A Hierarquical Distributed Diagnosis Algorithm Based on Clusters with Detours

Elias Procópio Duarte Jr.
Luiz Carlos P. Albini
Alessandro Brawerman
André L. P. Guedes

Federal University of Paraná, Dept. Informatics
Cx. Postal 19081 – Curitiba – 81531-990 PR – Brazil
Phone: +55-41-267-5244 Fax: +55-41-267-6874
e-mail: {elias,albini,ale,andre}@inf.ufpr.br

Technical Report # RT002/2003, Federal University of Parana, Dept. Informatics, http://www.inf.ufpr.br/info/techrep/index.html

Abstract

A distributed system-level diagnosis algorithm allows the fault-free nodes of a system to diagnose the state of all nodes in the system. In this paper we present a new hierarchical adaptive distributed system-level diagnosis algorithm, *Hi-ADSD with Detours*. A previously published algorithm [6] has latency at most $log_2^2 N$, but requires $O(N^2)$ tests in one round in the worst case. While the latency of the proposed algorithm is the same, the number of tests executed is reduced. Nodes running the new algorithm are grouped in clusters. If a tested node is faulty, instead of executing more tests, the tester will try to obtain information about the rest of the cluster from nodes tested fault-free outside the cluster, such that the diagnosis of the system is not delayed. Each such alternative path to a cluster is called a *detour*. An extra test is executed on a given cluster only when no detour is available. The worst case of the algorithm's latency is formally proved. Simulation results are presented.

Index Terms: System-Level Diagnosis, Distributed Diagnosis, Adaptive Diagnosis, Network Fault Monitoring.

1 Introduction

Consider a system composed of N nodes, which can be either faulty or fault-free. Assume that the system is fully connected, i.e. there is a link between any pair of nodes, and links

do not become faulty. A distributed system-level diagnosis algorithm allows the fault-free nodes in that system to determine the state of all nodes [4]. Nodes of a diagnosable system are capable of executing tests on other nodes. A fault-free tester is assumed to be able to determine correctly whether a tested node is faulty or fault-free [1, 2, 3]. In order to implement this assumption, the specification of the testing procedure often depends on the particular system technology. Distributed system-level diagnosis provides an efficient way to build fault-tolerant network monitoring systems [5].

In [6] Duarte and Nanya introduced the Hierarchical Adaptive Distributed System-Level Diagnosis (Hi-ADSD) algorithm. Hi-ADSD was implemented integrated to a network management system based on the Internet standard management framework, SNMP (Simple Network Management Protocol). The algorithm presents a diagnosis latency of at most $log^2 N$ testing rounds for a system of N nodes. All logarithms used in this work are base 2. Nodes are grouped in progressively larger logical clusters. Tests are executed in a hierarchical fashion, starting at the cluster with two nodes, going on to the cluster with four nodes, and so on, until the cluster with $N/2$ nodes is tested. In order to get information about a given cluster, a node executes tests until a fault-free node is found in the tested cluster, or all the cluster's nodes are tested faulty. Considering all testers, the largest number of tests executed in one round is $N^2/4$ tests in the worst case. This happens when $N/2$ nodes are faulty, these nodes are all in the same cluster, and the remaining $N/2$ nodes test that cluster in the same testing round.

In this work we present a new algorithm, Hi-ADSD with Detours, that requires less tests per testing rounds, while keeping the latency at $log^2 N$ rounds. The new algorithm takes advantage of the fact that a tester can obtain information about the nodes in one given cluster from nodes outside that cluster, without increasing the diagnostic latency. Each such alternative path to the cluster is called a *detour*.

Whenever a node tests a fault-free node in a given cluster, it gets diagnostic information about that whole cluster from the tested fault-free node. However, if the first node tested in a cluster is faulty, before executing more tests, the tester will try to get information about the rest of the cluster from nodes tested fault-free outside the cluster. The tester will look for detours to that cluster in the next $logN$ testing rounds, from clusters smaller than the current cluster, as shown in figure 1. In the next round this cluster is tested, if the tester has not found detours to nodes in the tested cluster, then it executes

tests sequentially on the cluster's nodes until a fault-free node is found or all nodes are tested faulty.

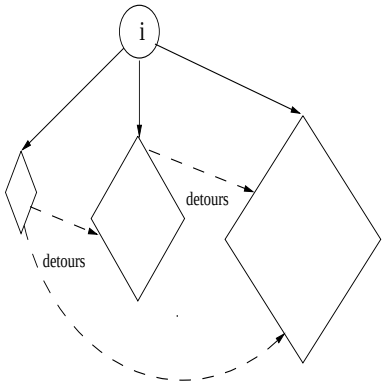

Figure 1: A node only employs detours to a given cluster from smaller clusters.

Other approaches to system-level diagnosis are described in [7, 8, 9]; other hierarchical approaches include [10, 11, 12, 13].

The rest of the paper is organized as follows. Section 2 contains preliminary definitions. In section 3 the new algorithm is specified and section 4 describes example executions. In section 5 the algorithm's latency is formally proved. Section 6 shows experimental results of diagnosis obtained through simulation. This is followed by conclusions in section 7.

2 Preliminary Definitions

Consider a system S consisting of a set of N nodes, n_0, n_1,...,n_{N-1}. We alternatively refer to node n_i as *node i*. The system is assumed to be fully connected, i.e. there is a communication link between any two nodes. Each node n_i is assumed to be in one of two states, *faulty* or *fault-free*. An *event* is defined as a change in the state of a node, either from *faulty* to *fault-free* or from *fault-free* to *faulty*. The collection of states of all nodes is the system's fault situation. Nodes perform tests on other nodes in a testing interval, and fault-free nodes report test results reliably.

Nodes are grouped in *clusters* for the purpose of testing. Clusters are sets of nodes. The size of a cluster is the number of nodes in the cluster. Initially, N is assumed to be a power of 2, and the system itself is a cluster of N nodes. A general cluster of p nodes n_j,...,n_{j+p-1} where j MOD $p = 0$, and p is a power of two, is recursively defined as either

a node, when $p = 1$, or the union of two clusters, one containing nodes n_j,...,$n_{j+p/2-1}$ and the other containing nodes $n_{j+n/2}$,..., n_{j+n-1}.

At each testing interval, fault-free nodes test nodes of one cluster. The lists of ordered nodes in which a given node i tests the nodes of a given cluster of size 2^{s-1} are denoted by $C_{i,s}$. An expression that completely characterizes list $C_{i,s}$ is given below:

$$C_{i,s} = \ i \text{ XOR } 2^{s-1},\ C_i \text{ xor } {}_{2^{s-1},s-1},\ C_i \text{ xor } {}_{2^{s-1},s-2},\ \ldots,\ C_i \text{ xor } {}_{2^{s-1},1}$$

A *testing round* is defined as the period of time in which every fault-free node in the system has tested at least one fault-free node or all nodes faulty in one cluster. The algorithm's latency is defined as the number of testing rounds that all nodes running the algorithm require to complete diagnosis of an event. An event occurs after the previous event has been fully diagnosed.

The *Tested Fault-Free* graph, $T(S)$, is a directed graph whose nodes are the nodes of S. There is an edge directed from node i to node j if node i has tested node j as fault-free in the most recent testing interval in which it tested the cluster to which node j belongs. When all nodes in the system are fault-free, $T(S)$ is a hypercube.

Let the *diagnostic distance* from node i to node p, called $d_{i,p}$, be the number of edges in the shortest path from node i to node p in $T(S)$ when all nodes are fault-free, i.e. in the hypercube. For instance, in figure 2 the diagnostic distance from node 0 to node 5 is $d_{0,5} = 2$. Furthermore, let set $D_{i,r}$ be the set of every node p such that $d_{i,p} \leq r$.

Let $R_{i,s,p}$ be the set of nodes that can be reached by node i from node p with a diagnostic distance less than or equal to s, furthermore node $k \in R_{i,s,p}$ only if $d_{i,k} \geq d_{i,p} + d_{p,k}$. As an example, figure 2 shows $R_{0,3,2}$. $R_{i,s,p}$ is given by the expression below:

$$R_{i,s,p} = \{k \in D_{p,s-d_{i,p}} \mid d_{i,k} = d_{i,p} + d_{p,k}\}$$

Let node i test node j in a given testing round such that node j belongs to a given $C_{i,s}$. A *detour* from node i to node j is a path in $T(S)$ from node i to node j that passes through nodes not in the $C_{i,s}$ to which node j belongs. Furthermore, a detour has exactly the same number of edges as the shortest path from node i to node j when all nodes are fault-free. Figure 3 shows all detours employed by node 0 to get information about nodes in the tested cluster with four faulty nodes. In this figure, node 0 employs detours to get information about node 5, node 6 and node 7 from node 1, node 2 and node 3 respectively.

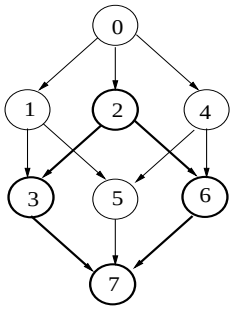

Figure 2: An example $R_{i,s,p}$: $R_{0,3,2}$.

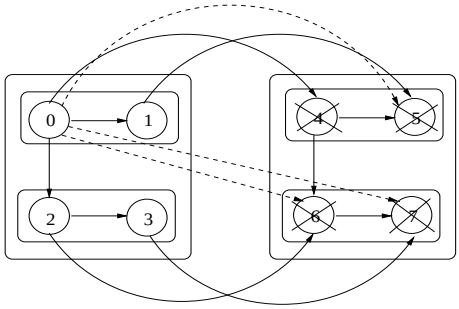

Figure 3: Dashed lines show tests node 0 avoids by using detours.

When the tested node in a given cluster is faulty and the tester does not obtain information about the cluster's remaining nodes, those nodes are said to be *blocked*. Set $B_{i,s}$ contains all blocked nodes in a given $C_{i,s}$.

3 Algorithm Specification

Nodes running Hi-ADSD with Detours execute tests on nodes of one cluster at each testing interval. In the first testing interval, the cluster which contains only one node is tested, in the second testing interval the cluster which contains 2 nodes is tested, and so on until the largest cluster which contains $N/2$ nodes is tested. After that, in the next testing interval, the cluster which contains one node is tested again and the whole process is repeated.

When a fault-free node is tested, the tester obtains diagnostic information about the whole cluster to which the tested node belongs. However when a faulty node is tested, the tester cannot obtain diagnostic information about the remaining nodes in the tested cluster, which are said to be blocked. Instead of executing more tests on the blocked nodes (like a node running Hi-ADSD would do) the tester will first try to find detours to those nodes, from nodes that are tested fault-free in other clusters. If no detours are

found, then extra tests will be executed on the blocked nodes in the next testing interval in which their cluster is tested. Note that, by definition, node i can only employ detours to nodes of a given $C_{i,s}$ from nodes that belong to a $C_{i,s'}$, such that $s' < s$.

Two functions are employed by the testing strategy: function `more-info` and function `more-tests`, defined below. The tester runs function `more-info` after a fault-free node is tested, to decide if detours through the tested node to blocked nodes are necessary. Function `more-tests` is executed after a faulty node is tested, to decide if more tests must be executed in the same cluster, i.e. if no detours were found to the blocked nodes. These functions are defined below.

Function `more-info` returns a list of nodes of other clusters about which the tester needs to obtain information from the tested node. The function is given below, where node i is the tester, node p is the tested fault-free node in $C_{i,s'}$, list $R_{i,log_N,p}$ contains all nodes about which node p can provide information to node i, and set $B_{i,s}$ contains the blocked nodes in a given $C_{i,s}$.

$$\texttt{more-info}(i,p) = R_{i,log_N,p} \cap B_{i,s}, s' < s \leq logN, p \in C_{i,s'}$$

Function `more-tests` is given by the formula below, where node i is the tester, node p is the tested faulty node, list $c_{i,s,p}$ contains all nodes about which node p can provide information to node i, where s varies according to the $c_{i,s}$ to which node p belongs:

$$\texttt{more-tests}(i,s) = B_{i,s} - R_{i,s,p} \forall p \in C_{i,s'}, 1 \leq s' < s \text{ tested fault-free}$$

The algorithm in pseudo-code is given below.

```
ALGORITHM Hi-ADSD with Detours {at node i}
 FOR s := 1 TO logN DO Bi,s = {};
 REPEAT
   FOR s := 1 TO logN DO
     p := first node in Ci,s;
     test(p);
     IF p is fault-free
     THEN get cluster diagnostic information;
          Bi,s := Bi,s - Ci,s;
          get information about nodes returned by more_info(i,p);
          Bi,s := Bi,s - more_info(i,p);
     ELSE {tested node is faulty}
          Bi,s := Bi,s U Ci,s - {p};
          WHILE more-tests(i,s) <> {} DO
             k := first node more_tests(i,s);
             test(k);
             Bi,s := Bi,s - {k};
             IF k is fault_free
             THEN get information about nodes in Bi,s;
                  Bi,s := Bi,s - Ci,s;
                  get information about nodes returned by more_info(i,k);
                  Bi,s := Bi,s - more_info(i,k);
             END IF;
          END WHILE;
     END IF;
     SLEEP(Testing Interval);
   END FOR;
 FOREVER
```

4 Example Executions

Consider the system in figure 3. When node 0 tests node 4 in $C_{0,3}$ as faulty, it updates Bi, s with $5, 6, 7$. Node 0 does not execute extra tests on this cluster, because it will try to get information about the blocked nodes through detours. The next time node 0 tests node 1 it gets information about blocked nodes $5, 7$, and the next time node node 0 tests node 2 it gets information about blocked node 6. This execution is analogous for other fault-free nodes in the system. Thus all fault-free avoid extra tests.

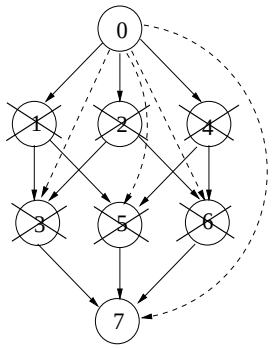

Figure 4: Node 0 executes an extra test on node 3, node 5, node 6 and node 7.

Consider the execution of Hi-ADSD with Detours at node 0 in the system shown in figure 4. Initially node 1 is tested faulty, then in the next round node 2 is also tested

faulty so node 0 runs `more-tests(0,2)` which returns node 3, that is then tested faulty. After node 4 is tested faulty function `more-tests(0,3)` returns nodes 5, 6 and 7, which are all sequentially tested as both node 5 and node 6 are faulty.

5 Latency

In this section we formally prove that the the latency of the algorithm is log^2N testing rounds in the worst case. We prove that the usage of detours does not affect the worst-case latency, being the same as that of Hi-ADSD [6].

Theorem 1

All fault-free nodes running Hi-ADSD with Detours take at most log^2N testing rounds to complete diagnosis.

Proof:

Consider a new event which occurs at node e. All nodes that have diagnostic distance equal to 1 with respect to node e diagnose the new system fault situation in at most $logN$ testing rounds. This happens because those nodes test node e at least once every $logN$ testing rounds.

Next, assume that all fault-free nodes with diagnostic distance equal to $k < i$ get diagnostic information about node e's new event in at most $k * logN$ testing rounds.

Now consider a fault-free node a with distance i to node e. In case there is a fault-free node, say node b, with distance 1 to node a and distance $(i-1)$ to node e, node a can get information about node e from node b, either directly or using node b as a detour, whitch takes at most $logN$ testing rounds. Thus it takes at most $(i-1)*logN+logN = i*logN$ testing rounds for the node a to diagnose node e's new event using node b.

If all nodes with distance $(i-1)$ to node e and distance 1 to node a are faulty, node a must get information about node e from a fault-free node with diagnostic distance $(i-j)$ to node e and distance j to node a, say, node c, which takes at most $j * logN$ rounds. As for node c it takes $(i-j) * logN$ rounds to diagnose node e's new event, it will take $j * logN + (i-j) * logN = i * logN$ rounds for node a to diagnose the event.

As the largest distance is $logN$, it may take at most $logN * logN = log^2N$ testing rounds for all fault-free nodes in the system to the complete diagnosis. □

Fault Situation	*Hi-ADSD*	*Hi-ADSD with Detours*
Average # Tests (32 random faulty nodes)	383	285
Hi-ADSD's Worst-Case	1184	191
Hi-ADSD with Detours' Worst-Case	384	384

Table 1: Comparison of the number of tests required by Hi-ADSD and Hi-ADSD with Detours.

6 Simulation Results

In this section, we present experimental results of Hi-ADSD with Detours obtained through simulation. The simulation was conducted using the discrete-event simulation language SMPL [14]. Nodes were modeled as SMPL facilities, and each node was identified by an SMPL token number. Three kinds of events were defined: test, fault, and repair. We conducted several experiments with networks of different sizes. We present results of two different experiments. In the first experiment, we compare the average number of tests required by Hi-ADSD and Hi-ADSD with Detours in different fault situations. In the second experiment we progressively increase the number of faulty nodes in a network of 64 nodes, and show that the number of tests needed by fault-free nodes running Hi-ADSD with Detours decreases as the number of faulty nodes increases.

6.1 A Comparison of the Average Number of Tests Required

The purpose of this experiment is to compare the number of tests required by Hi-ADSD and Hi-ADSD with Detours in different fault situations. We considered a system of 64 nodes. Both algorithms require the same number of tests when all nodes are fault-free. When the number of faulty nodes grows, Hi-ADSD requires more tests, while Hi-ADSD with Detours requires *less* tests.

The first row in table 1 shows the average number of tests required by Hi-ADSD and Hi-ADSD with Detours when 32 nodes, chosen at random, are faulty. In this experiment we can clearly see how the concept of detours has an impact on the average number of tests required to complete diagnosis. We considered the number of tests required by fault-free nodes in $logN$ consecutive testing rounds. On average, Hi-ADSD required 383 tests, while Hi-ADSD with Detours required 285 test, a 25% improvement.

# Faulty Nodes	*# Tests to diagnose the system*
0	384 (= $NlogN$)
1	378
2	374
4	369
6 (= $logN$)	367
12 (= $2logN$)	350
32 (= $N/2$)	320

Table 2: Number of tests required for all fault-free nodes to diagnose a 64-node system.

The second row in table 1 shows the number of tests required by both algorithms considering the worst case of Hi-ADSD, in which a cluster of $N/2$ nodes is faulty, and all the nodes in the other cluster test the faulty cluster in the same testing round; the total number of tests is $N^2/4$. In this situation, Hi-ADSD requires 1184 tests each $logN$ rounds, while Hi-ADSD with Detours requires only 191 tests.

Finally, the third row in table 1 shows the number of tests required by both algorithms considering the worst case of Hi-ADSD, in which all nodes are fault-free. In this case, both algorithms need 384 ($NlogN$) tests per $logN$ rounds.

From the results of this experiment we conclude that for every case Hi-ADSD with Detours requires a number of tests less or equal to that required by Hi-ADSD. On average, the number of tests required by Hi-ADSD with Detours is smaller than that of Hi-ADSD.

6.2 Number of Tests Required by Hi-ADSD with Detours

The purpose of this experiment is to show the number of tests required by the fault-free nodes in different situations. To create those situations we progressively increase the number of faulty nodes in a network of 64 nodes.

Table 2 shows the number of tests executed when the number of faulty nodes increases in the system. Initially all nodes are fault-free, then the number of faulty nodes is increased from 1 up to $N-1$. Simulation results show that the maximum number of tests occurs when all nodes are fault-free, and the total number of tests performed by the fault-free nodes to diagnose the system decreases as the number of faulty nodes increases.

7 Conclusions

In this work we introduced a new hierarchical adaptive distributed system-level diagnosis algorithm, Hi-ADSD with Detours, that allows all fault-free nodes in a fully-connected system to complete diagnosis in at most log^2N testing rounds, employing less tests than a previously published algorithm, Hi-ADSD. This algorithm also presents a latency of log^2N rounds, but nodes can employ up to $O(N^2)$ tests to get information about the whole system. Proofs were given for the algorithm's latency. Experimental results obtained through simulation were presented.

Hi-ADSD with Detours is a practical algorithm that can be used to monitor real local area networks. Considering the number of tests required, the impact on network performance is lower than that of previous algorithms with the same latency. Future work includes the evaluation of the algorithm under a dynamic fault situation.

Acknowledgements

The analysis of Hi-ADSD made by Prof. Douglas Blough, of Georgia Tech, was key for us to start developing Hi-ADSD with Detours. The recursive definition of list $C_{i,s}$ was proposed by Andreas Kiefer.

References

[1] 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.

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

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

[4] 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.

[5] 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.

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

[7] A. Sengupta, and A.T. Dahbura, "On Self-Diagnosable Multiprocessor Systems: Diagnosis by the Comparison Approach," *IEEE Transactions on Computers*, Vol.41, pp.1386-1396, 1992.

[8] D.M. Blough, and H.W. Brown, "The Broadcast Comparison Model for On-Line Fault Diagnosis in Multicomputer Systems: Theory and Implementation," *IEEE Transactions on Computers*, pp.470-493, Vol.48, No.5, May 1999.

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

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

[11] 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

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

[13] 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.

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