cluster* de tamanho 1, na próxima rodada de testes testa o *cluster* de tamanho 2 composto pelos nodos 2 e 3, por fim testa o *cluster* de tamanho 4 que contém os nodos 4, 5, 6 e 7 e então retorna a testar o *cluster* de tamanho 1, e assim sucessivamente.

O Algoritmo 1 mostra em pseudo-código o funcionamento do algoritmo *Hi-ADSD* executado no nodo i. O algoritmo realiza testes sequencialmente nos nodos retornados pela função $c_{i,s}$ até testar um nodo sem-falha. Ao testar um nodo sem-falha obtém-se

informações de diagnóstico sobre os outros nodos daquele *cluster*. A latência do algoritmo *Hi-ADSD* é de no máximo $\log_2^2 N$ rodadas de teste.

Esta estratégia de testes pode gerar um número quadrático de testes. Por exemplo, suponha que um sistema de N nodos está com $N/2$ nodos falhos, e que tais nodos formam um *cluster* com $N/2$ nodos. Portanto, este é um sistema onde existe um *cluster* de $N/2$ nodos falhos e outro de $N/2$ nodos não falhos. Seguindo a estratégia acima, suponha que todos os nodos do *cluster* não falho estão na rodada de testes em que irão testar outro *cluster* de tamanho $N/2$. Assim, cada nodo não falho irá realizar $N/2$ testes, resultando em um total de $N/2 * N/2 = N^2/4$, ou seja, O(N^2) testes em uma rodada de testes. Esta mesma complexidade pode ser obtida com um algoritmo de força bruta.

```
1  Algoritmo Hi-ADSD executado no nodo i
2  repita
3      para s ← 1 até log2 N faça
4          repita
5              node_to_test := próximo nodo em c_{i,s}
6              teste(node_to_test)
7              se node_to_test está sem-falha então
8                  atualiza informação de diagnóstico do cluster
9          até node_to_test está sem-falha ou todos os nodos em c_{i,s} estão falhos
10         se todos os nodos em c_{i,s} estão falhos então
11             apaga informações de diagnóstico do cluster
12         durma até o próximo intervalo de testes
13 para sempre
```

Algoritmo 1: Algoritmo *Hi-ADSD*.

Outros algoritmos de diagnóstico distribuído hierárquico já foram propostos, incluindo o algoritmo *Hi-ADSD with Detours* [Duarte Jr. et al. 2009]. Quando um nodo testa um nodo falho em um determinado *cluster*, o testador tenta obter informações sobre aquele *cluster* a partir de nodos de outros *clusters*, ao invés de executar mais testes. Esses caminhos alternativos são chamados *detours* e tem sempre tamanho máximo idêntico à distância no algoritmo original. No algoritmo *Hi-ADSD with Timestamps* após um nodo sem-falha ser testado, o testador obtém informações sobre um conjunto de $N/2$ nodos. Assim, um nodo obtém informações sobre um outro nodo através de vários caminhos. Para indicar qual informação é mais recente são usados *timestamps*, contadores de mudança de estados. No algoritmo *DiVHA (Distributed Virtual Hypercube Algorithm)* [Bona et al. 2008] os testes são assinalados aos nodos de forma que cada nodo tem apenas um testador em cada cluster. Para garantir esta unicidade, o algoritmo faz buscas e comparações no grafo local e determina quais os testes que cada nodo do sistema deve executar de acordo com a situação atual.

3. O Algoritmo VCube

Este capítulo descreve o algoritmo proposto por este trabalho, chamado de Hipercubo Virtual (*VCube*). O novo algoritmo representa uma nova estratégia de testes para o algoritmo *Hi-ADSD* que garante uma quantidade máxima de testes logarítmica mantendo o mesmo

limite para a latência. O restante deste capítulo está organizado da forma a seguir. A Seção 3.1 especifica e descreve o novo algoritmo. A Seção 3.2 exibe uma nova definição para a função $c_{i,s}$ utilizada no novo algoritmo. Na Seção 3.3 são desenvolvidas as provas necessárias para mostrar que o número de testes é no máximo $N \log_2 N$ e que a latência é no máximo $\log_2^2 N$. Por fim, na Seção 3.4 resultados experimentais obtidos através de simulações são apresentados e comparados com o algoritmo *Hi-ADSD*.

3.1. O Algoritmo

Considere um sistema S que consiste em um conjunto de N nodos, $n_0, n_1, \ldots, n_{N-1}$. Alternativamente chamamos o nodo n_i de nodo i. Assumimos que o sistema é completamente conectado, ou seja, existe um canal de comunicação entre qualquer par de nodos. Cada nodo pode assumir um de dois estados, falho ou sem-falha. A combinação dos estados de todos os nodos do sistema constitui a situação de falha do mesmo. Os nodos realizam testes em outros nodos em intervalos de testes, e testes feitos por nodos sem-falha são sempre corretos.

No algoritmo *VCube* os nodos são agrupados em *clusters* para realização de testes. Os *clusters* são conjuntos de nodos cujo tamanho é sempre uma potência de 2. O sistema como um todo é um *cluster* de N nodos. O algoritmo *VCube* executado pelo nodo i é apresentado em pseudo código no Algoritmo 2. No primeiro intervalo de testes o nodo i realiza testes em nodos do *cluster* que contém um nodo ($c_{i,1}$), no segundo intervalo de testes em nodos do *cluster* que contém dois nodos ($c_{i,2}$), e assim por diante, até que os nodos do *cluster* que contém $2^{\log_2 N - 1}$, ou $N/2$ nodos ($c_{i,\log_2 N}$). A equação 2 define a função $c_{i,s}$ para todo $i \in \{0, 1, \ldots, N-1\}$ e $s \in \{1, 2, \ldots, \log_2 N\}$. A função $c_{i,s}$ retorna a lista ordenada de nodos que o nodo i poderá testar durante a rodada de testes s. Define-se o símbolo $\oplus$ como a operação ou exclusivo (*XOR*). A Seção 3.2 descreve detalhadamente o funcionamento desta função.

$$c_{i,s} = i \oplus 2^{s-1} \parallel c_{i \oplus 2^{s-1},k} \mid k = 1, 2, \ldots, s-1 \tag{2}$$

O novo algoritmo é baseado na seguinte regra: antes do nodo i executar um teste no nodo $j \in c_{i,s}$, ele verifica se ele é o primeiro nodo sem-falha em uma lista ordenada de nodos que podem testar o nodo j na rodada s. Depois que o nodo i testa o nodo $j \in c_{i,s}$ como sem-falha, ele obtém novas informações de diagnóstico do nodo j sobre todos os nodos do sistema. Este processo é repetido para todos os nodos pertencentes a $c_{i,s}$, finalizando uma rodada de testes.

Como um testador pode obter informações sobre um determinado nodo através de vários outros nodos, o algoritmo utiliza *timestamps*. Inicialmente todos os nodos são considerados sem-falha e o *timestamp* correspondente é zero. Quando um evento é detectado, isto é, um nodo sem-falha se torna falho ou vice-versa, o *timestamp* correspondente é incrementado. Assim, um *timestamp* par corresponde a um nodo sem-falha e um *timestamp* ímpar corresponde a um nodo falho.

O uso de *timestamps* torna o algoritmo capaz de diagnosticar eventos dinâmicos, ou seja, eventos que ocorrem antes do evento anterior ter sido completamente diagnosticado. Considerando este modelo de falhas dinâmicas é necessário especificar um procedimento para a recuperação dos nodos que são reparados durante a execução do algoritmo.

Utilizamos a mesma estratégia introduzida pelo algoritmo *Hi-ADSD with Timestamps*, onde um campo chamado *u-bit* é utilizado para representar se um valor de *timestamp* é válido ou não.

O nodo i mantém um vetor de *timestamps* $t_i[0..N-1]$. Após obter informação de um nodo j testado como sem-falha, o nodo i atualiza um *timestamp* local apenas quando o valor correspondente obtido for maior que o atual. Uma vantagem adicional desta abordagem é que a latência média do algoritmo é melhorada, pois testadores podem obter novas informações de diagnóstico sobre todos os nodos do sistema através de qualquer nodo testado.

Para evitar que o todo o vetor $t_j[]$ seja transferido quando o nodo i testa o nodo j como sem-falha, é possível implementar uma solução simples na qual o nodo j só envia novas informações, ou seja, informações que mudaram desde a última vez que o nodo j foi testado pelo nodo i.

É importante observar que o sistema é assíncrono, ou seja, em um certo momento, nodos diferentes do sistema podem estar testando *clusters* de tamanhos diferentes. Um nodo executando o algoritmo não é capaz de determinar quais testes estão sendo realizados por outros nodos do sistema em um dado momento. Mesmo se inicialmente os nodos estão sincronizados, após a falha e recuperação de alguns deles o sistema perderia sua sincronicidade. Este fato gera consequências no desempenho do algoritmo, que são tratadas na Seção 3.3.

```
1  Algoritmo VCube executado no nodo i
2  repita
3      para s ← 1 até log2 N faça
4          para todo j ∈ c_{i,s} | i é o primeiro nodo sem falha ∈ c_{j,s} faça
5              teste(j)
6              se j está sem-falha então
7                  se t_i[j] mod 2 = 1 então t_i[j] = t_i[j] + 1
8                  obtém informações de diagnóstico
9              senão
10                 se t_i[j] mod 2 = 0 então t_i[j] = t_i[j] + 1
11         durma até o próximo intervalo de testes
12 para sempre
```

Algoritmo 2: Algoritmo *VCube*.

3.2. A Função $c_{i,s}$

A função $c_{i,s}$ é a base do algoritmo, pois ela é capaz de determinar para um nodo i quais são os *clusters* que ele deverá testar durante cada rodada de teste. A função $c_{i,s}$ retorna a lista ordenada de nodos que o nodo i poderá testar durante a rodada de testes s. Definimos o símbolo $\oplus$ como a operação ou exclusivo (*XOR*). A equação 3 mostra a definição da função $c_{i,s}$, e um exemplo de uso da função para um sistema com 8 nodos pode ser visto na Tabela 2.

$$c_{i,s} = i \oplus 2^{s-1} \parallel c_{i\oplus 2^{s-1},k} \mid k = 1, 2, \ldots, s-1 \tag{3}$$

Para entendermos melhor como a função $c_{i,s}$ funciona, primeiramente vamos lembrar que em um hipercubo existe uma aresta entre o nodo i e o nodo j se, e somente se, na representação binária de i e j existe apenas um *bit* diferente [LaForge et al. 2003]. Também devemos lembrar que dado um número i podemos inverter um de seus bits, utilizando a operação ou exclusivo (*XOR*), simplesmente fazendo $i \oplus 2^x$, onde x é a posição do bit que gostariamos de inverter e $\oplus$ é a operação ou exclusivo (*XOR*).

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

Tabela 2. $c_{i,s}$ para um sistema com 8 nodos.

A função $c_{i,s}$ mostrada na equação 3 é uma contribuição se comparada à versão original especificada pelo algoritmo *Hi-ADSD*. A ordem criada por ela tem propriedades importantes para o novo algoritmo, vistas a seguir.

A Tabela 3 mostra um exemplo da função $c_{i,s}$ calculada para um sistema com 8 nodos onde $s = 3$. Podemos ver que nenhum nodo é repetido nas colunas da tabela, ou seja, $\forall x, y \in \{0, 1, ..., N-1\}, k \in \{0, ..., 3\} \mid x \neq y : c_{x,3}[k] \neq c_{y,3}[k]$. Para provarmos este fato, utilizamos indução.

i	$c_{i,3}[0]$	$c_{i,3}[1]$	$c_{i,3}[2]$	$c_{i,3}[3]$
0	4	5	6	7
1	5	4	7	6
2	6	7	4	5
3	7	6	5	4
4	0	1	2	3
5	1	0	3	2
6	2	3	0	1
7	3	2	1	0

Tabela 3. Detalhe da $c_{i,3}$ para um sistema com 8 nodos.

Teorema 1: $\forall i, j, s, p \mid i \neq j : c_{i,s}[p] \neq c_{j,s}[p]$.

Prova: A prova será por indução em s.

Base: Considere $s = 1$. Para um nodo i a função $c_{i,1} = i \oplus 2^{s-1} = i \oplus 1$. Como 1 em binário tem apenas o *bit* menos significativo como 1, a operação *XOR* irá inverter o *bit* menos significativo de i. Considere i como sendo par, ou seja, seu *bit* menos significativo é 0: assim temos que $i \oplus 1 = i + 1$. Considere i como sendo ímpar, ou seja, seu *bit* menos significativo é 1: assim temos que $i \oplus 1 = i - 1$. Portanto, para qualquer $i, j \mid i \neq j$, $c_{i,1}[0] \neq c_{j,1}[0]$.

Hipótese de indução: Para todo $i, j, p \mid i \neq j : c_{i,k}[p] \neq c_{j,k}[p]$.

Passo: Considere $s = k + 1$.

Para um nodo i a função $c_{i,k+1} = i \oplus 2^k \parallel c_{i \oplus 2^k,1} \parallel c_{i \oplus 2^k,2} \parallel ... \parallel c_{i \oplus 2^k,k}$. Considere a operação $i \oplus 2^k$. Como 2^k tem apenas o k-ésimo *bit* como 1, a operação *XOR* irá inverter

o k-ésimo *bit* de i. Suponha que $j = i \oplus 2^k$, temos que $c_{j,k+1} = (i \oplus 2^k) \oplus 2^k, ...$ Temos então que $(i \oplus 2^k) \oplus 2^k = i$, ou seja, se $c_{i,k+1}[0] = j$ então $c_{j,k+1}[0] = i$.

Como $c_{i,k+1}[0]$ é único para cada i e pela hipótese $c_{i\oplus 2^k,1}$, $c_{i\oplus 2^k,2}$, ... , $c_{i\oplus 2^k,k}$ também são únicos, $c_{i,k+1}$ é único, e portanto, $c_{i,k+1}[p] \neq c_{j,k+1}[p], \forall j \neq i, p$.

Além da função $c_{i,s}$ representar quais nodos o nodo i pode testar na rodada de testes s, ela também representa para o nodo i durante a rodada de testes s, a ordem dos nodos que poderão testar o nodo i nesta rodada. Tecnicamente qualquer ordem pode ser utilizada para representar os nodos que podem testar um outro nodo em uma rodada de testes. Entretanto uma função que satisfaça o *Teorema 1* garante que a cada rodada de testes todos os nodos sem-falha irão realizar pelo menos um teste.

Considere, por exemplo, $c_{5,3} = \{1, 0, 3, 2\}$. Na terceira rodada de testes o nodo 5 irá verificar se ele é o primeiro nodo sem-falha nas listas ordenadas $c_{1,3}$, $c_{0,3}$, $c_{3,3}$ e $c_{2,3}$, respectivamente, e realizar o teste em caso positivo. É garantido que o nodo 5 irá realizar pelo menos um destes testes pois em alguma destas ordens ele será o primeiro. Por outro lado, os nodos 1, 0, 3 e 2, também na terceira rodada de testes, estando todos sem-falhas, irão verificar qual o primeiro nodo sem-falha em $c_{5,3}$, fazendo com que apenas um deles teste o nodo 5.

O uso de uma função $c_{i,s}$ que satisfaz o *Teorema 1* permite, de uma certa forma, manter as propriedades logarítmicas mesmo quando nodos se tornam falhos. A nova função $c_{i,s}$ mantém uma ordem progressiva de acordo com o hipercubo, fazendo com que as arestas criadas por nodos falhos conectem os nodos sem-falha de uma maneira mais similar ao hipercubo original. Suponha que em uma função $c_{i,s}$ o valor de $c_{i,3}$ fosse igual a $\{4, 5, 6, 7\}$ para todo $i \in \{0, 1, 2, 3\}$, ou seja, caso o nodo 4 esteja sem-falha ele será responsável por testar os nodos 0, 1, 2 e 3. Dependendo da função $c_{i,s}$ a carga de testes sobre um nodo pode chegar a ordem de N, e poucas falhas podem fazer com que o desempenho do algoritmo seja prejudicado consideravelmente. A função $c_{i,s}$ proposta distribui a responsabilidade pelos testes de forma igualitária entre todos os nodos.

3.3. Provas Formais

Nesta seção são apresentadas as provas de corretude e pior caso da latência de diagnóstico do algoritmo. Para estas provas assumimos que a situação de falhas do sistema não muda por um período suficiente de tempo. Vale lembrar que as provas de corretude dos algoritmos *Hi-ADSD* e *Adaptive-DSD* também consideram esta asserção.

Definição 1: *O grafo $T(S)$ é um grafo direcionado cujos vértices são os nodos de S. Para cada nodo i, e para cada cluster $c_{i,s}$, existe uma aresta de i para j se i testou j como sem-falha na rodada de testes mais recente em que i testou o cluster $c_{i,s}$.*

Lema 1: *Para qualquer nodo i, qualquer s, e a qualquer instante de tempo t_i, são necessárias, no máximo, $\log_2 N$ rodadas de testes para que o nodo i teste o cluster $c_{i,s}$.*

Prova: O lema segue da definição do algoritmo, ou seja, em um determinado intervalo de testes o nodo i testa pelo menos um nodo de $c_{i,s}$. Como o nodo i testa

$\log_2 N$ *clusters*, são necessárias $\log_2 N$ rodadas de testes para que todos os *clusters* sejam testados. Portanto, no pior caso, para t_i imediatamente após o teste de um dado *cluster*, poderá levar até $\log_2 N$ rodadas de testes para que este *cluster* seja testando novamente.

Teorema 2: *O menor caminho entre qualquer par de nodos sem-falha em $T(S)$ contém, no máximo, $\log_2 N$ arestas.*

Prova: A prova será por indução em t, para um sistema de 2^t nodos.

Base: Considere um sistema de 2^1 nodos. Cada nodo testa o outro, portanto o menor caminho entre eles em $T(S)$ contém uma aresta.

Hipótese de indução: Assuma que para um sistema com 2^k nodos o menor caminho entre qualquer par de nodos em $T(S)$ contém, no máximo, k arestas.

Passo: Por definição, um sistema de 2^{k+1} nodos é composto por dois *clusters* de 2^k nodos. Considere o subgrafo de $T(S)$ que contém apenas os nodos de um destes *clusters*. Pela hipótese anterior o menor caminho entre qualquer par de nodos neste subgrafo contém, no máximo, k arestas. Considere quaisquer dois nodos i e j. Se i e j estão no mesmo *cluster* de 2^k nodos, o menor caminho entre eles em $T(S)$ contém, no máximo, k arestas. Agora considere o caso em que i e j estão em *clusters* de tamanho 2^k diferentes. Sem perda de generalidade, considere o menor caminho entre i e j. O nodo j recebe um teste de algum nodo do *cluster* ao qual o nodo i pertence: chame este nodo de p. Em $T(S)$, a menor distância entre p e j é de uma aresta, e a menor distância de i até p é, no máximo, k arestas. Portanto, o menor caminho de i a j contém, no máximo, $k+1$ arestas.

Teorema 3: *Considere uma situação de falha em um dado momento. Após, no máximo, $\log_2^2 N$ rodadas de testes, cada nodo que permaneceu sem-falha por este período corretamente determina aquela situação de falha.*

Prova: Foi provado no Teorema 2 que o menor caminho entre qualquer par de nodos em $T(S)$ tem, no máximo, $\log_2 N$ arestas. Segundo o Lema 1, cada um dos testes correspondentes a uma aresta em $T(S)$ pode levar até $\log_2 N$ rodadas de testes para ser executado no pior caso. Então, temos até $\log_2 N$ testes diferentes para serem executados, e cada um deles pode levar até $\log_2 N$ rodadas de testes para ser executado. Então, no total, estes testes podem levar no máximo $\log_2 N * \log_2 N$ rodadas de testes para serem executados. Portanto, podem ser necessárias até $\log_2^2 N$ rodadas de teste para um nodo sem-falha obter informação de diagnóstico sobre um evento em S.

Teorema 4: *O número de testes executados por todos os nodos do sistema executando o algoritmo VCube é no máximo $N \log_2 N$ a cada $\log_2 N$ rodadas de testes.*

Prova: Considere os testes executados no nodo j. Para cada $c_{j,s} \mid s = 1, ..., \log_2 N$, o nodo j é testado pelo primeiro nodo sem-falha em $c_{j,s}$. Cada nodo sem-falha em $c_{j,s}$ utiliza a mesma lista ordenada de nodos que contém ele mesmo para verificar se ele é o primeiro nodo sem-falha ou não. Em caso afirmativo ele deve testar o nodo j. Assim cada nodo é testado no máximo $\log_2 N$ vezes a cada $\log_2 N$ rodadas de teste. Como o sistema tem N nodos, o número total de testes é no máximo $N \log_2 N$.

Deve ficar claro que, assim como nos algoritmos *Hi-ADSD* e *Adaptive-DSD*, o limite do número de nodos falhos para que os nodos sem-falha realizem um diagnóstico consistente é $N - 1$. Neste caso, se $N - 1$ nodos estão falhos, o nodo sem-falha irá testar

todos os nodos para diagnosticar o sistema.

Também devemos notar que o número de nodos do sistema, N, não necessariamente precisa ser uma potência de 2. Neste caso, os nodos testadores devem ignorar os $2^{\lceil \log_2 N \rceil} - N$ nodos inexistentes durante testes e recuperação de informações de diagnóstico.

3.4. Resultados Experimentais

Os resultados experimentais do algoritmo *VCube* foram obtidos tanto através de simulação, apresentados nesta seção, como em execução no Planet-Lab, apresentados no próximo capítulo. As simulações foram conduzidas utilizando a linguagem de simulação baseada em eventos discretos, SMPL [MacDougall 1987]. Nodos foram modelados como *facilities* SMPL e cada nodo foi identificado por um número de *token* SMPL. As falhas foram modeladas como um nodo sendo reservado, que é apenas um estado interno da ferramenta de simulação. Durante cada teste, o estado do nodo é verificado, e se este está reservado ele é considerado falho. Caso contrário o nodo é considerado sem-falha. Cada nodo inicia seus testes em um *cluster* escolhido aleatoriamente, fazendo com que os nodos do sistema não estejam completamente sincronizados.

Todos os experimentos foram executados em uma rede de 512 nodos. Os resultados mostrados são a latência média e a quantidade de testes média. A latência é o número de rodadas de testes necessárias para que todos os nodos sem-falha detectem um evento. A quantidade de testes mostrada representa o número de testes realizados em média no sistema em uma situação estável, ou seja, a quantidade de testes após todos os nodos diagnosticarem o evento em questão durante $\log_2 N$ rodadas de testes. A simulação leva em conta a asserção de que um evento ocorre apenas quando o evento anterior tiver sido completamente diagnosticado pelo sistema.

As Figuras 3 e 4 mostram a quantidade de testes e a latência média, respectivamente, em uma rede onde os nodos falham de uma maneira a simular o pior caso do algoritmo *Hi-ADSD*. Os nodos de identificadores 0 a 255 falham de maneira aleatória até que todos estejam falhos e em seguida o mesmo acontece com os nodos de identificadores 256 a 511. Assim, com $N/2$ nodos falhos no mesmo *cluster*, comparamos o comportamento do *VCube* com o *Hi-ADSD*.

Pode-se observar que o *Hi-ADSD* tem um comportamento quadrático quando o número de nodos falhos se aproxima de $N/2$, chegando ao seu pior caso quando o número de testes chega a $N^2/4 = 65536$. Durante os mesmos casos de testes, o algoritmo *VCube* se comporta de maneira logarítmica, nunca passando de seu limite de $N \log_2 N = 4608$ testes por $\log_2 N$ rodadas de testes. Analisando a latência média que ambos algoritmos apresentaram, podemos ver que a nova estratégia de testes apresenta uma diferença considerável se comparada à estratégia original, apesar de ambas apresentarem uma latência máxima teórica de $\log_2^2 N$ rodadas de testes. Esta diferença se deve ao fato de que o algoritmo *Hi-ADSD* original não utiliza *timestamps*, que diminuem consideravelmente a latência.

As Figuras 5 e 6 mostram a quantidade de testes e a latência média, respectivamente, em uma rede onde os nodos falham de uma maneira aleatória. Novamente podemos ver que a latência média do *VCube* é mais baixa que a do *Hi-ADSD*. Um resultado a ser destacado é que, na média, o número de testes executados pelos dois algoritmos é

muito parecido. Estes resultados foram obtidos a partir da execução de cerca de 7000 simulações de cenários de falhas para os dois algoritmos, porém existem $511!$ cenários de falhas possíveis. Como os nodos que irão falhar são escolhidos aleatoriamente, um caso como o pior caso do algoritmo *Hi-ADSD* tem uma probabilidade muito pequena de acontecer.

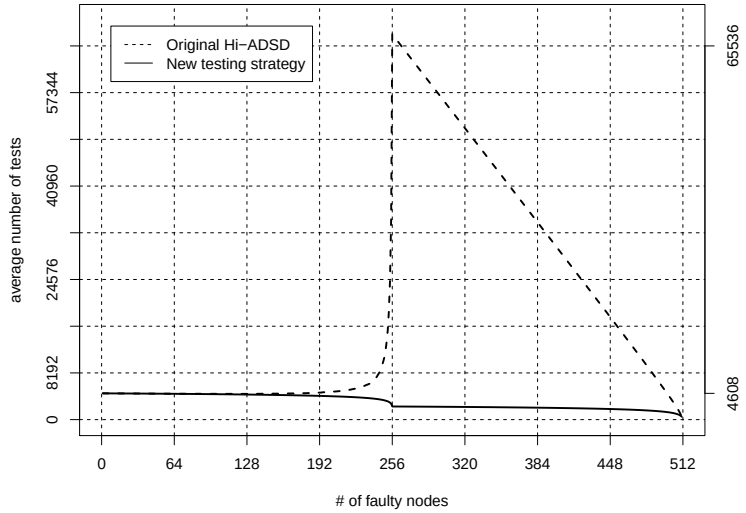

Figura 3. Número de testes médio em uma rede estável de 512 nodos no pior caso do algoritmo *Hi-ADSD*.

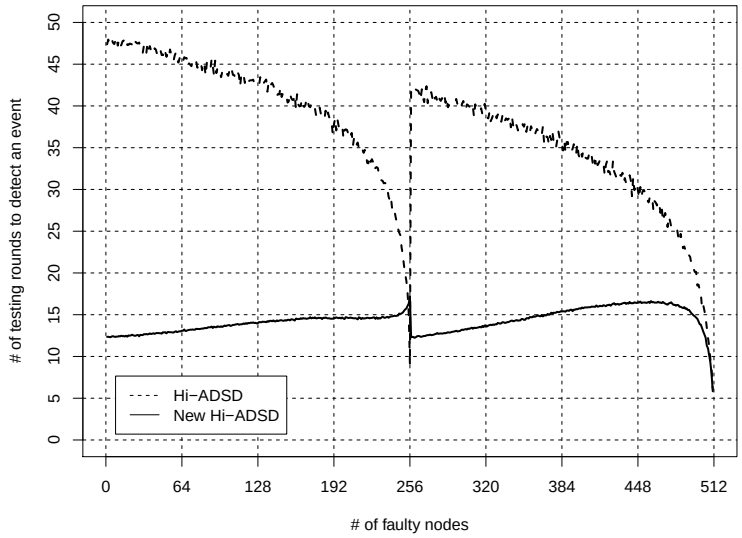

Figura 4. Latência média para detecção de um evento em uma rede de 512 nodos no pior caso do algoritmo *Hi-ADSD*.

4. Conclusão

Este trabalho apresentou o novo algoritmo *VCube* para o diagnóstico distribuído hierárquico que garante propriedades logarítmicas tanto para sua latência quanto para a quantidade máxima de testes executados. Em comparação com outro algoritmo de diagnóstico distribuído hierárquico, o *Hi-ADSD*, o algoritmo proposto neste trabalho reduz

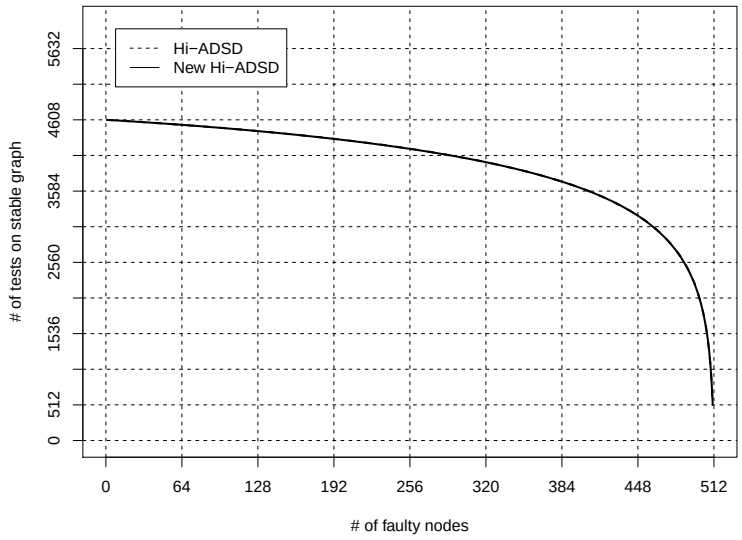

Figura 5. Número de testes médio em uma rede estável de 512 nodos.

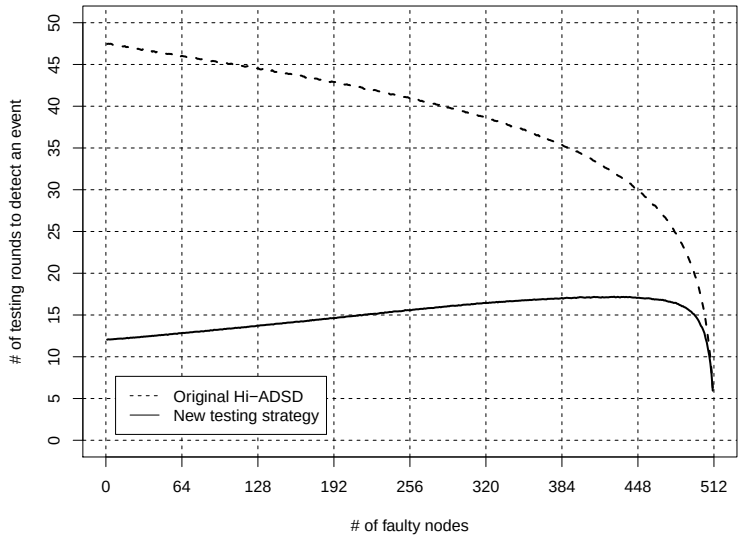

Figura 6. Latência média para detecção de um evento em uma rede de 512 nodos.

o número máximo de testes executados, mantendo o mesmo pior caso para a latência. Enquanto no *Hi-ADSD* o número de testes executados no pior caso é quadrático, no algoritmo proposto o limite máximo é de $N \log_2 N$ testes a cada $\log_2 N$ rodadas de testes. Ambos os algoritmos têm latência máxima de $\log_2^2 N$ rodadas.

O algoritmo *VCube* organiza a execução de testes em *clusters* progressivamente maiores. Na estratégia de testes proposta os testes são realizados com base na seguinte regra: um nodo i na rodada de testes s deve testar o nodo $j \in c_{i,s}$ se, e somente se, o nodo i é o primeiro nodo sem-falha em uma lista ordenada de nodos que podem testar o nodo j na rodada de teste s. Quanto um nodo sem-falha é testado informações de diagnóstico sobre todo o sistema são enviadas. Uma nova função $c_{i,s}$ foi apresentada para representar tanto os nodos que o nodo i pode testar na rodada de testes s quanto os nodos que podem testar o nodo i nesta mesma rodada.

Resultados de simulação inclusive comparando o novo algoritmo com o algoritmo *Hi-ADSD* foram apresentados. Com relação ao número de testes o novo algoritmo apresentou uma quantidade logarítmica de testes em cenários onde o algoritmo *Hi-ADSD* apresentou comportamento quadrático. A latência média do novo algoritmo foi inferior ao algoritmo *Hi-ADSD* em todos os cenários, devido tanto à nova estratégia de testes quanto à obtenção de informações de diagnóstico utilizando *timestamps*.

Trabalhos futuros incluem trabalhar em uma especificação do algoritmo que permite que cada nodo execute testes nos $\log_2 N$ clusters em uma única rodada, mantendo um custo razoável (logarítmico) e, ao mesmo tempo, baixando a latência máxima para $\log_2 N$ rodadas de teste ao invés de $\log_2^2 N$ no pior caso. Reduzir a computação necessária para a execução do algoritmo também deve ser considerada para sua execução em redes com um número grande de nodos. Diversas otimizações são possíveis para evitar que a toda rodada de testes os testadores executem múltiplas verificações para determinar exatamente quais testes devem realizar. Outra linha de trabalho pode expandir a definição do algoritmo para que seja possível realizar diagnóstico em redes de topologia arbitrária.

Referências

Bona, L., Fonseca, K., and Duarte Jr., E. P. (2008). Hyperbone: A scalable overlay network based on a virtual hypercube. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 1025–1030.

Duarte Jr., E., Ziwich, R., and Albini, L. (2011). A survey of comparison-based system-level diagnosis. *ACM Computing Surveys*.

Duarte Jr., E. P., Albini, L., Brawerman, A., and Guedes, A. (2009). A hierarquical distributed fault diagnosis algorithm based on clusters with detours. In *Network Operations and Management Symposium, 2009. LANOMS 2009. Latin American*, pages 1–6.

Duarte Jr., E. P. and Nanya, T. (1998). A Hierarquical Adaptive Distributed System-Level Diagnosis Algotithm. *IEEE Transactions on Computers.*, 47(1):34–45.

Hakimi, S. L. and Nakajima, K. (1984). On Adaptive System Diagnosis. *IEEE Transactions on Computers.*, C-33(3):234–240.

Hosseini, S. H., Kuhl, J. G., and Reddy, S. M. (1984). A Diagnosis Algorithm for Distributed Computing Systems with Dynamic Failure and Repair. *IEEE Transactions on Computers.*, C-33(3):223–233.

LaForge, L., Korver, K., and Fadali, M. (2003). What designers of bus and network architectures should know about hypercubes. *Computers, IEEE Transactions on*, 52(4):525–544.

MacDougall, M. (1987). *Simulating computer systems: techniques and tools*. MIT Press, Cambridge, MA, USA.

Masson, G., Blough, D., and Sullivan, G. (1996). Fault-tolerant computer system design. chapter System diagnosis, pages 478–536. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Preparata, F., Metze, G., and Chien, R. T. (1967). On the Connection Assignment Problem of Diagnosable Systems. *IEEE Transactions on Computers.*, 16:848–854.