MoDiVHA: Uma Estratégia Hierárquica para Assinalamento de Testes Distribuídos

Jefferson Paulo Koppe[1], **Elias P. Duarte Jr.**[1], **Luis C. E. Bona**[1]

[1] Departamento de Informática –Universidade Federal do Paraná (UFPR)
Caixa Postal 19081 – 81531-990 –Curitiba – PR – Brazil

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

Abstract. *Distributed diagnosis allows fault-free nodes of a system to determine the state of all nodes. The diagnosis is performed based on tests assigned to system nodes. In this work we present a hierarchical testing strategy that is scalable by definition. This strategy, called MoDiVHA, is based on DiVHA's test assignment [1], but produces less tests. Reducing the number of tests is important because it can represent a significant reduction on resources required by the diagnosis algorithm. Experimental results were obtained from three series of simulations conducted to compute the number of tests and diagnosis latency for various system sizes and fault situations.*

Resumo. *O diagnóstico distribuído permite que nodos sem-falha de um sistema determinem o estado de todos os nodos do sistema. O diagnóstico é realizado a partir dos resultados de testes assinalados aos nodos do sistema. Neste trabalho apresentamos uma estratégia de testes hierárquica que permite o diagnóstico escalável. Essa estratégia, denominada MoDiVHA, é baseada naquela do algoritmo DiVHA [1], mas resulta em um número de testes inferior. A diminuição do número de testes é importante pois pode representar uma redução significativa de recursos de sistema alocados para realização do diagnóstico. Os resultados experimentais foram obtidos através de três séries de simulações, conduzidas para computar o número de testes e a latência em diversos tamanhos de sistemas e configurações de falhas.*

1. Introdução

Um sistema de gerenciamento de falhas deve ser tolerante a falhas, uma vez que roda na rede e precisa ser confiável mesmo quando a rede apresentar falhas. O diagnóstico em nível de sistema é uma abordagem eficaz para construção de sistemas de gerenciamento distribuídos tolerantes a falhas [2], [3].

Considere um sistema com N nodos onde cada nodo pode estar em um de dois estados, falho ou sem-falha. Considere ainda que existe um enlace de comunicação entre qualquer par de nodo do sistema e que os enlaces de comunicação nunca falham. O objetivo do diagnóstico distribuído em nível de sistema é permitir que os nodos não-falhos determinem o estado, falho ou não-falho, de todos os outros nodos pertencentes ao sistema [4], [5], [6]. Utilizando algoritmos de diagnóstico distribuído é possível construir sistemas de monitoração de rede tolerante a falhas [8].

Os nodos em um sistema de diagnóstico são capazes de testar outros nodos e determinar seu estado corretamente. O conjunto de testes assinalados para cada um dos nodos, bem como a realização do diagnóstico a partir do resultado dos testes, são funções elementares de um algoritmo de diagnóstico distribuído.

O sistema é representado por um grafo direcionado $G = (V, E)$, onde os vértices do grafo representam os nodos do sistema e as arestas indicam os testes que são realizados. A figura 1 mostra um exemplo de sistema com 4 nodos representado por um grafo, o vértice na cor preta indica que o nodo está falho. Uma aresta saindo do nodo u para o nodo v significa que o nodo u testa o nodo v. Cada teste corresponde a um estímulo enviado cuja resposta classificará o nodo testado em falho ou sem-falha.

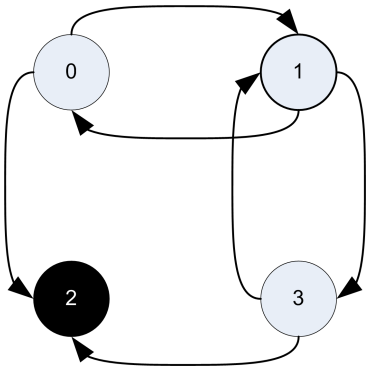

Figura 1. Um exemplo de assinalamento de testes; o nodo 2 está falho.

O desempenho de um algoritmo de diagnóstico distribuído é mensurado, principalmente, por dois valores: (1) a quantidade de testes necessários por rodada; (2) a latência de propagação de um evento. A latência é expressa em rodadas de testes e corresponde ao tempo necessário para que todos os nodos sem-falha descubram a ocorrência de um novo evento [1]. Uma rodada de testes é definida como o período de tempo em que todos os nodos no estado sem-falha executam seus testes. Quanto menor forem esses valores (quantidade de testes e latência) mais eficiente é o algoritmo. No entanto, a busca por um algoritmo ideal não é tão simples pois, o ganho com uma métrica, geralmente implica em perdas na outra métrica. O Adaptive-DSD [8] é um bom exemplo desta contraposição, sua necessidade de apenas N testes para concluir uma rodada de testes o torna muito eficiente neste quesito. Em contra partida este algoritmo tem uma latência de propagação de evento muito longa, podendo ser necessário até N rodadas de testes para que um novo evento seja conhecido por todos os nodos participantes do sistema.

Este trabalho apresenta uma abordagem para assinalamento de testes alternativa para algoritmo DiVHA [1], onde o número de testes por rodada é significativamente menor. Esta nova abordagem foi denominada MoDiVHA (*Modified Distributed Virtual Hypercube Algorithm*). O MoDiVHA mantêm intactas as propriedades principais do DiVHA, em particular a latência no pior caso, de forma que pode ser usado em sua substituição.

Implementações de algoritmos de diagnóstico distribuído, de forma geral, usam a troca de mensagens em rede como ferramenta para testar e compartilhar informações sobre o estado de um ou mais nodos. Desta forma, a otimização da quantidade de testes e,

consequentemente, dos *links* para troca de informações, é importante pois pode representar uma redução significativa de recursos de rede alocados para realização do diagnóstico.

O algoritmo foi implementado através de simulação e os resultados experimentais são apresentados mostrando o funcionamento do algoritmo para diversos tamanhos de sistemas e configurações de falhas. Foram conduzidas três séries de simulações. A primeira simulação mostra o número de testes necessários sem considerar a ocorrência de eventos e/ou nodos falhos. O resultado deste experimento é então comparado com os dados obtidos de experimento similar baseado no algoritmo DiVHA. A segunda série de experimentados apresentados considera diversas configurações de falhas e computa o número de testes necessários para cada uma delas. Finalmente, o terceiro experimento tem como objetivo determinar a latência do MoDiVHA para uma série de situações de falhas geradas aleatoriamente.

O restante deste artigo está organizando da seguinte maneira. A sessão 2 apresenta os principais modelos de diagnóstico distribuído em nível de sistema. Na sessão 3 o algoritmo MoDiVHA é especificado. Os resultados de simulações e uma avaliação empírica são apresentados na sessão 4. A sessão 5 conclui este trabalho.

2. Trabalhos Relacionados

O primeiro modelo de diagnóstico foi o PMC, cujo nome deriva das iniciais dos autores [7]. O modelo PMC assume a existência de um observador central que coleta todos os resultados de testes realizados pelos nodos, analisa esses resultados e então determina os estados dos nodos do sistema. Os testes entres os nodos são fixados previamente e o conjunto de testes possibilita que o diagnostico correto do sistema seja possível. Além do diagnóstico, os autores também definem a *diagnosticabilidade* da seguinte maneira: um sistema é *t-diagnosticável* se dada uma síndrome, o supervisor central completa o diagnóstico se o número de unidades falhas não exceder t.

Ao contrário do modelo PMC, o Adaptive-DSD (BIANCHINI; BUSKENS, 1991, 1992) realiza diagnóstico distribuído e adaptativo, de forma que não necessita de uma unidade central para a realização do diagnóstico e a configuração dos testes é determinada dinamicamente. No algoritmo Adaptive-DSD cada nodo testa outros nodos até encontrar um nodo não-falho, do qual colhe dados para atualizar suas informações de diagnóstico. Em termos de quantidade de testes, o algoritmo Adaptive-DSD é considerado eficiente sendo executados no máximo N testes por rodada. Quanto à latência, podem ser necessárias até N rodadas para que todos os nodos realizem o diagnóstico de um evento [1].

O algoritmo Hi-ADSD (DUARTE Jr.; NANYA, 1998) realiza os testes de forma hierárquica utilizando o conceito de *clusters*. A figura 2 mostra um sistema de oito nodos organizado em *clusters*. Durante a execução dos testes os nodos são agrupados em *clusters*, cujo tamanho é sempre uma potência de 2. Um nodo inicialmente realiza testes em *clusters* de tamanho 2 (2^1), nas próximas rodadas os *clusters* têm tamanho 4 (2^2), 8 (2^3) e assim sucessivamente até atingir tamanho $N/2(2^{(log_2 N)-1})$. No pior caso o número de testes executados pelo Hi-ADSD [9] é de $N^2/2$. A latência é de $log_2 N$ rodadas de testes.

O Algoritmo DiVHA (*Distributed Virtual Hypercube Algorithm*) [1], determina um grafo de testes $T(S)$ com topologia baseada em um hipercubo. O algoritmo de

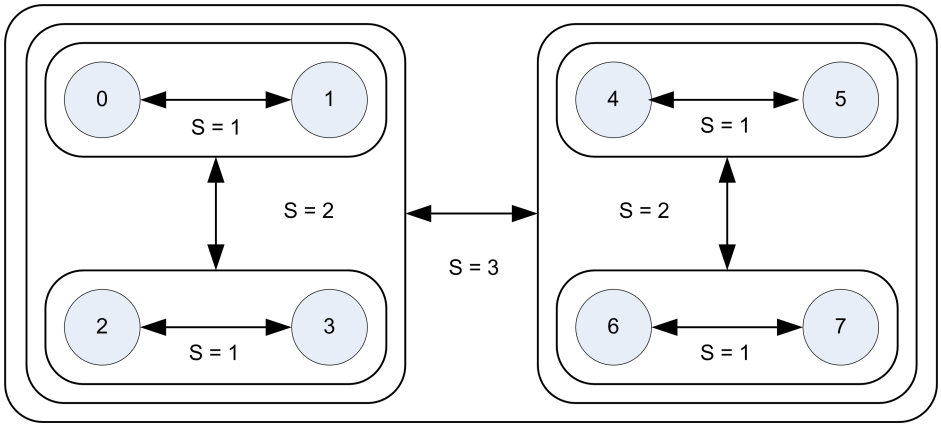

Figura 2. Sistema de 8 nodos agrupados em clusters.

diagnóstico utiliza o algoritmo DiVHA para atribuir os testes e determinar o fluxo de informação no sistema [1]. Na existência de nodos falhos testes adicionais são incluídos de forma a preservar o diâmetro logarítmico do hipercubo. O algoritmo DiVHA, da mesma forma que o algoritmo Hi-ADSD, utiliza o conceito de *clusters* para agrupar os nodos e organizar os testes. A figura 2 ilustra como o agrupamento em *clusters* é estruturado.

A construção de $T(S)$ pode ser dividida em duas fases. Na primeira fase $T(S)$ recebe todas as arestas correspondentes ao hipercubo dos nodos sem falha, de forma que cada nodo sem falha fica responsável por testar seus $log_2 N$ nodos vizinhos. A ocorrência de nodos falhos pode exigir a inclusão de arestas de teste adicionais. A inclusão de um teste adicional entre um par de nodo qualquer i e j baseia-se na distância no hipercubo entre este par de nodos bem como a atual distância dos mesmos no grafo de testes $T(S)$. A figura 3 mostra os assinalamentos de testes em um sistema de 8 nodos. Tome o nodo 7 como exemplo. O nodo 1 é escolhido como testador porque está a distância 2, enquanto 0 está a distância 3.

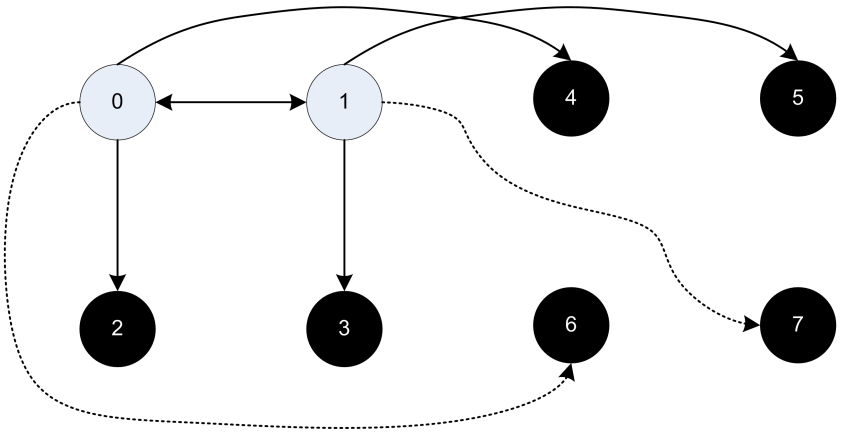

Figura 3. Assinalamento de testes do DiVHA; linha pontilhada indica teste adicional.

O DiVHA permite que o algoritmo de diagnóstico determine o estado de todos os nodos do sistema em no máximo $log_2 N$ rodadas de testes. O número máximo de testes

por rodada é de $N \times log_2 N$. A latência média do algoritmo DiVHA é, em geral, menor que sua latência máxima de $log_2 N$. Resultados experimentais indicam que a maior parte dos eventos são diagnosticados em cerca de $(log_2 N)/2$ rodadas [1].

3. MoDiVHA

O algoritmo MoDiVHA tem como função principal determinar quais serão os testes que cada um dos nodos devem executar. De forma similar ao algoritmo DiVHA, os nodos e os testes são representados por um grafo cuja a topologia é baseada em um hipercubo. No entanto, o grafo de testes produzido pelo MoDiVHA possui menos arestas que o equivalente gerado pelo DiVHA. A intuição é incluir no grafo de testes apenas as arestas estritamente necessárias, de forma a reduzir a redundância de caminhos entre os pares de nodos mas, ao mesmo tempo, garantir as propriedades que permitem que o diagnóstico do sistema seja realizado. Na sessão de resultados experimentais é realizado um comparativo da quantidade de testes necessários para diferentes tamanhos de sistema.

3.1. O Algoritmo MoDiVHA

A topologia do sistema segue a estruturada de um hipercubo e é representada pelo grafo $H(S)$, conforme a figura 4. Para contextualizar e exemplificar o funcionamento do algoritmo usaremos um sistema simples, composto por apenas 8 nodos. Os vértices de $H(S)$ correspondem aos nodos e as arestas direcionadas representam as conexões virtuais entre nodos, formando uma rede *overlay*. A distância mínima entre o nodo i e um nodo qualquer j é definida por $h(i, j)$, e é tamanho do menor caminho entre o nodo i e o nodo j em $H(S)$.

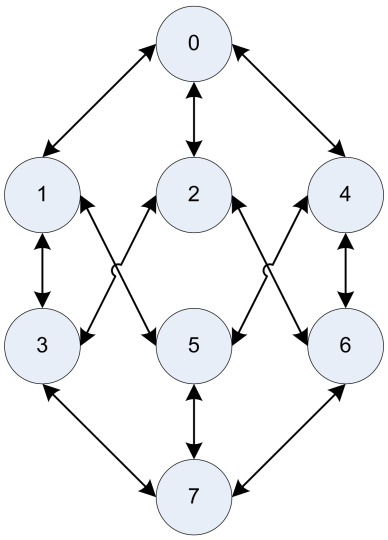

Figura 4. Grafo $H(S)$ para um sistema de 8 nodos.

O MoDiVHA faz uso de $H(S)$ para determinar os testes que devem ser realizados por todos os nodos sem-falha do sistema. O resultado desta operação é o grafo $T(S)$. Uma aresta direcionada de $T(S)$ entre, por exemplo, os nodos i e j indica que o nodo i testa o nodo j. Na ocorrência de um evento (falha ou recuperação de um nodo), todos os nodos do sistema devem recalcular $T(S)$ de forma a, novamente, determinar seus testes

considerando os testes realizados pelos demais nodos e pelas informações que consegue obter dos nodos testados. Na condição do sistema estar estabilizado $T(S)$ é igual para todos os nodos.

A construção do grafo $T(S)$ é realizada de forma a garantir que cada nodo falho seja testado por pelo menos um nodo sem-falha, e que a distância entre quaisquer nodos em $T(S)$ seja de no máximo $log_2 N$. A manutenção da distância garante a latência de diagnóstico. Desta forma uma aresta entre os nodos i e j é adicionada ao grafo $T(S)$ ou quando não existe um caminho entre i e j, ou quando o caminho que existe é maior que caminho mínimo esperado. O algoritmo verifica progressivamente o atendimento das distâncias mínimas entre os nodos. Inicialmente são verificados os nodos conectados por distância mínima 1, aumentando esta distância até alcançar $log_2 N$. Quando a verificação da distância mínima entre dois nodos falha, é inserida uma aresta ligando diretamente estes nodos.

A figura 5 mostra aspecto final de $T(S)$ para um sistema de 8 nodos, onde o nodo de índice 3 está falho. Os testes em linha contínua tem origem das conexões de $H(S)$. Neste exemplo o nodo falho compromete a condição do caminho mínimo, pois não existe rota de tamanho menor ou igual a $log_2 N$ entre os nodos 1 e 7. Desta forma testes adicionais devem ser incluídos, indicados pela linha tracejada.

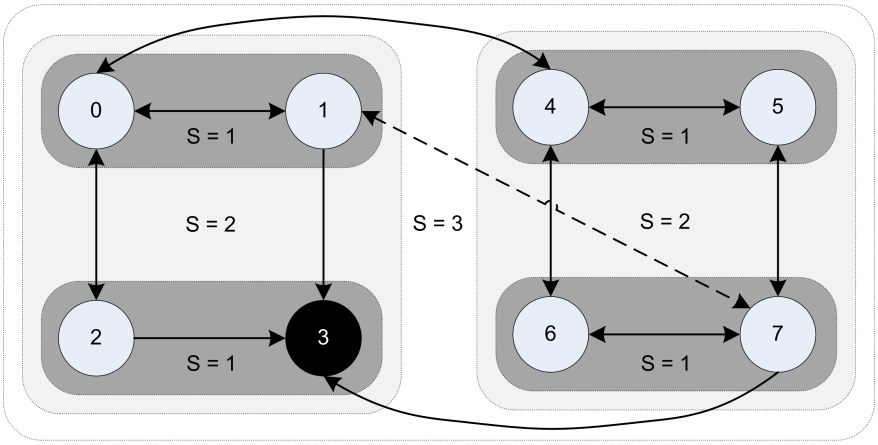

Figura 5. Grafo $T(S)$ representando o assinalamento de testes.

O MoDiVHA utiliza o conceito de *clusters* [9] para agrupar os nodos. O tamanho de um *cluster* é definido pelo índice s e determinado por uma potência de 2 (2^s). Um *cluster* de índice $s = 3$ tem tamanho igual a 8 (2^3) e é composto por dois *clusters* de índice 2 e, por conta da recursividade, contêm quatro *clusters* de índice 1. A figura 5 ilustra como esse agrupamento é realizado para um sistema de 8 nodos.

A lista de nodos de um determinado *clusters* é calculada pela função $C(i, s)$ [9] e pode ser definida recursivamente pela expressão: $C_{i,s} = C_{j,s-1} \cup C_{i,s-1}$, $j = i \oplus 2^{s-1}$ e $C_{i,1} = j$, $j = i \oplus 1$. A tabela da figura 6 lista os possíveis resultado da $C(i, s)$ para nosso exemplo de sistema.

O pseudo-código do MoDiVHA é apresentado na figura 7. O algoritmo começa sua operação reinicializando $T(S)$ para o estado no qual o número de arestas é 0, conforme indicado pela linha 2. Em seguida, nas linhas 3 a 6, são executados quatro *loops*

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

Figura 6. Resultado da função $C(i,s)$ para um sistema de 8 nodos.

em cascata, onde cada *loop* varre uma determinada propriedade do modelo do sistema. O laço mais externo percorre progressivamente todos os possíveis índices de *clusters* para o sistema S. O laço seguinte varre todas as possíveis distâncias d em $H(S)$ para o índice de *cluster* corrente.

```
1. MoDiVHA(system S)
2.     T(S) initially does not have any edge
3.     FOR s = 1 TO log2(N) DO
4.         FOR d = 1 TO s DO
5.             FOREACH fault free node i = [0 ... N-1] DO
6.                 FOREACH node j that belongs C(i, s) DO
7.                     IF h(i, j) == d AND d(i, j) > s THEN
8.                         add edge(i, j) to T(s)
```

Figura 7. O algoritmo MoDiVHA.

O terceiro *loop* faz uma busca por todos os nodos i sem falha do sistema S e, finalmente, o último *loop* percorre os nodos j pertencentes a $C(i,s)$ atual. Desta forma são verificadas as arestas entre o nodo 0 e nodos pertencentes ao *cluster* $C(0,s)$, entre o nodo 1 e os nodos pertencentes ao *cluster* $C(1,s)$, e assim por diante até o nodo $N-1$ e os nodos pertencentes ao *cluster* $C(N-1,s)$.

A linha 7 verifica as condições necessária para inclusão de uma nova aresta em $T(S)$ e pode ser lida da seguinte maneira: ***SE** a distância de i para j no hipercubo for igual ao d corrente **E** a distância de i para j no grafo T for maior que o s corrente **ENTÃO***. A primeira condicional assegura que as arestas dos *clusters* de menor índice são verificadas antes que as arestas dos *clusters* de índice maior. A segunda condicional garante que uma aresta é adicionada entre o nodo i e o nodo $j \in C(i,s)$ sempre que não existir um caminho do nodo i para o nodo j em $T(S)$ com distância $\leq s$.

A função $d(i,j)$ retorna a distância entre os nodos i e j em $T(S)$ e corresponde ao tamanho do menor caminho entre o nodo i e o nodo j em $T(S)$. Se não existir um caminho entre um par de nodos a distância é tida como infinita.

Satisfeitas as condições a inclusão da aresta de i para j é realizada pela operação da linha 8 do pseudo-código. Esta nova aresta é um assinalamento que indica que nodo i deve testar no nodo j.

A redução de testes obtida por este algoritmo é consequência do fato que, diferentemente do DiVHA, o MoDiVHA submete, inclusive as arestas de $H(S)$ ao teste da existência de um caminho mínimo. Observe a figura 8, considere que o MoDiVHA está no exato momento onde verifica a necessidade de incluir uma aresta de teste entre os no-

dos 1 e 5. Pelo grafo $H(S)$ este teste deveria existir, mas como é possível encontrar um caminho entre 1 e 5 de tamanho menor ou igual a 3 ($s = 3$) o teste é ignorado. Neste caso o nodo 1 consegue obter informação sobre o nodo 5 pelo caminho $1 \rightarrow 0 \rightarrow 4 \rightarrow 5$.

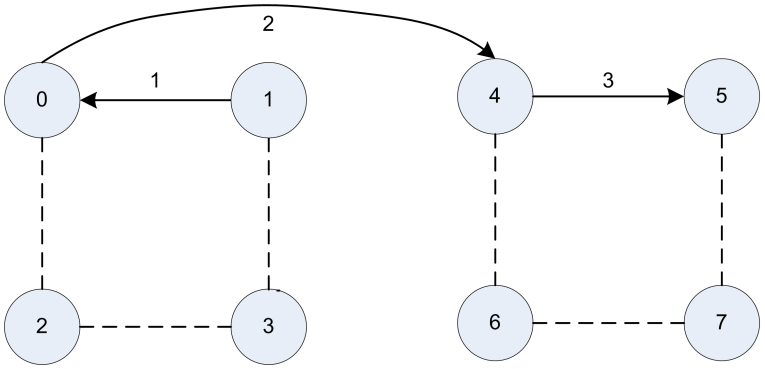

Figura 8. Construção do grafo $T(S)$.

Em um sistema onde todos os nodos estão sem falha o grafo $T(S)$ resultante do algoritmo DiVHA é um hipercubo completo, uma cópia exata de $H(S)$. O mesmo não se aplica ao MoDIVHA, conforme ilustrado pela figura 9.

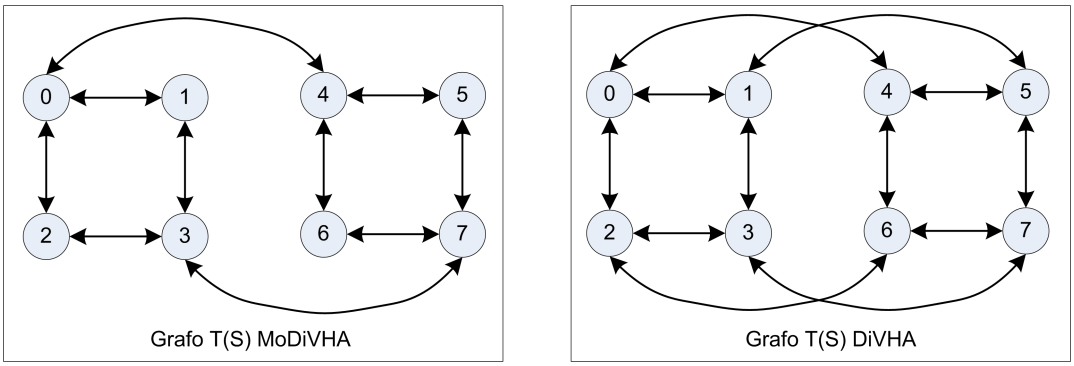

Figura 9. Grafos $T(S)$ gerados pelo MoDiVHA e DiVHA para um sistema de 8 nodos sem falhas.

4. Resultados Experimentais

Nesta seção são apresentados resultados obtidos de experimentos realizados com algoritmo MoDiVHA. A implementação do algoritmo foi realizada em linguagem de programação C e os experimentos foram conduzidos utilizando a biblioteca de simulação de eventos discretos SMPL (*Simple Portable Simulation Language*) [10].

Foram realizados três séries de simulações. A primeira simulação mostra o número de testes necessários sem considerar a ocorrência de eventos e/ou nodos falhos. O resultado deste experimento é então comparado com os dados obtidos de experimento similar baseado no algoritmo DiVHA. A segunda série de experimentados apresentados considera diversas configurações de falhas e computa o número de testes necessários para cada uma delas. Finalmente, o terceiro experimento tem como objetivo determinar a latência do MoDiVHA para uma série de situações de falhas geradas aleatoriamente.

4.1. Número de Testes Necessários

O propósito deste experimento é comparar o número de testes necessários em um sistema executando os algoritmos de assinalamento MoDiVHA e DiVHA. Foram simulados sistemas de 2 até 1024 nodos. A tabela da figura 10 reproduz os resultados obtidos.

N	Testes DiVHA	Testes MoDiVHA	Economia de Testes
2	2	2	0%
4	8	8	0%
8	24	20	17%
16	64	48	25%
32	160	102	36%
64	384	220	43%
128	896	458	49%
256	2048	944	54%
512	4608	1926	58%
1024	10240	3922	62%

Figura 10. Tabela comparativa; *DiVHA vs MoDiVHA*.

A primeira coluna da tabela indica o tamanho do sistema simulado, na segunda coluna está a quantidade de testes exigidos pelo DiVHA para cada tamanho de sistema. Em seguida a quantidade de testes referente ao MoDiVHA e, na última coluna, temos a economia de testes percentual do MoDiVHA em relação ao DiVHA.

Em sistemas pequenos, com poucos nodos, ambos algoritmos executam o mesmo número de testes. Conforme o tamanho do sistema aumenta o algoritmo MoDiVHA começa a apresentar uma redução significativa, e crescente, do número de testes. Em um sistema de 1024 nodos o algoritmo MoDiVHA executa 62% menos testes que o DiVHA. A figura 11 representa graficamente estes resultados.

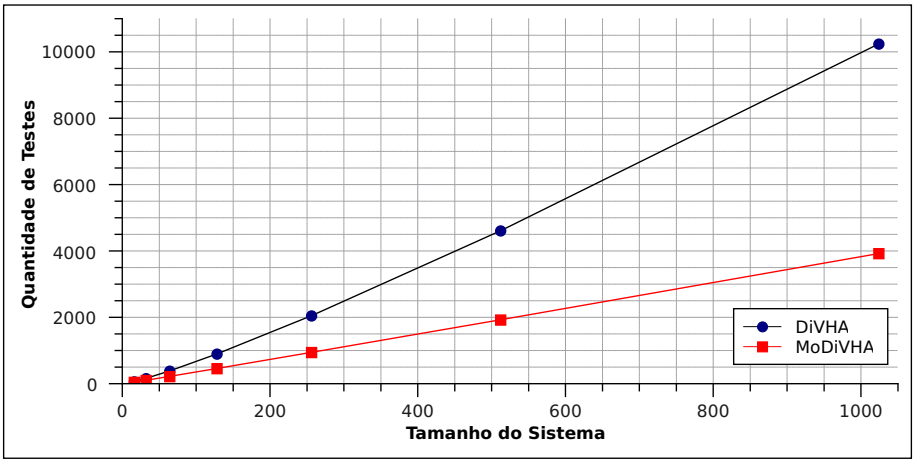

Figura 11. Número de testes necessários para diversos tamanhos de sistema.

4.2. Número de Testes Considerando Falhas

O segundo experimento considera um sistema de 256 nodos onde foram simuladas diversas configurações falhas. O propósito é medir o número de testes necessários na

ocorrência de 1 até $N - 1$ falhas em nodos aleatórios. O gráfico da figura 12 apresenta os resultados obtidos pela simulação. O eixo X representa a quantidade de nodos falhos e o eixo Y apresenta o número de testes que foram executados para cada situação de falha simulada. Estes resultados mostram a tendência da quantidade de testes diminuir conforme o número de nodos falhos aumenta.

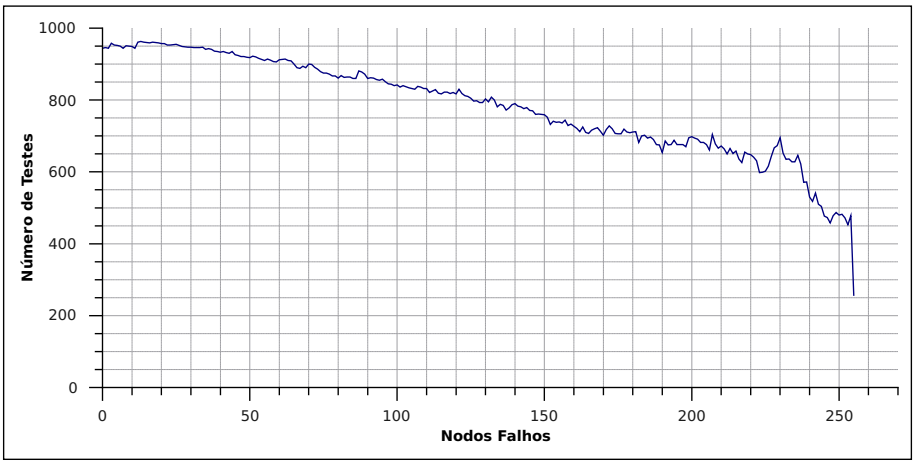

Figura 12. Número de testes *versus* falhas para um sistema com 256 nodos.

4.3. Latência para Detecção de Eventos

A latência é o número de rodadas de testes necessárias para que todos nodos sem-falha realizem o diagnostico de um evento. O objetivo do terceiro experimento é verificar a latência para detectar um evento sob diferente situações de falhas. O sistema simulado é composto por 128 nodos inicialmente sem-falhas. Foram introduzidos neste sistema falhas em nodos aleatórios. A latência de propagação destes eventos foram monitoras e registradas tanto para o algoritmo MoDIVHA quanto para o DiVHA, conforme exposto nas figuras 13 e 14 respectivamente.

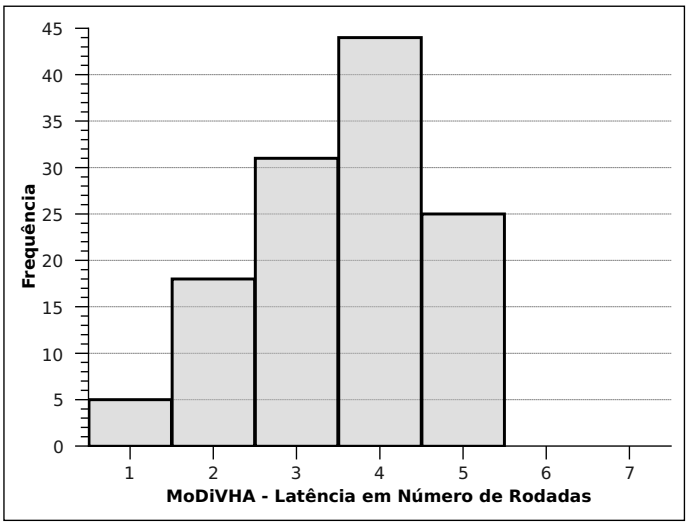

Figura 13. Latência MoDiVHA para um sistema com 128 nodos.

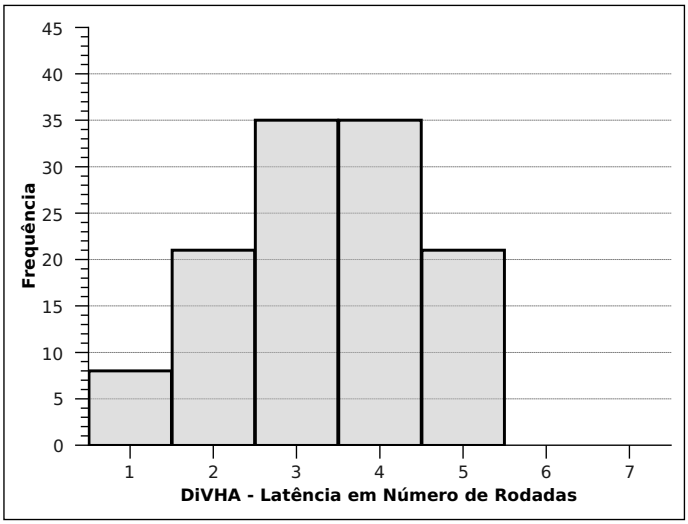

Figura 14. Latência DiVHA para um sistema com 128 nodos.

O histograma na figura 13 mostra a distribuição de frequência da latência observada para os eventos simulados usando o MoDiVHA. A maioria dos eventos foi diagnosticada em um número de rodadas abaixo da latência máxima teórica de $log_2 N$ rodadas, mas, na média, acima dos valores observados para o algoritmos DiVHA (figura 14). Este comportamento já era esperado e é consequência direta da redução de arestas no grafo de testes $T(S)$. As arestas de $T(S)$, além de indicarem o assinalamento dos testes, também são usadas para o determinar a rota de troca de informações entre os nodos do sistema.

5. Conclusões

Este trabalho apresentou uma estratégia de assinalamento de testes hierárquica que permite o diagnóstico escalável de sistemas distribuídos. Esta nova estratégia foi denominada MoDiVHA sendo baseada naquela do algoritmo DiVHA [1], mas visa, principalmente, a redução dos recursos de sistema necessários para a realização do diagnóstico. Os dados experimentais demostram que quanto maior o sistema maior será a redução do número de teste necessários pelo MoDiVHA em relação ao DiVHA. Este resultado evidencia o fato que o MoDiVHA é escalável, tornando-o uma opção para aplicações de diagnóstico distribuído quando é necessário reduzir o número de testes.

Trabalhos futuros incluem a aplicação da estratégia distribuída hierárquica de testes MoDiVHA em redes *Ad hoc*, que implicará em mudanças inclusive no modelo de sistema considerado [11]. E está sendo considerada a aplicação do MoDiVHA para testes hierárquicos em serviços distribuídos na Internet [12]; neste caso um dos desafios é adaptar o modelo às características do sistema.

Referências

[1] Luis C. E. Bona, Elias P. Duarte Jr., Keiko V. O. Fonseca, Samuel L. V. Mello, "HyperBone: A Scalable Overlay Network Based on a Virtual Hypercube", *The 8th International Symposium on Cluster Computing and the Grid (CCGRID'2008)*, pp. 58-64, Lyon, France, 2008.

[2] N. Blum, P. Jacak, F. Schreiner, D. Vingarzan, and P. Weik, "Towards Standardized and Automated Fault Management and Service Provisioning for NGNs", *Journal of Network and Systems Management*, Vol. 16, No. 1, 2008.

[3] Hui Yang, Mourad Elhadef, Amiya Nayak, Xiaofan Yang, "An evolutionary approach to system-level fault diagnosis", *IEEE Congress on Evolutionary Computation*, 2009.

[4] Elias P. Duarte Jr., Luiz C. P. Albini, Alessandro Brawerman, Andre L. P. Guedes, "A Hierarchical Distributed Fault Diagnosis Algorithm Based on Clusters with Detours", *The 6th IEEE Latin American Network Operations and Management Symposium (LANOMS)*, pp. 1-6, Punta del Este, Uruguay, 2009.

[5] S.-Y. Hsieh and Y.-S. Chen, "Strongly Diagnosable Systems Under the Comparison Model", *IEEE Transactions on Computers*, Vol. 57, No. 12, 2008.

[6] X. Yang, and Y. Y. Tang, "Efficient Fault Identification of Diagnosable Systems under the Comparison Model", *IEEE Transactions on Computers*, Vol. 56, No. 12, 2007.

[7] Elias P. Duarte Jr., Roverli P. Ziwich, Luiz C. P. Albini, "A Survey of Comparison-Based System-Level Diagnosis", *ACM Computing Surveys*, ISSN 0360-0300, Aceito para publicação.

[8] R.P. Bianchini, and R. Buskens, "An Adaptive Distributed System-Level Diagnosis Algorithm and Its Implementation", *Proc. FTCS-21*, pp. 222-229, 1991.

[9] E. P. Duarte Jr., and T. Nanya, "A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm", *IEEE Transactions on Computers*, Vol. 47, pp. 34-45, No. 1, Jan 1998.

[10] M.H. MacDougall, "Simulating Computer Systems: Techniques and Tools", *The MIT Press*, Cambridge, MA, 1987.

[11] Andréa Weber, Alexander Robert Kutzke, Stefano Chessa, "Energy-Aware Test Connection Assignment for the Diagnosis of a Wireless Sensor Network", *The 5th IEEE Latin American Dependable Computing Symposium (LADC)*, São José dos Campos, SP, 2011.

[12] Alan Nakai, Edmundo Madeira, Luiz Eduardo Buzato, "Load Balancing for Internet Distributed Services using Remote Resource Reservation", *The 5th IEEE Latin American Dependable Computing Symposium (LADC)*, São José dos Campos, SP, 2011.