Um Algoritmo para Diagnóstico Distribuído com Informações Datadas

Elias Procópio Duarte Jr.
Alessandro Brawerman
Luiz Carlos P. Albini

Universidade Federal do Paraná, Dept. Informática
Cx. Postal 19081 – Curitiba – 81531-990 PR – Brasil
Fone: 041-267-5244 Fax: 041-267-6874
e-mail: elias@inf.ufpr.br, {ab,lcpa}@pos.inf.ufpr.br

Resumo

Neste trabalho apresentamos um novo algoritmo para diagnóstico de sistemas de grafo completo, Hi-ADSD with Timestamps. Neste algoritmo todos os nodos mantém um timestamp para o estado de cada nodo do sistema. Este timestamp é implementado como um contador de trocas de estado dos diversos nodos. Desta forma, cada testador pode obter informação sobre um nodo específico do sistema através de mais de um nodo testado sem causar inconsistências, isto é, sem confundir um estado mais antigo com outro mais recente. Nodos testam clusters progressivamente maiores cujo tamanho é sempre uma potência de dois. O número de testes executados no total, a cada $log_2 N$ rodadas de testes é de, no máximo, $N log_2 N$ testes. Ao testar um nodo sem falha, o testador obtém informação de diagnóstico sobre $N/2$ nodos, para um sistema de N nodos. Um dos critérios mais importantes para avaliar um algoritmo de diagnóstico a nível de sistema é sua latência, isto é, o tempo que os nodos sem-falha do sistema necessitam para completar o diagnóstico de todo o sistema. Resultados de simulação do algoritmo para redes de 64 nodos e 512 nodos confirmam que esta estratégia reduz significativamente a latência média do algoritmo quando comparado a outros previamente publicados. Entretanto, mostramos que a latência no pior caso ainda é de $log_2^2 N$ rodadas de testes. Apesar deste fato, além do overhead da manutenção e transferência de timestamps, a simulação do algoritmo indica que se trata de um boa opção para a implementação prática de diagnóstico.

Abstract

In this paper we present a new system-level diagnosis algorithm for fully-connected networks, Hi-ADSD with Timestamps. In this algorithm every node keeps a timestamp for the state of each node in the system. This timestamp is implemented as a counter, which is incremented every time a node changes its state. In this way, each tester may get information about a given node in the system from more than one tested node without causing any inconsistencies, i.e. without taking an older state for a newer one. Nodes test progressively larger clusters which have a number of nodes that is a power of two. The number of tests executed each $log_2 N$ testing rounds is, at most, $N log_2 N$ tests. When a fault-free node is tested, the tester gets diagnostic information about $N/2$ nodes, for a system of N nodes. One of the most important parameters to evaluate a system-level diagnosis algorithm is its latency, i.e. the time it takes for every fault-free node to complete the diagnosis of the system. Simulation results for 64-node and 512-node networks confirm that this approach reduces significantly the average latency of the algorithm when it is compared to other similar results. However, we show that the latency in the worst case is still $log_2^2 N$ testing rounds. Nevertheless, even considering the worst case of the latency and the overhead of keeping and transfering timestamps, the algorithm simulation shows that this algorithm is a good option for practical diagnosis implementation.

Palavras Chaves: Diagnóstico a Nível de Sistema, Diagnóstico Distribuído, Diagnóstico Adaptativo, Monitoração de Redes e Tolerância a Falhas.

1 Introdução

Considere um sistema composto de N nodos, cada um dos quais pode assumir um de dois estados, *falho* ou *sem-falha*. Assuma que o grafo que representa o sistema é completo, isto é, existe um *link* de comunicação ligando qualquer par de nodos, e os *links* não falham. Este modelo pode ser utilizado tanto para redes físicas, por exemlo, as redes locais baseadas em difusão, como para sistemas lógicos.

A principal meta de um algoritmo de diagnóstico a nível de sistema é permitir que os nodos sem-falha obtenham informação sobre o estado de todos os nodos do sistema [1, 2]. Assume-se que os nodos sem-falha são confiáveis, isto é, eles são capazes de executar testes e relatar seus resultados com precisão [3, 4]. Estes testes são executados em intervalos pré-definidos, por exemplo, 30 segundos. Por outro lado, os nodos falhos simplesmente deixam de funcionar, não respondendo aos testes a que são submetidos.

O grafo de testes do sistema é definido como um grafo direcionado no qual os vértices são os nodos do sistema e existe uma aresta direcionada do nodo i para o nodo j se o nodo i testar o nodo j.

Um algoritmo de diagnóstico a nível de sistema é chamado adaptativo quando o conjunto de testes que um nodo executa é dinâmico, sendo escolhido a cada rodada de testes de acordo com os resultados de testes anteriores [7].

Os primeiros algoritmos de diagnóstico a nível de sistema assumiam a existência de um monitor, ou seja um nodo que recebia o resultado de todos os testes executados e efetuava o diagnóstico. Por outro lado, os algoritmos que não assumem a existência do monitor, são chamados distribuídos [8]; neles os próprios nodos que realizam os testes fazem o diagnóstico completo do sistema.

Três critérios são geralmente utilizados para avaliar estes algoritmos: o número de testes executados, a quantidade de informação transferida em cada teste, e a latência, isto é, o tempo necessário para que os nodos sem-falha completem o diagnóstico do sistema.

O primeiro algoritmo adaptativo *e* distribuído é o algoritmo Adaptive-DSD (*Adaptive Distributed System-Level Diagnosis*) [9]. Aos nodos que executam o algoritmo são assinalados identificadores sequênciais, de 0 a $N-1$. Os nodos executam testes sequencialmente até encontrarem um nodo sem-falha, isto é, o nodo i testa o nodo $i+1$, se este estiver falho, o nodo i testa o nodo $i+2$ e assim por diante. Desta forma o grafo de testes do sistema é um anel, como mostrado na figura 1.

Quando um nodo sem-falha é testado, o nodo testador obtém informações de diagnóstico do nodo testado. Estas informações incluem os resultados dos testes realizados pelo nodo testado, além das informações de diagnóstico por ele obtidas. A latência deste algoritmo é definida em termos de *rodadas de testes*. Uma rodada de testes é definida como o período de tempo no qual todos os nodos sem-falha do sistema testam um nodo sem-falha.

O algoritmo Adaptive-DSD tem a propriedade de que cada nodo é testado exatamente uma vez a cada rodada de testes, e cada nodo sem-falha testa exatamente um outro nodo sem-falha por rodada de testes. Entretanto, se apenas um nodo estiver sem-falha e $N-1$ nodos estiverem falhos, o nodo sem-falha testará todos os $N-1$ nodos falhos. O número total de testes, considerando todos os nodos, é N a cada rodada de testes, e a latência do algoritmo é N rodadas de testes.

Em 1998 Duarte e Nanya [5] propuseram o algoritmo Hi-ADSD (*Hierarchical Adap-*

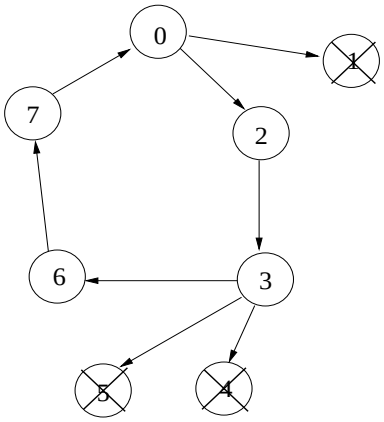

Figura 1: Exemplo de conjunto de testes executados por nodos rodando Adaptive-DSD.

tive Distributed System-Level Diagnosis), no qual os nodos são agrupados em *clusters* para propósito de testes. Clusters são conjuntos de nodos. O número de nodos de um cluster, isto é, seu tamanho, é sempre uma potência de dois. O sistema como um todo é considerado um cluster com N nodos.

Um cluster de n nodos n_j, ..., n_{j+n-1}, onde j MOD $n = 0$, e n é uma potência de 2, é recursivamente definido como um nodo, no caso de $n = 1$; ou como a união de 2 clusters, um contendo os nodos n_j, ..., $n_{(j+n/2)-1}$ e o outro contendo nodos $n_{j+n/2}$, ..., n_{j+n-1}. A figura 2 mostra um sistema com 8 nodos organizados em clusters executando o algoritmo Hi-ADSD.

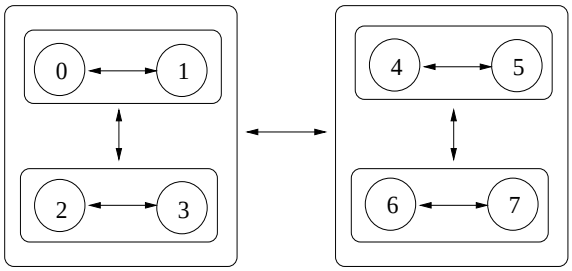

Figura 2: Nodos executando testes do algoritmo Hi-ADSD.

No algoritmo Hi-ADSD cada nodo executa seus testes começando pelo menor cluster, prosseguindo para o cluster com dois nodos, e assim por diante, até testar o maior cluster. Entretanto, os testes são executados assincronamente, ou seja, em uma determinada rodada de testes, cada nodo pode estar testando um cluster de tamanho diferente dos outros clusters. Para obter informação de diagnóstico sobre um determinado cluster, o nodo testador deve executar testes naquele cluster até encontrar um nodo sem-falha, ou testar todos os nodos do cluster, caso todos estiverem falhos. Se um nodo sem-falha é encontrado, o testador obtém informação de diagnóstico sobre todo o cluster do nodo testado. Por exemplo, na figura 2, quando o nodo 0 testar o nodo 4 e este estiver sem-falha, o nodo 0 obterá informação de diagnóstico sobre todo o cluster do nodo 4, ou seja, sobre o nodo 4, o nodo 5, o nodo 6 e o nodo 7.

A latência do algoritmo Hi-ADSD é de, no máximo, $log^2 N$ por rodada de testes. Todos

os logaritmos deste artigo possuem base 2. O número máximo de testes por rodada pode chegar a $N^2/4$ testes. Isto ocorre quando $N/2$ nodos falhos estão em um único cluster, e os $N/2$ restantes, estão testando este cluster ao mesmo tesmpo.

No trabalho proposto por Duarte, Brawerman e Albini [6], é apresentado o algoritmo *Hi-ADSD with Detours*, que utiliza a mesma estratégia de organizar os nodos em clusters introduzida pelo Hi-ADSD, mas possui uma estratégia de testes diferente, que garante que o número máximo de testes executados a cada $logN$ rodadas de testes é no máximo $NlogN$. Neste algoritmo uma rodada de testes é definida como o período de tempo no qual todos os nodos sem-falha realizam os testes necessários em um único cluster. A latência do algoritmo não é maior que log^2N, rodadas de testes.

Neste trabalho apresentamos um novo algoritmo, *Hi-ADSD with Timestamps*. Considere a distância de diagnóstico entre o nodo i e o nodo j definida como o número de nodos através dos quais a informação de diagnóstico do nodo j deve passar até atingir o nodo i, incluindo o próprio nodo j, quando todos os nodos estão sem-falha. Neste algoritmo, quando um nodo sem-falha testa outro nodo sem-falha, o testador obtém informação sobre o conjunto de nodos cuja distância de diagnóstico começando pelo nodo testado é de até $logN - 1$. Este conjunto tem sempre $N/2$ nodos.

Desta forma, um nodo i pode obter informação de diagnóstico sobre um nodo j a partir de vários outros nodos. Para determinar qual informação é mais recente, os nodos mantém um contador de estados [16, 17].

Resultados de simulação deste algoritmo mostram que sua latência média é significativamente menor que a dos algoritmos anteriores. Para redes de 512 nodos, simulações dos algoritmos *Hi-ADSD* e *Hi-ADSD with Detours* apresentaram latência média de 40 rodadas de testes para diagnosticar um evento, enquanto que o novo algoritmo apresentou latência média de 12 rodadas de testes.

O restante deste artigo está organizado da seguinte maneira. Na seção 2 é apresentada a estratégia de testes do algoritmo, que é especificado na seção 3. Na seção 4 a latência e o número máximo de testes necessários são examinados. A seção 5 mostra resultados experimentais da simulação do algoritmo para redes de diversos tamanhos. E esta é seguida pela conclusão na seção 6.

2 Estratégia para a Execução de Testes

Nesta seção apresentamos a estratégia para a execução de testes utilizada pelo algoritmo *Hi-ADSD with Timestamps*. Esta estratégia foi inicialmente proposta em [6]. Ela garante que o número máximo de testes executado a cada $logN$ rodadas é, no máximo, $NlogN$.

Considere a distância de diagnóstico entre o nodo i e o nodo j definida como o número de nodos através dos quais a informação de diagnóstico sobre o nodo j deve passar até atingir o nodo i, incluindo o próprio nodo j, quando todos os nodos do sistema estão sem-falha. Por exemplo, na figura 3 a distância de diagnóstico entre o nodo 0 e o nodo 7 é igual a 3.

No algoritmo *Hi-ADSD with Timestamps*, a cada intervalo de testes um nodo testa um único cluster. No primeiro intervalo, o cluster a ser testado possui tamanho igual a 1, ou seja, possui apenas 1 nodo. No segundo intervalo, o cluster a ser testado possui 2 nodos, no terceiro intervalo o cluster a ser testado possui 4 nodos e assim por diante, até

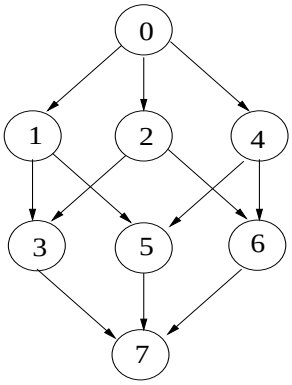

Figura 3: A distância de diagnóstico entre o nodo 0 e o nodo 7 é igual a 3.

o cluster de tamanho $N/2$ ser testado. Após testar este cluster, o cluster de tamanho 1 é testado novamente, e repete-se todo o processo.

Quando um nodo testa um determinado cluster e o nodo testado estiver falho, o testador não precisa executar mais testes naquele cluster no mesmo intervalo de testes, pois ele poderá obter as informações de diagnóstico sobre o restante do cluster através de nodos sem-falha situados fora deste cluster, utilizando-se de caminhos alternativos chamados *detours*. Esta situação é mostrada na figura 4, na qual o nodo 0 testa o nodo 4 falho, então utiliza-se de um *detour* através do nodo 1 para obter informação do nodo 5. Porém, se quando o testador voltar a testar este cluster ainda existirem nodos sobre os quais ele não conseguiu obter as informações de diagnóstico, o testador vai então executar mais testes neste cluster até obter as informações necessárias.

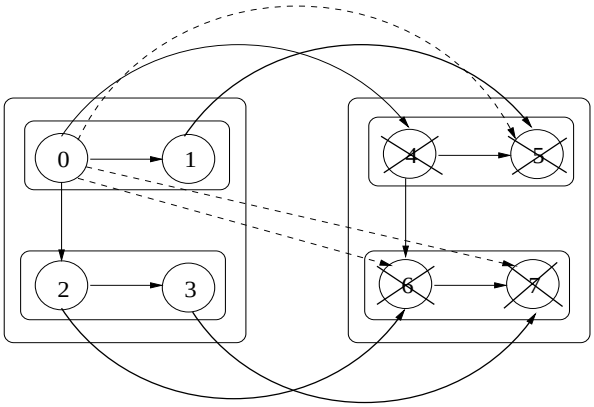

Figura 4: As linhas pontilhadas correspondem aos testes que o nodo 0 evita ao utilizar desvios.

A estratégia de testes do algoritmo *Hi-ADSD with Timestamps* garante que o número de testes executados no sistema diminui à medida em que aumenta o número de nodos falhos no sistema.

3 Hi-ADSD with Timestamps: Especificação do Algoritmo

Considere um sistema com N nodos, os quais podem assumir um de dois estados: *falho* ou *sem-falha*. Assuma que o grafo que representa o sistema é completo, isto é, existem *links* entre todos os pares de nodos, e os *links* não falham. Nodos sem-falha executam testes em intervalos de testes e reportam os resultados dos mesmos confiavelmente. No restante do artigo nos referimos a n_i também como nodo i.

Seja o *grafo de testes livre de falhas do sistema*, $T(S)$, definido como um grafo direcionado no qual os vértices são nodos do sistema S. Existe uma aresta do nodo a para o nodo b, se o nodo a testar o nodo b sem-falha no mais recente intervalo de testes no qual o nodo a testou o cluster ao qual o nodo b pertence. Quando todos os nodos do sistema estão sem-falha, $T(S)$ tem a forma de um hipercubo. A figura 5 exemplifica um grafo $T(S)$ para um sistema de 8 nodos.

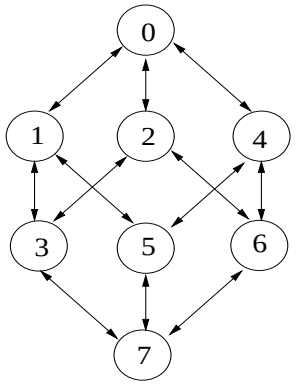

Figura 5: $T(S)$ para um sistema de 8 nodos.

Um *grafo de testes livre de falhas do nodo* i, TFF_i, é definido como um grafo direcionado no qual os nodos são nodos do sistema S e existe um caminho direcionado do nodo a para o nodo b, se o nodo a testa o nodo b e a distância de diagnóstico do nodo i para o nodo a é menor do que a distância de diagnóstico do nodo i para o nodo b. A figura 6 exemplifica TFF_0 para um sistema com 8 nodos.

A lista ordenada de nodos que podem ser atingidos pelo nodo i, em TFF_i, a partir do nodo p, com distância de diagnóstico até o nodo p menor ou igual a $s-1$, é chamada $c_{i,s,p}$. Em outras palavras, se o nodo i pode obter informação sobre o nodo j a partir do nodo p, e o caminho do nodo i para o nodo j passando pelo nodo p na TFF_i possui no máximo s arestas, então o nodo j pertence a $c_{i,s,p}$. Por exemplo, na figura 7 o nodo 4, o nodo 5, o nodo 6 e o nodo 7 pertencem a $c_{0,3,4}$.

No algoritmo, é possível que um nodo i obtenha informação de diagnóstico sobre um nodo j a partir de dois ou mais nodos, por exemplo, do nodo p e do nodo p'. Neste caso, é necessário garantir que o nodo i armazene somente a informação mais recente sobre o estado do nodo j. Como exemplo, na figura 7, o nodo 0 vai receber informação de diagnóstico sobre o nodo 5 através do nodo 1 e do nodo 4, pois o nodo 5 pertence a $c_{0,logN,1}$ e a $c_{0,logN,4}$. Portanto é necessário garantir que o nodo 0 armazene somente a informação mais recente sobre o nodo 5.

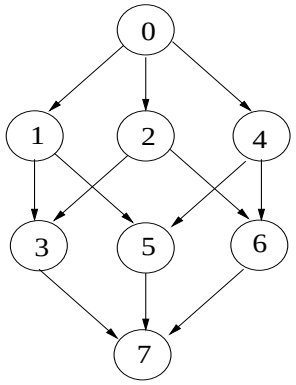

Figura 6: TFF_0 para um sistema com 8 nodos.

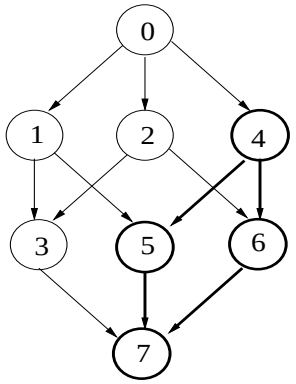

Figura 7: Exemplo de $c_{i,s,p}$: $c_{0,3,4}$ - inclui o nodo 4, o nodo 5, o nodo 6 e o nodo 7.

Para garantir que o nodo i armazene somente a informação mais recente sobre o estado do nodo j adotamos uma estratégia já utilizada em algoritmos de diagnóstico em redes de topologia arbitrária [16, 17]. Esta estratégia consiste na utilização de *timestamps*, ou seja, um mecanismo que permita datar a informação de diagnóstico. Os *timestamps* empregados são implementados como contadores de estados. Um contador deve ser incrementado toda vez que um teste for executado e o testador descobrir que houve uma troca de estado no nodo testado.

Para assegurar que o nodo i mantenha sempre a informação sobre o estado mais recente do nodo j, quando o nodo i testar o nodo p, o nodo i deverá comparar o seu contador com o que receber do nodo p sobre o nodo j. Se o contador do nodo i for igual ou maior que o contador do nodo p, ou seja, estiver recebendo uma informação mais antiga que a sua, o nodo i deverá manter seu contador inalterado. Porém, se o contador do nodo i for menor que o contador que ele receber do nodo p, o nodo i deverá atualizar a informação sobre o estado do nodo j e seu contador.

Na figura 8 o nodo 0 obtém informação sobre o nodo 5 através do nodo 1. Depois, no intervalo de teste em que o nodo 0 testa o nodo 4, como o nodo 5 continua no mesmo estado, então o nodo 0 não precisa atualizar a informação sobre o nodo 5. Já na figura 9 é necessário que o nodo 0 atualize sua informação sobre o nodo 5 através do nodo 4, pois o nodo 4 possui uma informação mais recente sobre o nodo 5.

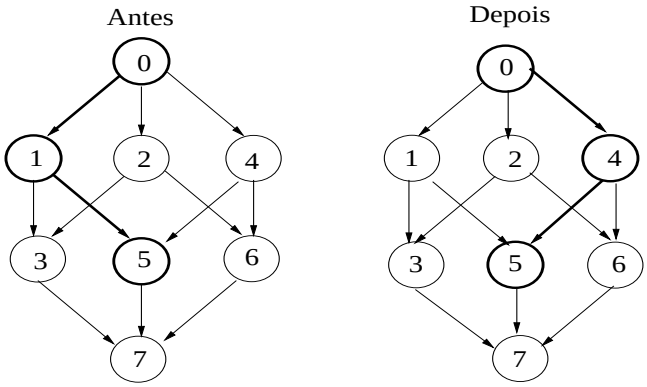

Figura 8: O nodo 0 não necessita atualizar a informação sobre o nodo 5 quando testa o nodo 4.

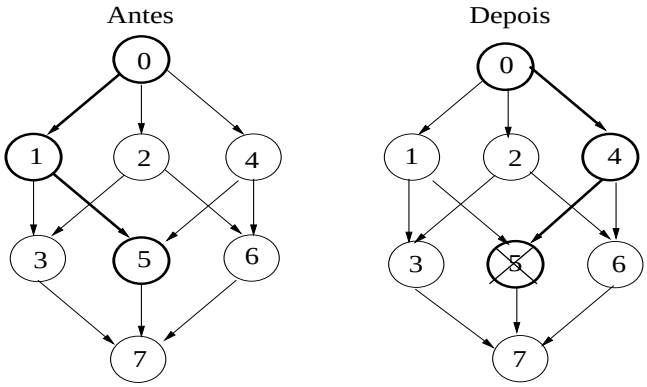

Figura 9: O nodo 0 precisa atualizar a informação sobre o estado do nodo 5 quando testa o nodo 4.

O algoritmo em pseudo-código segue abaixo:

```
ALGORITHM Hi-ADSD with Timestamps
{at node i}

REPEAT FOREVER
   FOR s := 1 TO logN DO
      REPEAT
         next_to_test := next in c(i,s,p);
         test(next_to_test);
         IF (next_to_test IS fault-free)
         THEN
            FOR j := 1 TO N/2 DO
               IF (new timestamp[j] > current timestamp[j])
               THEN update diagnostic information about j;
                    update timestamps;
         IF (next_to_test has changed its state)
         THEN update diagnostic information about next_to_test;
              update timestamp[next_to_test];
      UNTIL (no more_tests are needed in this cluster);
```

4 Latência e Número de Testes Necessários

Seja uma rodada de testes definida como o período de tempo no qual todos os nodos sem-falha do sistema executam testes em pelo menos um cluster. Quando o nodo i testa um cluster, duas situações podem ocorrer: o nodo testado pode estar falho ou sem-falha. Se o nodo i testar um nodo sem-falha, ele obterá informação de diagnóstico sobre o conjunto de nodos cuja distância de diagnóstico começando pelo nodo testado é de até $logN - 1$. Este conjunto tem sempre $N/2$ nodos.

Entretanto, se o nodo testado estiver falho, duas situações diferentes podem ocorrer. Na primeira, se o nodo i estiver testando este cluster pela primeira vez, ele não executará mais testes neste cluster, pois ele tentará obter informação de diagnóstico sobre os nodos deste cluster, através de outros nodos de fora do cluster. A segunda ocorre quando o nodo i já sabe que não conseguirá obter informações de alguns nodos deste cluster por fora do cluster, então ele deverá testar estes nodos sequencialmente até encontrar o primeiro nodo sem-falha ou testar todos estes nodos falhos.

Como ilustrado nas figuras 8, 9 um nodo pode obter informação sobre um outro nodo através de mais de um nodo testado, sendo assim, a latência do algoritmo é reduzida. Entretanto, no pior caso, é mantida a latência dos algoritmos anteriores. Neste caso existe apenas um caminho pelo qual toda a informação de diagnóstico deve trafegar.

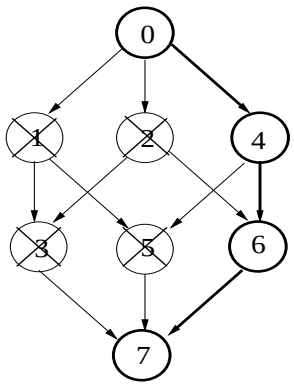

Figura 10: Pior caso da latência

A figura 10 exemplifica o pior caso da latência para um sistema com 8 nodos executando o algoritmo *Hi-ADSD with Timestamps*. Neste pior caso, existe apenas um caminho que liga dois nodos, isto é, existe um único caminho ligando o nodo 0 e o nodo 7.

Como o algoritmo *Hi-ADSD with Timestamps* usa a mesma estratégia de testes do algoritmo *Hi-ADSD with Detours*, eles empregam o mesmo número de testes, que é de no máximo $NlogN$ testes a cada $logN$ rodadas de testes. Este número é menor que o limite máximo do número de testes do algoritmo Hi-ADSD.

O exemplo na figura 11 mostra quando são necessários testes extras. Neste caso, como o nodo 1 e o nodo 2 estão falhos o nodo 0 deve testar o nodo 3; como o nodo 1 e o nodo 4 estão falhos o nodo 0 deve testar o nodo 5; como o nodo 2 e o nodo 4 estão falhos o nodo 0 deve testar o nodo 6; e ainda como todos estes nodos estão falhos o nodo 0 deve testar o nodo 7, pois não existe nenhuma outra opção de caminho para que o nodo 0 obtenha a informação de diagnóstico sobre o nodo 7.

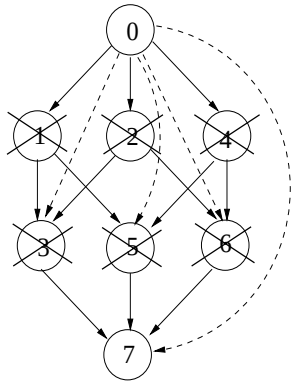

Figura 11: Nodo 0 executa testes extras no nodo 3, no nodo 5, no nodo 6 e no nodo 7.

5 Resultados da Simulação

Nesta seção apresentamos resultados de simulação do algoritmo *Hi-ADSD with Timestamps*. Foram realizados diversos experimentos em redes de tamanhos variados. A linguagem de simulação de eventos discretos SMPL [11] foi usada para obter os resultados que serão apresentados. Os nodos foram modelados como *facilities* do SMPL. Três tipos de eventos foram definidos: teste, falha e reparo.

Resultados de três tipos de experimentos serão apresentados. No primeiro experimento comparamos a latência média do algoritmo *Hi-ADSD with Timestamps* com a latência média dos algoritmos *Hi-ADSD* e *Hi-ADSD with Detours*. Resultados deste experimento mostram que a latência média do algoritmo *Hi-ADSD with Timestamps* é cerca de 70% menor do que a do algoritmo *Hi-ADSD* e do algoritmo *Hi-ADSD with Detours*.

No segundo experimento, é avaliado o número total de testes necessários para que nodos executando *Hi-ADSD with Timestamps* realizem o diagnóstico de um evento. Resultados deste experimento mostram que o número total de testes necessários para um diagnóstico completo do sistema diminui à medida que o número de nodos falhos aumenta. O terceiro experimento ilustra o pior caso da latência, isto é log^2N rodadas de testes, do algoritmo *Hi-ADSD with Timestamps*.

5.1 Comparação da Latência Média dos Algoritmos

Este experimento tem por objetivo medir o tempo médio necessário para que todos os nodos sem-falha de um sistema, executando o algoritmo *Hi-ADSD with Timestamps*, realizem o diagnóstico completo do sistema. Resultados obtidos ilustram a redução da latência deste algoritmo quando comparado com os anteriores.

Primeiramente, em uma rede de 512 nodos executando *Hi-ADSD with Timestamps*, foi realizado um experimento que mostra o número de rodadas de testes necessárias para que um evento seja diagnosticado no sistema. Foram utilizadas duas abordagens para se chegar aos resultados deste experimento: na primeira, os nodos sem-falha sempre testam clusters de mesmo tamanho em uma determinada rodada de testes; por exemplo, se um determinado nodo está testando o cluster com 8 nodos, todos os outros nodos sem-falha deverão estar testando clusters de 8 nodos. Na segunda abordagem, cada nodo pode

# de rodadas	*Abordagem 1*	*Abordagem 2*
0	0	0
1	1	1
2	3	11
3	7	31
4	15	56
5	31	124
6	63	209
7	127	295
8	255	391
9	511	453
10		488
11		504
12		508
13		508
14		509
15		511

Tabela 1: Rodadas necessárias para que os nodos sem-falha, realizem diagnóstico sobre um evento; $N = 512$.

Algoritmo Utilizado	*Latência Média*
Hi-ADSD	40 rodadas
Hi-ADSD with Detours	40 rodadas
Hi-ADSD with Timestamps	12 rodadas

Tabela 2: Comparação da latência média nos 3 algoritmos; $N = 512$

.

executar seus testes em clusters de tamanhos diferentes em uma determinada rodada de testes. Os resultados deste experimento aparecem na tabela 1 e no gráfico na figura 12, que mostram o número total de nodos que conseguiram diagnosticar o evento a cada rodada de testes, até que todos os nodos realizem o diagnóstico deste evento.

Os resultados obtidos com as simulações do algoritmo *Hi-ADSD with Timestamps* ilustram que, na média, ele se mostra cerca de quatro vezes mais rápido que o algoritmo *Hi-ADSD* e que o algoritmo *Hi-ADSD with Detours*, como pode ser constatado pelos resultados ilustrados na tabela 2. Na prática, este é um fator significativo, pois se os intervalos de testes forem, por exemplo, de 30 segundos, o tempo necessário ao diagnóstico de um evento cai de 20 minutos (*Hi-ADSD* e *Hi-ADSD with Detours*) para apenas 6 minutos (*Hi-ADSD with Timestamps*).

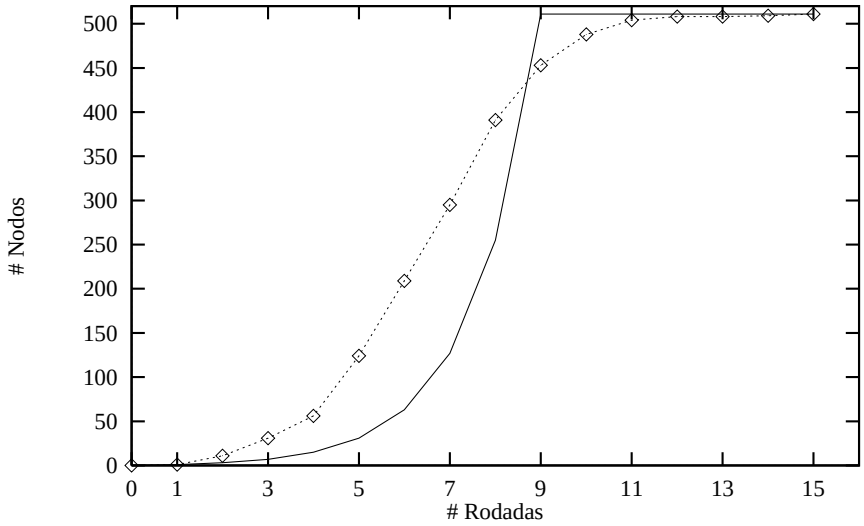

Figura 12: Latência em sistema de 512 nodos, considerando as duas abordagens.
Abordagem1: Linha Contínua;
Abordagem2: Linha Pontilhada.

5.2 Número de Testes do Algoritmo

Este experimento tem por objetivo demonstrar que o número total de testes necessários para o diagnóstico completo do sistema diminui à medida que o número de nodos falhos aumenta. Ele foi realizado considerando uma rede de 64 nodos executando *Hi-ADSD with Timestamps*.

A tabela 3 mostra o número total de testes necessários para que todos os nodos sem-falha completem o diagnóstico de um sistema de 64 nodos. Para obter estes resultados, foi necessário executar a simulação para diferentes números de nodos falhos. Inicialmente, os 64 nodos estão sem-falha; na simulação seguinte, um nodo está falho; depois, dois nodos ficam falhos, e este número é então aumentado gradativamente até o sistema possuir 1 nodo sem-falha e $N-1$ nodos falhos. Os resultados deste experimento mostram que o

Nodos Falhos	*Testes até o diagnóstico completo*
0	384 ($= NlogN$)
1	378
2	374
4	369
6 ($= logN$)	368
12 ($= 2logN$)	352
32 ($= N/2$)	320
63 ($N-1$)	63

Tabela 3: Número de testes necessários para todos os nodos sem-falha completarem o diagnóstico de um evento; $N = 64$.

Rodadas de testes após evento	*Nodos que já diagnosticaram o evento*
0	
6	nodos: 62
12	nodos: 62 e 60
18	nodos: 62, 60 e 56
24	nodos: 62, 60, 56 e 48
30	nodos: 62, 60, 56, 48 e 32
36	nodos: 62, 60, 56, 48, 32 e 0

Tabela 4: A coluna 2 mostra quais nodos diagnosticaram o evento por rodada de testes (coluna 1).

número total de testes executados pelos nodos sem-falha diminui conforme a quantidade de nodos falhos aumenta e confirma que o pior caso do número de testes acontece quando todos os nodos do sistema estão sem-falha.

5.3 Pior caso da Latência no Algoritmo

Este experimento tem por objetivo demonstrar o pior caso da latência no algoritmo *Hi-ADSD with Timestamps*. Ele foi realizado em uma rede de 64 nodos, deixando-se propossitadamente apenas 6 nodos sem-falha no sistema, constituíndo assim, uma configuração que eleva ao máximo a latência do algoritmo.

A tabela 4, mostra o número de rodadas de testes necessárias para que um evento, a falha do nodo 63, seja diagnosticado em um sistema com 64 nodos, em uma configuração que eleva a latência ao máximo. Esta configuração é ilustrada na figura 13.

Este último experimento confirma que, apesar do novo algoritmo apresentar uma latência média menor que a de algoritmos anteriores, o seu pior caso permanece inalterado.

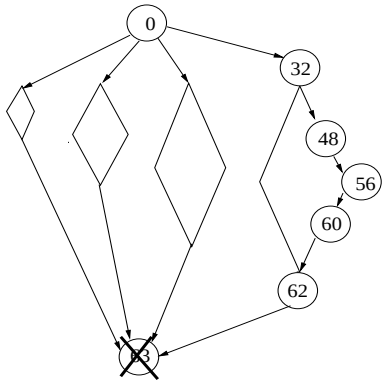

Figura 13: Configuração do sistema que leva ao pior caso da latência.

6 Conclusão

Os algoritmos de diagnóstico a nível de sistema têm sido utilizados para construção de sistemas confiáveis de gerência de falhas de redes de computadores [1, 2]. Estes sistemas são confiáveis por serem tolerantes a falhas, isto é, o processo de diagnóstico continua independente do número de nodos falhos no sistema. Além da confiabilidade, é importante que o algoritmo usado apresente uma boa latência. Neste trabalho, introduzimos um novo algoritmo para diagnóstico de sistemas de grafo completo, *Hi-ADSD with Timestamps*, que tem por objetivo permitir que os nodos sem-falha do sistema completem o diagnóstico em menos tempo que outros algoritmos similares.

Para melhorar a latência de algoritmos anteriores, o novo algoritmo permite que um nodo testador obtenha informações sobre o estado de um outro nodo do sistema a partir de mais de um nodo. O mecanismo que permite esta abordagem é o uso de *timestamps* para datar as informações de diagnóstico no sistema. Quando só existe um caminho entre dois nodos, a latência do algoritmo é a mesma que a de abordagens anteriores. Porém, resultados de simulação comprovam que a latência média do algoritmo é cerca de quatro vezes menor que a latência dos algoritmos anteriores (*Hi-ADSD* e *Hi-ADSD with Detours*), um resultado que pode ser considerado bastante significativo.

O número total de testes executados a cada $logN$ rodadas de testes é de, no máximo, $NlogN$ testes. Ao testar um nodo sem-falha o testador obtém informação de diagnóstico sobre $N/2$ nodos, para um sistema de N nodos.

Trabalhos futuros incluem um estudo do impacto de falhas dinâmicas sobre o algoritmo, isto é, quando os nodos alteram seus estados aleatoriamente. Além disso, mecanismos para permitir que os nodos executem testes de forma síncrona, garantiriam uma latência de $logN$ rodadas de testes, resta saber o custo de tais mecanismos.

Referências

[1] E.P. Duarte Jr., "SNMP-based Fault-Tolerance Network Monitoring," *Proc. XXV Seminário Integrado de Software e Hardware*, pp.38-55, Belo Horizonte, 1998.

[2] E.P. Duarte Jr., "Um algoritmo para Diagnóstico de Redes de Topologia Arbitrária," *Proc. I Workshop de Tolêrancia a Falhas*, pp.50-55, Porto Alegre, 1998.

[3] F. Preparata, G. Metze, and R.T. Chien, "On The Connection Assignment Problem of Diagnosable Systems," *IEEE Transactions on Electronic Computers*, Vol. 16, pp. 848-854, 1968.

[4] S.L. Hakimi, and A.T. Amin, "Characterization of Connection Assignments of Diagnosable Systems," *IEEE Transactions on Computers*, Vol. 23, pp. 86-88, 1974.

[5] 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.

[6] E.P. Duarte Jr., A. Brawerman, and L.C.P. Albini, "System-Level Diagnosis in log^2N Rounds with at most $NlogN$ Tests per $logN$ Rounds," *submetido à IEEE SRDS-18.*

[7] S.L. Hakimi, and K. Nakajima, "On Adaptive System Diagnosis" *IEEE Transactions on Computers*, Vol. 33, pp. 234-240, 1984.

[8] S.H. Hosseini, J.G. Kuhl, and S.M. Reddy, "A Diagnosis Algorithm for Distributed Computing Systems with Failure and Repair," *IEEE Transactions on Computers*, Vol. 33, pp. 223-233, 1984.

[9] R.P. Bianchini, and R. Buskens, "Implementation of On-Line Distributed System-Level Diagnosis Theory," IEEE Transactions on Computers, Vol. 41, pp. 616-626, 1992.

[10] G. Masson, D. Blough, and G. Sullivan, "System Diagnosis," in *Fault-Tolerant Computer System Design,* ed. D.K. Pradhan, Prentice-Hall, 1996.

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

[12] M. Malek, and J. Maeng, "Partitioning of Large Multicomputer Systems for Efficient Fault Diagnosis," *Proc. FTCS-12*, pp. 341-348, 1982.

[13] A. Bagchi, "A Distributed Algorithm for System-Level Diagnosis in Hypercubes," *Proc. 1992 IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pp. 106-113, 1992

[14] M. Barborak, and M. Malek, "Partitioning for Efficient Consensus," *Proc.* 26^{th} *Hawaii International Conference on System Sciences, Vol. II*, pp. 438-446, 1993.

[15] J. Altman, F. Balbach, and A. Hein, "An Approach for Hierarchical System-Level Diagnosis of Massively Parallel Computers Combined with a Simulation-Based Method for Dependability Analysis," *Proc.* 1^{st} *European Dependable Computing Conference*, LNCS 852, pp. 371-385, 1994.

[16] S. Rangarajan, A.T. Dahbura, and E.A. Ziegler, "A Distributed System-Level Diagnosis for Arbitrary Network Topologies," *IEEE Transactions on Computers*, Vol. 44, pp. 312-333, 1995.

[17] E.P. Duarte Jr., F. Mansfield, T Nanya, and S Noguchi, "Non-Broadcast Network Fault-Monitoring Based on System-Level Diagnosis," *Proc. IFIP/IEEE IM'97*, pp. 597-609, 1997.