

Detecção Distribuída de Violação de Integridade em Sistemas com Conteúdo Replicado

Roverli Pereira Ziwich¹, Elias Procópio Duarte Jr.¹, Luiz Carlos Pessoa Albini²

¹Universidade Federal do Paraná, Departamento de Informática
Caixa Postal 19081 - 81531-990, Curitiba, PR Brasil

²Unicenp, Departamento de Informática
R. Prof. Parigot de Souza, 5300, Curitiba, PR Brazil

{roverli,elias}@inf.ufpr.br, albin@unicenp.br

Resumo. Este trabalho apresenta um novo modelo de diagnóstico baseado em comparações e um novo algoritmo, chamado Hi-Dif para a detecção de alterações em sistemas com conteúdo replicado. Um nodo sem-falha executando o algoritmo Hi-Dif, testa outro nodo do sistema para classificar seu estado. O modelo classifica os nodos do sistema em conjuntos de acordo com o resultado dos testes. Um teste é realizado através do envio de uma tarefa para um par de nodos. Após o recebimento das duas saídas da execução destas tarefas, o testador compara estas saídas e, se a comparação indicar igualdade, os nodos são classificados no mesmo conjunto. Por outro lado, se a comparação das saídas indicar diferença, os nodos são classificados em conjuntos distintos, de acordo com o resultado da tarefa. Um dos conjuntos contém os nodos sem-falha do sistema. Uma diferença fundamental do modelo proposto para outros modelos publicados anteriormente é que a comparação por um nodo sem-falha, sobre as saídas produzidas por dois nodos falhos, pode resultar em igualdade. Considerando um sistema com N nodos, prova-se que o algoritmo Hi-Dif possui latência igual a $\log_2 N$ rodadas de testes; que o número máximo de testes requeridos pelo algoritmo é de $O(N^2)$ no pior caso; e, que o algoritmo é $(N-1)$ -diagnosticável. Resultados experimentais obtidos através de simulações e através de implementação do algoritmo aplicado à Web são apresentados.

Abstract. This work presents a new comparison-based diagnosis model and a new algorithm, called Hi-Dif, based on this model. The algorithm is used for checking the integrity of systems with replicated data, for instance, unauthorized Web page modifications. Fault-free nodes running Hi-Dif send a test to two other nodes and the test results are compared. Based on task results, tested nodes are classified in sets. The outputs are then compared; if the comparison produces a match, the two nodes are classified in the same set. On the other hand, if the comparison results in a mismatch, the two nodes are classified in different sets, according to their task results. One of the sets always contains all fault-free nodes. One fundamental difference of the proposed model to previously published models is that the new model allows the task outputs of two faulty nodes to be equal to each other. Considering a system of N nodes, it is proved that the algorithm has latency equal to $\log_2 N$ testing rounds; that the maximum number of tests required is $O(N^2)$ in the worst case; and, that the algorithm is $(N-1)$ -diagnosable. Experimental results obtained by simulation and by the implementation of the algorithm applied to the Web are presented.

1. Introdução

Atualmente já passa de 600 milhões o número estimado de pessoas que utilizam a Internet. O bom funcionamento da rede é cada vez mais importante para indivíduos e organizações. Por outro lado, ataques e ações de vandalismo como, por exemplo, modificações não autorizadas de conteúdo na Web têm se tornado cada vez mais comuns [1].

Segundo [2], um serviço especializado em registrar invasões de sites, em 2001 o número de sites invadidos e vandalizados virtualmente, ou seja, com conteúdo modificado, em todo o mundo foi de 22,4 mil. Já em 2003, estatísticas mostravam que o número de sites invadidos já chegava a mais de 137 mil [1]. Com isso, cresce a preocupação com a monitoração de sistemas visando a detecção de violações de integridade. Neste trabalho, integridade é definida como a garantia de que os dados sempre são alterados de forma autorizada [3].

O objetivo de um sistema de monitoração é descobrir quais são as unidades falhas de um sistema. É essencial que a monitoração seja tolerante a falhas, ou seja, capaz de funcionar corretamente mesmo na presença de unidades falhas. Este trabalho trata da monitoração, utilizando diagnóstico distribuído [4] baseado em comparações [5], para sistemas com conteúdo replicado em uma rede como, por exemplo, a Internet. Uma das aplicações práticas deste trabalho é o diagnóstico de vandalismo em servidores Web com dados replicados [6].

Existem muitas ferramentas e trabalhos publicados para detecção de alterações, como a detecção de modificações ou atualizações em uma página Web que é frequentemente acessada [7, 8, 9, 10, 11]. Estas ferramentas utilizam comparações para poder determinar quando houve alguma alteração. Muitas destas ferramentas armazenam uma cópia local dos arquivos [7] ou um *checksum* dos arquivos [8] que estão sendo monitorados para se poder comparar com o arquivo ou com a parte de interesse do arquivo que está na Web. Um exemplo que funciona como este modelo é a ferramenta WebMon [7]. Outras ferramentas ainda executam como um serviço na própria Web, como, por exemplo, a ferramenta URL-minder [8].

O novo modelo de diagnóstico proposto neste trabalho é distribuído, ou seja, não há um elemento centralizado responsável pela monitoração. Ao contrário, as unidades – que possuem os dados replicados – se monitoram de acordo com um modelo de diagnóstico distribuído baseado em comparações. Cada nodo sem-falha executa o algoritmo *Hi-Dif*, introduzido neste trabalho, e realiza testes de forma distribuída e hierárquica em outros nodos do sistema. Um teste é realizado através do envio de uma tarefa para um par de nodos do sistema. Após o recebimento das duas saídas da execução destas tarefas, o testador compara estas saídas e classifica os nodos em conjuntos de acordo com o resultado da tarefa. Um dos conjuntos contém os nodos sem-falha do sistema.

Como o sistema é assíncrono, falsas suspeitas de falhas podem acontecer. Estas suspeitas são tratadas pelo sistema como se fossem falhas reais, e considera-se que se foram identificadas pela execução do procedimento de testes é porque refletem queda de desempenho que pode ser interpretada como uma falha real. As falsas suspeitas são disseminadas normalmente, como qualquer falha real. Quando a comunicação é retomada, o nodo é considerado recuperado.

As provas para a latência, número máximo de testes e a diagnosticabilidade do algoritmo são apresentadas. Também são apresentados resultados experimentais obtidos através de simulações e através de implementação do algoritmo aplicado à Web.

O restante deste trabalho está organizado da seguinte maneira. A seção 2 apresenta o diagnóstico baseado em comparações. Em seguida, a seção 3 descreve o novo modelo proposto. A seção 4 apresenta o algoritmo *Hi-Dif*. Em seguida, na seção 5, são descritos os resultados experimentais através de simulação e implementação do algoritmo utilizado para o diagnóstico de nodos com conteúdo replicado na Web. E, na seção 6, seguem as conclusões e trabalhos futuros.

2. Diagnóstico Baseado em Comparações

O objetivo do diagnóstico em nível de sistema é identificar quais unidades do sistema estão falhas e quais estão sem-falhas [4]. As unidades do sistema realizam testes sobre as outras unidades e comunicam resultados de testes visando obter o diagnóstico completo do sistema.

Diferentes modelos de diagnóstico em nível de sistema [4] têm sido propostos. Os modelos de diagnóstico baseados em comparações foram propostos inicialmente por Malek [12] e em seguida por Chwa e Hakimi [13]. Nestes modelos, assume-se que em um sistema de N unidades, a comparação dos resultados produzidos na saída da execução de tarefas de qualquer par de unidades é possível. Qualquer diferença na comparação indica que uma ou ambas as unidades estão falhas. Estes modelos também assumem que existe um observador central que armazena as saídas de tarefas e que através das comparações das saídas chega-se ao diagnóstico

completo do sistema, determinando as unidades falhas. Este observador central é uma unidade confiável que não sofre nenhuma falha. A diferença deste modelo proposto em [13] para o modelo proposto por Malek [12], é que duas unidades falhas, quando submetidas à mesma tarefa, podem produzir as mesmas saídas, podendo assim, causar uma igualdade na comparação destas saídas.

O modelo MM (Maeng e Malek) [14], é uma extensão do modelo de diagnóstico baseado em comparação proposto inicialmente por Malek [12]. Este modelo assume que os resultados das comparações das saídas são enviados para um observador central que realiza o diagnóstico completo do sistema, mas permite que as comparações das saídas das tarefas sejam feitas pelas próprias unidades. A única restrição é que a unidade que realiza a comparação deve ser diferente das duas unidades que produzem as saídas da tarefa executada. Sengupta e Dahbura em [15] propuseram uma generalização do modelo MM chamada de modelo MM*. Este modelo permite que a própria unidade testadora seja uma das unidades que produzem as saídas.

Blough e Brown em [16] apresentaram um modelo de diagnóstico baseado em comparações com broadcast confiável. Neste modelo, uma tarefa é atribuída como entrada para um par de nodos diferentes. Estes dois nodos executam a tarefa e as saídas produzidas são comparadas para detectar uma possível falha em um dos dois nodos. Estes dois nodos que estão sendo comparados fazem broadcast das suas saídas para todos os nodos do sistema, inclusive eles próprios. As comparações das saídas são realizadas em todo nodo sem-falha do sistema.

Em [17] Wang propõe um modelo de diagnóstico baseado em comparações utilizando hipercubos e também nos chamados hipercubos melhorados (*enhanced hypercubes*). Araki e Shibata em [18] estudam a diagnosticabilidade de redes borboletas através do diagnóstico baseado em comparações; e, Fan em [19] avalia a diagnosticabilidade de cubos cruzados – uma variante dos hipercubos, mas com diâmetro menor.

O modelo genérico de diagnóstico distribuído e adaptativo em nível de sistema baseado em comparações é apresentado por Albini e Duarte em [5]. Este modelo usa a mesma estratégia hierárquica de testes similar à implementada no algoritmo *Hi-ADSD with Timestamps* [20], mas não é limitado a falhas do tipo crash. Um nodo sem-falha testa outro nodo do sistema para identificar seu estado. Este modelo possui a seguinte asserção: quando um nodo sem-falha compara as saídas produzidas por dois nodos falhos, o resultado sempre indica diferença. O algoritmo *Hi-Comp (Hierarchical Comparison-Based Adaptive Distributed System-Level Diagnosis Algorithm)* é apresentado e proposto para o modelo genérico de diagnóstico baseado em comparações [5].

3. Um Modelo Genérico de Diagnóstico Baseado em Comparações

Neste trabalho é proposto um novo modelo genérico de diagnóstico distribuído e adaptativo em nível de sistema baseado em comparações. Considera-se que um sistema S com N nodos é completamente conectado, e é também representado por um grafo $G=(V, E)$, onde V é um conjunto de vértices e E é um conjunto de arestas. Cada vértice do grafo corresponde a um nodo do sistema e as arestas correspondem aos enlaces de comunicação. Neste modelo os enlaces de comunicação não falham. Os nodos do sistema podem estar falhos ou sem-falhas. Um nodo falho pode tanto estar com falha do tipo crash, como apenas estar respondendo de maneira incorreta à tarefa enviada como teste. Um *evento* é definido como a mudança de estado de um nodo detectada pelo algoritmo de diagnóstico.

Um nodo sem-falha testa outro nodo do sistema para *classificar* seu estado. Os nodos são classificados em conjuntos de acordo com o resultado dos testes. Um teste é realizado através do envio de uma tarefa para um par de nodos do sistema. Após o recebimento das duas saídas da execução destas tarefas, o testador compara estas saídas e, se a comparação indicar igualdade, os nodos são classificados no mesmo conjunto. Por outro lado, se a comparação das saídas indicar diferença, os nodos são classificados em conjuntos distintos, de acordo com o resultado da tarefa. Um dos conjuntos contém os nodos sem-falha do sistema. O simples fato dos nodos serem classificados em mais de um conjunto indica que há nodos falhos no sistema.

Como este novo modelo faz uma classificação dos nodos em conjuntos a partir das saídas das tarefas recebidas e considerando a aplicação para a detecção de violação de integridade de

sistemas com conteúdo replicado, este modelo permite tanto a identificação dos nodos que estão com falha do tipo crash, como a identificação dos nodos que possuem o mesmo conteúdo.

As seguintes asserções são necessárias neste modelo:

- 1) a comparação realizada por um nodo sem-falha sobre as saídas produzidas por dois nodos sem-falha sempre resulta em igualdade;
- 2) a comparação realizada por um nodo sem-falha sobre as saídas produzidas por um nodo falho e outro sem-falha sempre resulta em diferença;
- 3) o tempo para um nodo sem-falha produzir uma saída para uma tarefa é limitado.

Uma diferença do modelo proposto para outros modelos distribuídos propostos anteriormente, é que a comparação realizada por um nodo sem-falha sobre as saídas produzidas por dois nodos falhos pode resultar em igualdade.

Um multigrafo $M(S)$, representa os testes executados no sistema. $M(S)$ é um multigrafo direcionado definido sobre o grafo G , onde todos os nodos do sistema são sem-falha. Os vértices de $M(S)$ correspondem aos nodos do sistema S . Cada aresta de $M(S)$ representa que um nodo está enviando uma tarefa para ser executada por outro nodo, isto é, existe uma aresta do nodo i para o nodo j se o nodo i envia uma tarefa para ser executada pelo nodo j .

Considere como exemplo o multigrafo $M(S)$ na figura 3.1, onde o nodo 0 envia um par de tarefas para si próprio e para o nodo 1, e também outro par de tarefas para si próprio e para o nodo 2. Neste exemplo as arestas do grafo são $(0, 0)_1$, $(0, 1)_0$, $(0, 0)_2$ e $(0, 2)_0$, todas direcionadas do nodo 0 para outros nodos ou para si próprio. A aresta $(0, 1)_0$ indica que o nodo 0 está enviando uma tarefa para o nodo 1 e a saída desta tarefa será comparada com a saída da tarefa enviada do nodo 0 para o próprio nodo 0, e obrigatoriamente deve existir a aresta $(0, 0)_1$.

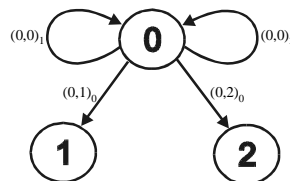


Figura 3.1: Exemplo de multigrafo $M(S)$.

4. O Algoritmo *Hi-Dif*

Nesta seção é apresentado um algoritmo para o diagnóstico adaptativo distribuído e hierárquico baseado em comparações, chamado *Hi-Dif*, baseado no modelo de diagnóstico apresentado na seção anterior. Este novo algoritmo permite a detecção distribuída de violação de integridade em sistemas com conteúdo replicado. Este novo algoritmo ainda permite a identificação dos nodos que possuem uma mesma modificação de conteúdo e sua diferenciação de outros nodos que possuem outras modificações.

4.1 Descrição do Algoritmo *Hi-Dif*

O algoritmo utiliza uma estratégia de testes representada por um grafo de testes de nodos sem-falha, $T(S)$, que é um grafo direcionado do sistema definido sobre o multigrafo $M(S)$. $T(S)$ é um hipercubo quando todos os nodos do sistema estão sem-falhas e quando a quantidade de nodos do sistema é uma potência de 2. A figura 4.1(a) mostra um grafo $T(S)$ para um sistema de 8 nodos.

A *distância de diagnóstico* entre o nodo i e o nodo j é definida como a menor distância entre o nodo i e o nodo j em $T(S)$, isto é, o caminho com o menor número de arestas entre o nodo i e o nodo j no grafo $T(S)$. Apesar de ser o menor caminho, pode existir mais de um caminho entre estes dois nodos com a mesma distância de diagnóstico. Por exemplo, na figura 4.1(a), a distância de diagnóstico entre o nodo 0 e o nodo 3 é 2.

O grafo de testes de nodos sem-falha do nodo i , $T_i(S)$, é um grafo direcionado sobre $T(S)$ e mostra como a informação de diagnóstico é transmitida no sistema. Existe uma aresta em $T_i(S)$ do nodo a para o nodo b se existe uma aresta em $T(S)$ do nodo a para o nodo b e a distância de diagnóstico do nodo i para o nodo a for menor que a distância de diagnóstico do nodo i para o

nodo b . A figura 4.1(b) mostra a $T_0(S)$ para um sistema de 8 nodos. Chamamos de *filhos* do nodo i na $T_i(S)$, os nodos que possuem ligação direta com o nodo i . No exemplo da figura 4.1(b) os filhos do nodo 0 são os nodos 1, 2 e 4.

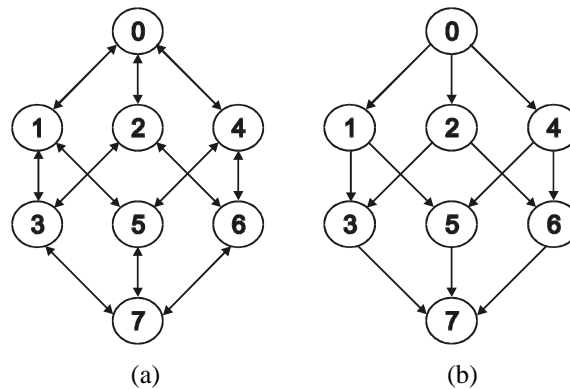


Figura 4.1: (a) $T(S)$ para um sistema com 8 nodos. (b) $T_0(S)$ para um sistema com 8 nodos.

Uma *rodada de testes* é um intervalo de tempo no qual todos os nodos sem-falha do sistema obtêm informação de diagnóstico sobre todos os nodos do sistema. Uma asserção é feita, na qual após o nodo i já ter testado o nodo j em uma certa rodada de teste, o nodo j não pode sofrer mais nenhum evento nesta rodada de testes. Esta asserção é necessária para garantir a propagação das informações de diagnóstico pelo sistema.

A estratégia de testes agrupa os nodos em clusters como o algoritmo *Hi-ADSD with Timestamps* [20]. Uma função $c_{i,s,p}$ baseada na distância de diagnóstico define a lista de nodos sobre os quais o nodo i pode obter informação de diagnóstico através e a partir de um nodo p , com distância de diagnóstico menor ou igual a $s-1$ na $T_i(S)$. Neste algoritmo, s é sempre igual a $\log_2 N$, o que implica que cada cluster tem sempre $N/2$ nodos. A figura 4.2 mostra esta divisão de clusters para um sistema de 8 nodos para $T_0(S)$. Os clusters são: (a) $c_{0,3,1}$: nodos $\{1, 3, 5, 7\}$, (b) $c_{0,3,2}$: nodos $\{2, 3, 6, 7\}$ e (c) $c_{0,3,4}$: nodos $\{4, 5, 6, 7\}$.

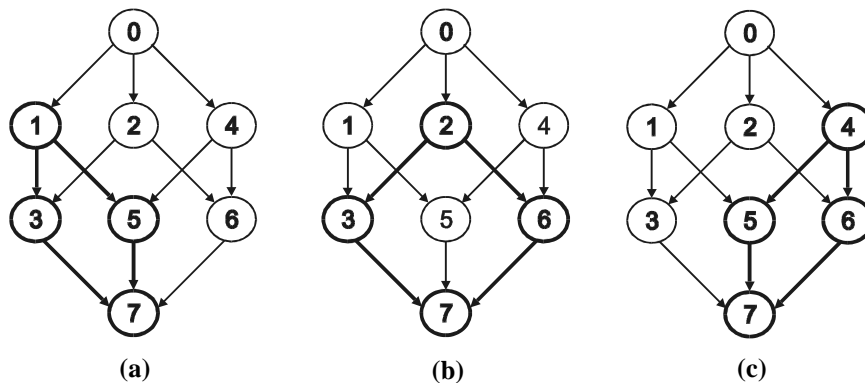


Figura 4.2: Divisão de clusters para um sistema de 8 nodos para $T_0(S)$.

Em comparação com outros modelos de diagnóstico, o modelo do algoritmo *Hi-Dif* tem uma diferença importante: a asserção segundo a qual a comparação realizada por um nodo sem-falha sobre as saídas produzidas por dois nodos falhos sempre resulta em diferença foi eliminada. Para eliminar esta asserção, neste algoritmo, um nodo i sempre se considera como sem-falha. Além disso, a estratégia de testes dos nodos será a de enviar tarefas em pares, sempre uma para si próprio e outra para um nodo a ser testado. Ou seja, um nodo i testador, sempre envia uma tarefa para si próprio e para outro nodo j e compara o resultado destas tarefas.

Se a comparação do resultado destas tarefas do próprio nodo i e do nodo j indicar igualdade, o nodo i irá obter informação de diagnóstico deste nodo j . Desta forma o nodo i sempre considera como sem-falha somente os nodos que gerarem saída para a tarefa enviada, igual à saída gerada pela tarefa enviada ao próprio nodo testador.

Por este motivo, este algoritmo possui as seguintes asserções sobre o observador externo que é o “usuário” do algoritmo: este observador externo é quem realiza o diagnóstico do sistema, determinando corretamente se um nodo está sem-falha a fim de obter os dados de diagnóstico do sistema a partir daquele nodo. Desta forma, este observador nunca irá escolher um nodo que esteja falho – ou no caso específico a que se aplica o algoritmo, um nodo invadido ou com conteúdo incorreto – para obter as informações de diagnóstico do sistema.

No algoritmo *Hi-Dif* os testes são realizados através do envio da tarefa para 2 nodos do sistema que a executam e devolvem os resultados para o nodo testador. Além disso, no algoritmo um destes nodos sempre será o próprio nodo testador. Assim, quando um nodo testador i envia uma tarefa para o par de nodos i e j , o nodo i considera o nodo j como sem-falha quando o resultado das tarefas enviadas indicar igualdade. Caso contrário, o nodo j será considerado falho.

4.2 Classificação de Nodos com Mesmo Resultado

Além de apresentar o diagnóstico do sistema, o algoritmo também executa uma classificação dos nodos do sistema em conjuntos. Estes conjuntos são construídos de forma a permitir a fácil identificação de quais nodos do sistema estão respondendo com o mesmo resultado para as tarefas enviadas como teste. Um dos objetivos desta classificação é, por exemplo, permitir a fácil identificação de quais nodos estão com falha do tipo crash, ou ainda permitir identificar quais conjuntos de nodos estão invadidos e vandalizados com o mesmo tipo de alteração sobre o conteúdo que deveria estar replicado.

Para isso, o algoritmo classifica os nodos em conjuntos com identificador numérico e sequencial e estes conjuntos fazem a seguinte classificação: o conjunto 0 para nodos com falhas do tipo crash; o conjunto 1 para nodos que possuem o conteúdo correto, ou seja, que estão respondendo corretamente às tarefas; e, os conjuntos com identificador maior que 1, para os nodos com falhas por estarem respondendo às tarefas enviadas de forma diferente da considerada correta.

Para isso, sempre que um nodo testador envia uma tarefa para um par de nodos – para si próprio e para um nodo y – é realizada a comparação do resultado das tarefas e o testador classifica o nodo y como falho ou sem-falha, de acordo com o resultado da comparação. Além disso, o testador coloca o nodo y no conjunto 0 se este nodo não responder ao teste, coloca o nodo testado no conjunto 1 se o resultado da tarefa enviada ao nodo for igual ao resultado da tarefa enviada ao próprio nodo testador. Mas se o resultado da tarefa do nodo y for diferente do resultado da tarefa do próprio nodo i , o testador irá colocar este nodo em um dos conjuntos com identificador maior que 1 de acordo com o resultado da tarefa. Caso o resultado da tarefa não se enquadre em nenhum dos conjuntos existentes, será criado um novo conjunto para este nodo y .

4.3 Especificação do Algoritmo

O algoritmo *Hi-Dif* agrupa os nodos em dois conjuntos de estados: o conjunto dos nodos falhos: F , e o conjunto dos nodos sem-falha: FF . Estes conjuntos são sempre disjuntos e a união deles sempre resulta em V , isto é $F \cap FF = \emptyset$ e $F \cup FF = V$. Além disso, o algoritmo utiliza também uma lista de conjuntos, *result-set-list*, para classificar o nodo de acordo com o resultado da tarefa enviada. Cada nodo do sistema mantém estes 2 conjuntos F e FF e mantém também a lista de conjuntos *result-set-list*. Ao final de cada rodada de testes cada nodo deve estar em um dos dois conjuntos F ou FF , e deve estar em exatamente um dos conjuntos da lista *result-set-list*.

Quando o nodo i compara as saídas de uma tarefa realizada pelo próprio nodo i e por outro nodo p , e essa comparação indicar igualdade, o nodo i identifica o nodo p como sem-falha. O nodo i coloca o nodo p no conjunto FF retirando-o do conjunto F caso estivesse neste conjunto. Além disso, o nodo p é colocado no conjunto 1 da lista *result-set-list*, ou seja, é colocado no conjunto dos nodos com conteúdo correto. Quando o nodo i identifica um nodo sem-falha, ele obtém desse nodo informações de diagnóstico sobre todo o cluster desse nodo. Além disso, como são utilizados *timestamps* [20] para datar as informações, o nodo i deve testar se as informações que está recebendo são mais novas que as que ele já possui. Os *timestamps* são implementados através de contadores de troca de estados, que cada nodo possui para todos os nodos do sistema.

Essa estratégia permite determinar a ordem em que os eventos foram detectados e garantir que o nodo i receba sempre a informação de diagnóstico mais recente sobre o estado de todos os nodos do sistema. Caso as informações que está recebendo não sejam mais recentes, o nodo i deve simplesmente descartar estas informações.

Quando o nodo i envia uma tarefa para ser realizada pelo próprio nodo i e por outro nodo p , mas o nodo p não responde, o nodo i identifica o nodo p como falho. O nodo i coloca o nodo p no conjunto F retirando-o do conjunto FF caso estivesse neste conjunto. Além disso, o nodo p é colocado no conjunto 0, ou seja, é colocado no conjunto dos nodos com falha do tipo crash da lista *result-set-list*.

Quando o nodo i compara as saídas de tarefas realizadas pelo próprio nodo i e por outro nodo p , e essa comparação indicar diferença, o nodo i identifica o nodo p como falho. O nodo i coloca o nodo p no conjunto F retirando-o do conjunto FF caso estivesse neste conjunto. Quando o nodo i identifica um nodo p como falho porque o resultado da tarefa do nodo p foi diferente do resultado da tarefa enviada ao nodo i , é procurado um conjunto na lista *result-set-list* para se colocar o nodo p . Caso nenhum dos conjuntos já existentes contenham nodos com o mesmo tipo de alteração de conteúdo, será criado um novo conjunto na lista *result-set-list* e colocado o nodo p neste conjunto.

Desta forma, o nodo i testa primeiramente seus filhos na $T_i(S)$ – definida na seção 4.1 – enviando uma tarefa sempre para o próprio nodo i e para o seu próximo filho a testar. Assim, através da comparação do resultado das tarefas enviadas, o nodo i determina o estado de todos os seus nodos filhos. Quando o nodo i acaba de testar seus filhos, e ainda restam nodos não testados em que não se tem informação sobre seu estado, o nodo i testa diretamente estes nodos, um por um. O nodo i então testa primeiramente os nodos com menor identificador e com distância de diagnóstico igual a 2, ou seja, os nodos filhos dos seus filhos. Como exemplo, a figura 4.3(a) mostra um sistema de 8 nodos onde os nodos 2 e 4 são falhos. Neste exemplo, o nodo 0 após testar todos os seus filhos, recebe informações de diagnóstico sobre os nodos 3, 5 e 7, mas ainda não tem nenhuma informação sobre o estado do nodo 6. Neste momento o nodo 0 realiza mais um teste enviando um par de tarefas para o nodo 3 e para o nodo 6.

Se depois de testados estes nodos com distância de diagnóstico igual a 2, ainda existirem nodos que não foram testados e que não se obteve informação sobre seu estado na rodada de testes corrente, o nodo i então testa os nodos, por ordem de identificador, com distância de diagnóstico igual a 3, e assim por diante até que se conheça o estado de todos os nodos do sistema. Como exemplo, a figura 4.3(b) mostra um sistema de 8 nodos no qual os nodos 1, 2 e 4 estão falhos. Neste exemplo, o nodo 0 testa seus filhos, isto é, testa os nodos 1, 2 e 4 e descobre que estão falhos e não recebe nenhuma informação de diagnóstico sobre os demais nodos do sistema. Neste momento o nodo 0 começa a realizar testes adicionais até obter informação sobre todos os nodos do sistema. Primeiramente o nodo 0 testa o primeiro nodo com distância de diagnóstico igual a 2 e com menor identificador, ou seja, o nodo 3. Em seguida testa o nodo 5 e por fim testa o nodo 6.

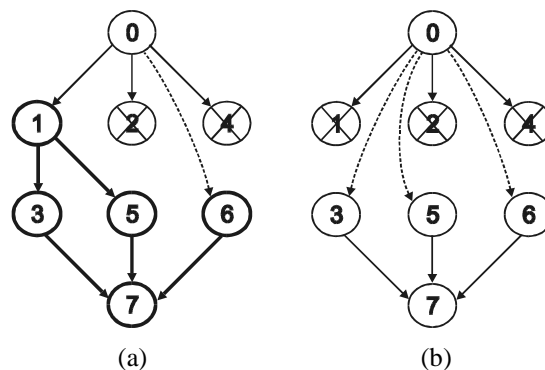


Figura 4.3: (a) Sistema de 8 nodos onde os nodos 2 e 4 são falhos.
(b): Sistema de 8 nodos com os nodos 1, 2 e 4 falhos.

Assim, ao final de cada rodada de testes, todo nodo i sem-falha terá todos os nodos do sistema em um dos dois conjuntos F ou FF , isto é $F \cup FF = V$. Além disso todos os nodos

também estarão em um dos conjuntos da lista *result-set-list*. Um nodo que estiver no conjunto 0 da lista *result-set-list*, necessariamente também estará no conjunto *F*. Um nodo que estiver no conjunto 1 da lista *result-set-list*, necessariamente estará no conjunto *FF*. E um nodo que estiver em qualquer outro conjunto da lista *result-set-list*, necessariamente também estará no conjunto *F*. Abaixo é apresentado o algoritmo em pseudo-código.

```

Algoritmo rodando no nodo i:
F = EMPTY; FF = EMPTY;
REPETIR PARA SEMPRE
  Inicializar( result-set-list );
  TO_TEST = {ALL NODES};
  REPETIR
    p = próximo_nodo para testar em TO_TEST;
    envia_tarefa( i, p );
    TO_TEST = TO_TEST - {p}
    SE ( resultado(p) == Ø )
      ENTÃO
        FF = FF - {p};
        F = F + {p};
        result-set-list[conjunto 0] =
          result-set-list[conjunto 0] + {p};
    SENÃO SE ( resultado(i) == resultado(p) )
      ENTÃO
        F = F - {p};
        FF = FF + {p};
        result-set-list[conjunto 1] =
          result-set-list[conjunto 1] + {p};
        OBTER informações de diagnóstico do cluster de p;
        COMPARAR timestamps dos nodos do cluster;
        ATUALIZAR informações locais se necessário;
        TO_TEST = TO_TEST - {nodos com info atualizadas};
    SENÃO SE ( resultado(i) != resultado(p) )
      ENTÃO
        FF = FF - {p};
        F = F + {p};
        id_conj = NULO;
        PARA (x começando em 2;
              x <= result-set-list.id_maior_conjunto E id_conj == NULO ) FAÇA
          SE ( resultado(um nodo result-set-list[conjunto x])
              == resultado(p) )
            ENTÃO
              id_conj = x;
          FIM_SE
        FIM_PARA
        SE ( id_conj != NULO )
          ENTÃO
            result-set-list[conjunto id_conj] =
              result-set-list[conjunto id_conj] + {p};
          SENÃO
            id_novo_conj = result-set-list.criar_novo_conjunto;
            result-set-list[conjunto id_novo_conj] = {p};
          FIM_SE
        FIM_SE
    ATÉ ( foi encontrado o estado de todos os nodos )
  FIM_REPETIR

```

4.4 Provas

Nesta seção são apresentadas as provas da latência, do número de testes e da diagnosticabilidade (*diagnosability*) do algoritmo *Hi-Dif*.

Teorema 1: *Todos os nodos sem-falha executando o algoritmo Hi-Dif necessitam de, no máximo, $\log_2 N$ rodadas de testes para atingir o diagnóstico completo do sistema.*

Prova:

Considere um novo evento que ocorre no nodo a. Pela definição de rodada de testes, todos os filhos do nodo a, realizam o diagnóstico desse evento na primeira rodada de testes posterior ao evento. Considerando o grafo $T_a(S)$, ilustrado na figura 4.4, na primeira rodada de testes posterior ao evento, os filhos do nodo a realizam o diagnóstico do evento que ocorre em a.

Já na segunda rodada de testes, os nodos que possuem distância de diagnóstico igual a 2 em relação ao nodo a realizam o diagnóstico, recebendo informações dos nodos com distância igual a 1, ou testando o nodo a diretamente, caso os nodos com distância 1 estejam falhos. Na

$T_a(S)$, ilustrada na figura 4.4, os nodos que são filhos de a realizam o diagnóstico do evento em questão, recebendo informações dos filhos de a ou testando o próprio nodo a .

Assuma que o nodo i sem-falha com distância de diagnóstico igual a d com relação ao nodo a realiza o diagnóstico de um evento ocorrido no nodo a em d rodadas de testes. Considere agora um nodo j com distância de diagnóstico igual a $d+1$ até o nodo a . Pela definição de distância de diagnóstico, todo nodo com distância de diagnóstico igual a $d+1$ com relação ao nodo a é filho de algum nodo com distância de diagnóstico igual a d em relação ao nodo a . Então o nodo j é filho de algum nodo i . Pela definição de rodada de testes, um nodo obrigatoriamente testa todos os seus filhos em cada rodada de testes, então o j testa o nodo i em todas as rodadas de testes.

Como o nodo j testa o nodo i em todas as rodadas de testes, o nodo j pode demorar uma rodada de testes para receber novas informações através do nodo i . Então, se o nodo i demora d rodadas de testes para realizar o diagnóstico do evento ocorrido em a e o nodo j demora mais uma rodada de testes para realizar o diagnóstico através do nodo i , o nodo j demora $d+1$ rodadas de testes para realizar o diagnóstico do evento ocorrido em a . Portanto, para um nodo j realizar o diagnóstico de um evento ocorrido em um nodo a , com distância de diagnóstico entre eles de $d+1$, o nodo j demora $d+1$ rodadas de testes.

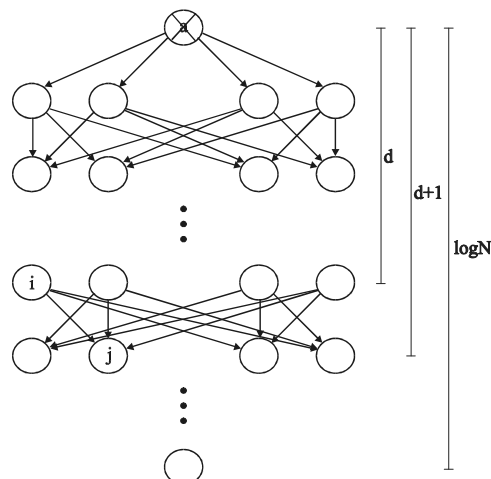


Figura 4.4: Ilustração de um grafo $T_a(S)$.

Concluindo, se a distância de diagnóstico entre dois nodos for x então um desses nodos demora até x rodadas de testes para realizar o diagnóstico de um evento ocorrido no outro nodo. Ou seja, um nodo pode demorar x rodadas de testes para realizar o diagnóstico de um evento em um nodo com distância de diagnóstico x até ele.

A latência máxima do algoritmo ocorre entre os nodos com maior distância de diagnóstico do sistema. Pela definição de um hipercubo e de distância de diagnóstico já apresentada, a maior distância de diagnóstico entre dois nodos no sistema é de $\log_2 N$. Portanto a latência máxima do algoritmo é de $\log_2 N$ rodadas de testes. \square

Teorema 2: O número máximo de testes realizados pelos nodos sem-falha em uma rodada de testes é $O(N^2)$.

Prova:

Considerando um sistema com N nodos, o pior caso para a quantidade de testes em uma rodada de testes é o pior caso de testes para cada um dos N nodos. No pior caso, a quantidade de testes necessárias para 1 nodo ocorre quando este precisa testar todos os outros $N-1$ nodos do sistema; um exemplo ocorre na situação onde estes $N-1$ nodos estão falhos. Neste caso o nodo sem-falha executa $N-1$ testes.

Considere este caso, em que há somente um nodo i sem-falha no sistema e $N-1$ nodos falhos, mas não com falha do tipo crash. Também considere que cada um dos $N-1$ nodos está com o conteúdo replicado alterado de forma que todos os nodos estão com conteúdos diferentes entre si, ou seja, considerando os N nodos, tem-se N conteúdos diferentes e

somente 1 correto.

Para realizar o diagnóstico, o nodo sem-falha envia testes para si próprio e para cada um dos nodos falhos, então o número de testes realizados é $N-1$, pois existem $N-1$ nodos falhos no sistema. O mesmo acontece com cada um dos outros $N-1$ nodos falhos. Como estão com falha que não é do tipo crash, cada nodo considera-se sem-falha e realiza também outros $N-1$ testes, pois para este nodo existem $N-1$ nodos falhos (nodos que estão com conteúdo diferente deste nodo testador). Ao considerar esta configuração do sistema, temos o pior caso do número de testes para cada nodo, ou seja, temos a seguinte quantidade de testes: $N * (N-1) = N^2 - N$, que é $O(N^2)$. □

Teorema 3: Um sistema executando o algoritmo *Hi-Dif*, é $(N-1)$ -diagnosticável.

Prova:

Inicialmente considere um sistema com somente um nodo sem-falha e $N-1$ nodos falhos. O nodo sem-falha vai testar todos os nodos do sistema, enviando testes para si próprio e para cada um dos nodos falhos, identificando o estado de todos os nodos.

Agora considere um sistema com mais de um nodo sem-falha. Um desses nodos sem-falha realiza testes até encontrar outro nodo sem-falha. Quando o testador encontra um nodo sem-falha, o testador obtém informações de diagnóstico do nodo testado. Juntando as informações recebidas do nodo sem-falha testado com as informações coletadas pelos seus próprios testes, o nodo sem-falha realiza um diagnóstico completo do sistema. Entretanto, se ocorrer a situação ilustrada pela figura 4.5, na qual o nodo a obtém informações de diagnóstico sobre o nodo c através do nodo b e o nodo b obtém informações de diagnóstico sobre o nodo c através do nodo a, os nodos a e b não completariam o diagnóstico do sistema.

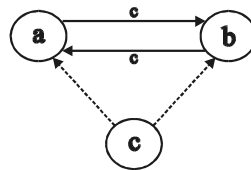


Figura 4.5: Caso impossível em um sistema executando o algoritmo *Hi-Dif*.

Porém essa situação jamais ocorre, pois para o nodo a receber informações de diagnóstico do nodo c, através do nodo b, a distância de diagnóstico do nodo a para o nodo c deve ser maior que a distância de diagnóstico do nodo b para o nodo c. Já para o nodo b receber informações sobre o nodo c através do nodo a, a distância do nodo b para o nodo c deve ser maior que a distância do nodo a para o nodo c. O que resulta em uma contradição, e nunca ocorre no sistema.

Desta forma, mesmo se houver apenas um único nodo sem-falha, esse nodo será capaz de completar corretamente o diagnóstico do sistema. Portanto, o algoritmo é considerado $N-1$ -diagnosticável. □

5. Resultados Experimentais

Nesta seção são apresentados os resultados obtidos através da realização de experimentos de simulações do algoritmo *Hi-Dif* e também da implementação de uma ferramenta prática para a detecção de violação de integridade de nodos com conteúdo replicado na Web.

5.1 Experimentos com Simulações do Algoritmo *Hi-Dif*

Os experimentos através de simulação foram realizados em sistemas com 128 nodos e foram realizados em duas etapas. Inicialmente foram realizados 600 experimentos nos quais até 32 nodos – escolhidos aleatoriamente – são falhos. Em seguida foram realizados mais 600 experimentos nos quais até 64 nodos – escolhidos também aleatoriamente – são falhos. Em todos os experimentos o tipo da falha (falha do tipo crash ou falha por alteração de conteúdo) foi

escolhida de forma aleatória. E em todos os experimentos foram analisadas as situações onde os nodos possuíram probabilidade de falha de 30%, 60% e 90%.

5.1.1 Experimentos com Falha de até 32 Nodos

Foram realizados 600 experimentos em um sistema com 128 nodos considerando-se a probabilidade de no máximo 32 nodos – escolhidos aleatoriamente – falharem. Nos experimentos todos os nodos do sistema estavam inicialmente sem-falha. Todas as falhas ocorreram ao mesmo tempo. As falhas que ocorreram nos nodos também foram escolhidas de forma aleatória, ou seja, um nodo pode ter falhado por falha do tipo crash ou por modificação de conteúdo. Dos 600 experimentos realizados, 200 foram realizados com a probabilidade de 30% dos 32 nodos ficarem falhos. Outros 200 experimentos foram realizados com a probabilidade de 60% de falha dos 32 nodos. E os últimos 200 experimentos foram realizados com a probabilidade de falha dos nodos de 90%. Foram analisadas as médias da latência e do número de testes realizados em cada experimento a partir do momento em que os nodos falharam até o momento em que todos os nodos sem-falha do sistema identificaram a mudança de estado de todos os nodos que falharam.

Na figura 5.1(a) é mostrada a média da latência de cada um dos 200 experimentos realizados. Nota-se que com a probabilidade de falha igual a 30%, a média da latência, ou seja, do número de rodadas de testes para que todos os nodos do sistema completassem o diagnóstico, foi de 4.22 rodadas de testes. Já com a probabilidade de falha igual a 90% a média da latência foi de 3.94 rodadas de testes.

Na figura 5.1(b) é mostrada a média da quantidade de testes realizados de cada um dos 200 experimentos. Pode-se notar que com a probabilidade de falha de 30%, a média de testes realizados foi de 2133 testes. Já com probabilidade de falha igual a 90% a média de testes realizados foi de 2100 testes.

Nota-se com base nos dados da figura 5.1, que mesmo com probabilidade maior de falha, o número de testes se mantém praticamente estável. Nota-se também que quanto maior a probabilidade de falha dos nodos, menor é a média de rodadas de testes necessárias para o diagnóstico completo do sistema. Isto se deve ao fato de que quanto maior o número de nodos falhos, mais testes são realizados pelos nodos sem-falha do sistema e então mais rápido estes nodos identificam o estado de todo o sistema.

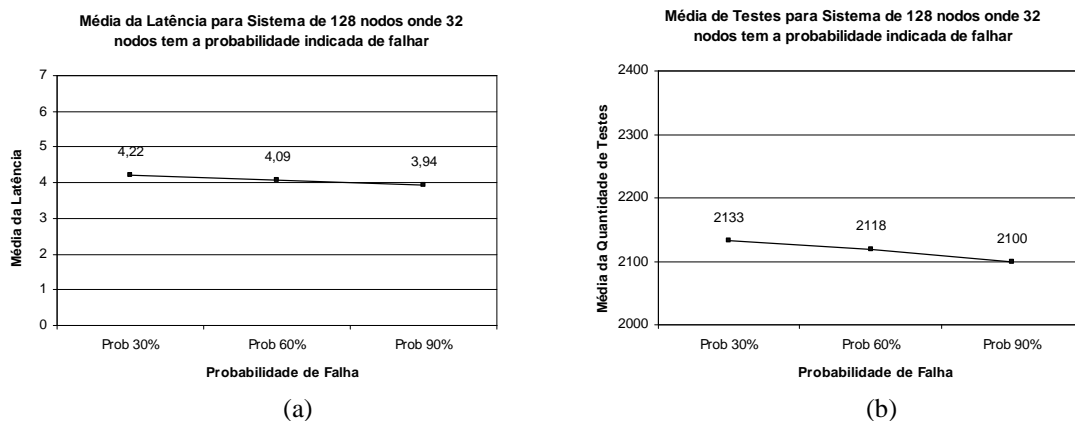


Figura 5.1: Média da latência (a) e média da quantidade de testes (b) em um sistema de 128 nodos onde 32 nodos têm probabilidade de falha de 30%, 60% e 90%.

5.1.2 Experimentos com a Falha de até 64 Nodos

Foram realizados 600 experimentos em um sistema com 128 nodos, similares aos experimentos da sub-seção 5.1.1, mas considerando-se a probabilidade de no máximo 64 nodos – que foram escolhidos aleatoriamente – falharem. As falhas que ocorreram nos nodos também foram escolhidas de forma aleatória. Dos 600 experimentos realizados, também foram analisados de 200 em 200 a probabilidade dos nodos falharem de 30%, 60% e 90%.

Na figura 5.2(a) é mostrada a média da latência de cada um dos 200 experimentos realizados. Nota-se que com a probabilidade de falha dos 64 nodos em 30%, a média da latência foi de 4.96 rodadas de testes. Já com a probabilidade de falha igual a 90% a média da latência foi de 4.25 rodadas de testes. Na figura 5.2(b) é mostrada a média da quantidade de testes realizados de cada um dos 200 experimentos. Pode-se notar que com a probabilidade de falha de 30%, a média de testes realizados foi de 2394 testes. Já com probabilidade de falha igual a 90% a média de testes realizados foi de 2264 testes.

Pode-se notar que nestes experimentos onde houve a probabilidade de ocorrência de até 64 falhas, em comparação com os experimentos onde houve a probabilidade de ocorrência de até 32 falhas, a média da latência continuou entre os valores 4 e 5, mas a média do número de testes teve um aumento maior. Isto ocorre porque quanto maior o número de nodos falhos, mais testes extras são necessários serem executados por cada nodo sem-falha do sistema, aumentando esta média.

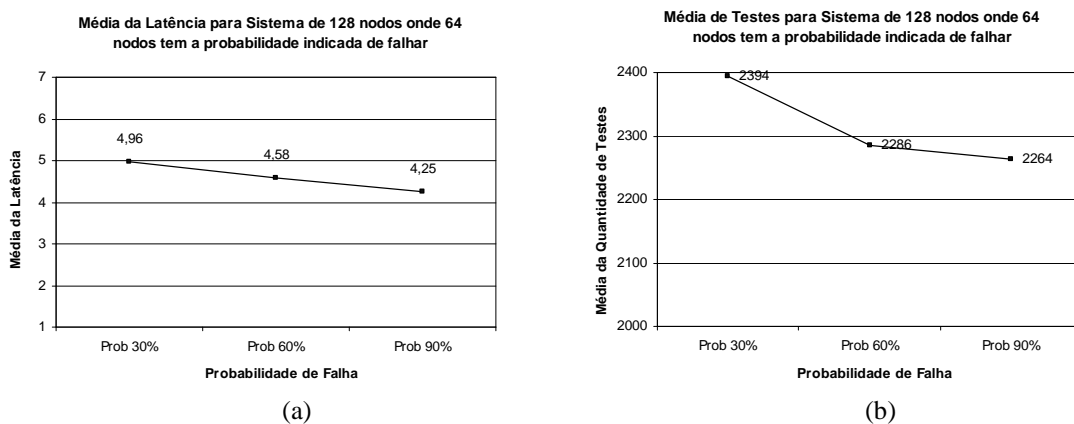


Figura 5.2: Média da latência (a) e média da quantidade de testes (b) em um sistema de 128 nodos onde 64 nodos têm probabilidade de falha de 30%, 60% e 90%.

5.2 Experimento através de Implementação do Algoritmo

Foi realizado experimento através da implementação do algoritmo em uma ferramenta prática para a verificação de integridade de nodos com conteúdo replicado na Web. A ferramenta foi implementada através de dois componentes: um servidor e um cliente. O servidor é o componente responsável pela resposta aos testes e informações solicitadas por outros nodos do sistema. O cliente é o componente no qual o algoritmo *Hi-Dif* está implementado. Este cliente é quem controla as rodadas de testes, realiza os testes, pede informações de diagnóstico para outros nodos do sistema e realiza o diagnóstico do sistema. Estes 2 componentes executam como processos individuais no sistema operacional, ou seja, a ferramenta não executa junto com um servidor Web mas sim podendo até ser executada em uma outra máquina caso não se queira incrementar a utilização de processamento dos servidores Web.

A figura 5.3 mostra um grupo de quatro servidores Web com conteúdo replicado executando a ferramenta implementada. Na figura, a máquina 1 está falha, a máquina 2 está com modificação no conteúdo que é replicado, e as máquinas 3 e 4 estão sem-falha. A ferramenta que executa em cada máquina, realiza testes em todas as máquinas configuradas para a verificação da integridade do conteúdo replicado de todos os outros servidores.

Um experimento foi realizado em um sistema com 32 nodos no qual simultaneamente 8 nodos ficaram falhos sofrendo alteração de seu conteúdo. Na implementação da ferramenta, foi configurado um intervalo de 10 segundos de pausa entre cada rodada de testes. O gráfico da figura 5.4 mostra a quantidade e em que momento que os outros 24 nodos identificam a falha dos 8 nodos. Neste gráfico os dados foram agrupados de 10 em 10 segundos, ou seja, todos os nodos que identificaram a todas as falhas nos primeiros 10 segundos, aparecem na primeira coluna, e assim por diante. Pode-se notar que nos primeiros 10 segundos após a falha dos 8 nodos, 2 nodos identificam as falhas. Nota-se também que a maior parte dos nodos identificam a falha entre o

intervalo de 20 e 30 segundos, quando 9 nodos identificam todas as falhas. Após 50 segundos todos os nodos já haviam identificado as falhas dos 8 nodos.

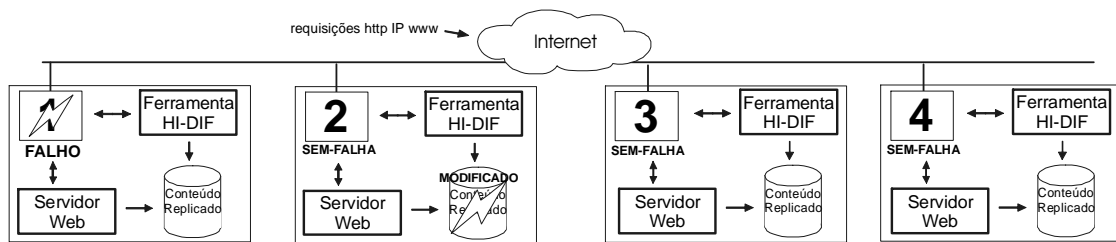


Figura 5.3: Um grupo de servidores Web executando a ferramenta implementada.

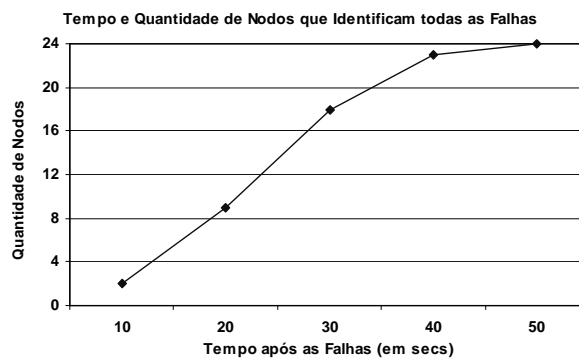


Figura 5.4: Experimento em sistema de 32 nodos onde 8 nodos falham, mostrando o tempo e a quantidade de nodos que identificam as falhas.

6. Conclusão

Este trabalho apresentou um novo modelo genérico de diagnóstico hierárquico adaptativo distribuído e baseado em comparações, e apresentou um novo algoritmo, *Hi-Dif*, baseado neste modelo, para a detecção distribuída de violação de integridade em sistemas com conteúdo replicado em uma rede como, por exemplo, a Internet.

No algoritmo *Hi-Dif*, um nodo sem-falha testa outro nodo do sistema para classificar seu estado. O algoritmo classifica os nodos do sistema em conjuntos de acordo com o resultado dos testes e um teste é realizado através do envio de uma tarefa para um par de nodos do sistema. Após o recebimento das duas saídas da execução destas tarefas, o testador compara estas saídas e, se a comparação indicar igualdade, os nodos são classificados no mesmo conjunto de nodos. Por outro lado, se a comparação das saídas indicar diferença, os nodos são classificados em conjuntos distintos, de acordo com o resultado da tarefa. Um dos conjuntos contém os nodos sem-falha do sistema. Uma diferença do modelo apresentado para outros modelos propostos anteriormente, é que a comparação, realizada por um nodo sem-falha, das saídas produzidas por dois nodos falhos pode resultar em igualdade. Por esta classificação, o algoritmo permite a identificação dos nodos que estão com conteúdo modificado e diferenciá-los, por exemplo, de outros nodos que venham a possuir outras modificações de conteúdo.

Foi demonstrado que o algoritmo *Hi-Dif* possui uma latência de $\log_2 N$ rodadas de testes para um sistema de N nodos. O algoritmo também é $(N-1)$ -diagnosticável e o número máximo de testes requeridos pelo algoritmo é de $O(N^2)$, no pior caso. Experimentos através de simulações do algoritmo e experimentos através da implementação de uma ferramenta prática para a verificação de integridade de conteúdo replicado na Web também foram realizados. Os resultados dos experimentos comprovam o número máximo de testes necessários e a latência do algoritmo.

Trabalhos futuros incluem o estudo da aplicação do algoritmo em redes *peer-to-peer*, e a implementação de um pacote para facilitar a instalação e configuração da ferramenta implementada para a Web.

Referências

- [1] CERT Coordination Center, <http://www.cert.org>, Acessado em 05/09/2004.
- [2] ALDAS, Analytisches Labor Dr. Axel Schumann, <http://www.aldas.de>, Acessado em 05/10/2003.
- [3] S. Garfinkel, G. Spafford, e A. Schwartz, *Practical Unix & Internet Security*, O'Reilly, 3a. ed., Fev. 2003.
- [4] A. Subbiah, D. M. Blough, "Distributed Diagnosis in Dynamic Fault Environments," *IEEE Transaction on Parallel and Distributed Systems*, Vol 15, No. 5, pp. 453-467, Mai. 2004.
- [5] L. C. P. Albin, E. P. Duarte Jr., "Generalized Distributed Comparison-Based System-Level Diagnosis," *2nd IEEE Latin American Test Workshop*, pp. 285-290, Set. 2001.
- [6] D. Ingham, S. K. Shrivastava, F. Panzieri, "Constructing Dependable Web Services," *IEEE Internet Computing*, Vol 4, No. 1, pp 25-33, Jan/Fev 2000.
- [7] B. Tan, S. Foo, S. C. Hui, "Monitoring Web Information Using PBD Technique," *Proc. 2nd International Conference on Internet Computing (IC'2001)*, Las Vegas, USA, pp. 666-672, Jun. 2001.
- [8] Url Minder, <http://www.netmind.com/URL-minder/URL-minder.html>. Acessado em 22/09/2003.
- [9] B. Lu, S. C. Hui, Y. Zhang, "Personalized Information Monitoring Over the Web," *1st International Conference on Information Technology & Applications (ICITA 2002)*, Nov. 2002.
- [10] V. Boyapati, K. Chevrier, A. Finkel, N. Glance, T. Pierce, R. Stockton, C. Whitmer, "ChangeDetectorTM: A Site-Level Monitoring Tool for the WWW," *International World Wide Web Conference*, Hawaii, USA, pp. 570-579, Mai. 2002.
- [11] S.-J. Lim, Y.-K. Ng, "An Automated Change-detection Algorithm for HTML documents Based on Semantic Hierarchies," *Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*, Heidelberg, Alemanha, pp. 303-312, Abr. 2001.
- [12] M. Malek, "A Comparison Connection Assignment for Diagnosis of Multiprocessor Systems," *Proc. 7th International Symp. Computer Architecture*, pp. 31-36, 1980.
- [13] K. Y. Chwa, S. L. Hakimi, "Schemes for Fault-Tolerant Computing: A Comparison of Modularly Redundant and t-Diagnosable Systems," *Information and Control*, Vol. 49, pp. 212-238, 1981.
- [14] J. Maeng, M. Malek, "A Comparison Connection Assignment for Self-Diagnosis of Multiprocessor Systems," *Digest 11th International Symp. Fault Tolerant Computing*, pp. 173-175, 1981.
- [15] A. Sengupta, A. T. Dahbura, "On Self-Diagnosable Multiprocessor Systems: Diagnosis by Comparison Approach," *IEEE Transactions on Computers*, Vol. 41, No. 11, pp. 1386-1396, 1992.
- [16] D. M. Blough, H. W. Brown, "The Broadcast Comparison Model for On-Line Fault Diagnosis in Multicomputer Systems: Theory and Implementation," *IEEE Transactions on Computers*, Vol. 48, pp. 470-493, 1999.
- [17] D. Wang, "Diagnosability of Hypercubes and Enhanced Hypercubes under the Comparison Diagnosis Model," *IEEE Transactions on Computers*, Vol. 48, No. 12, pp. 1369-1374, 1999.
- [18] T. Araki, Y. Shibata, "Diagnosability of Butterfly Networks under the Comparison Approach," *IEICE Trans. Fundamentals*, Vol E85-A, No. 5, Mai. 2002.
- [19] J. Fan, "Diagnosability of Crossed Cubes," *IEEE Transactions on Computers*, Vol. 13, No. 10, pp. 1099-1104, Out. 2002.
- [20] E. P. Duarte Jr., A. Brawerman, L. C. P. Albin, "An Algorithm for Distributed Hierarchical Diagnosis of Dynamic Fault and Repair Events," *Proc. IEEE International Conference on Parallel and Distributed Systems 2000*, pp. 299-306, 2000.