Distributed Integrity Checking for Systems with Replicated Data

Roverli Pereira Ziwich, Elias Procópio Duarte Jr.
Federal University of Paraná, Dept. Informatics
P.O.Box 19018 – 81531-990, Curitiba, PR Brazil
{roverli,elias}@inf.ufpr.br

Luiz Carlos Pessoa Albini
Unicenp, Dept. Informatics
R. Prof. Parigot de Souza, 5300, Curitiba, PR Brazil
albini@unicenp.br

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, detecting unauthorized Web page modifications. Fault-free nodes running* Hi-Dif *send a task to two other nodes and the task results are 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, we prove that the algorithm has latency equal to* $\log_2 N$ *testing rounds in the worst case; that the maximum number of tests required is* $O(N^2)$*; and, that the algorithm is* $(N-1)$*-diagnosable. Experimental results obtained by simulation and by the execution of a tool implemented applied to the Web are presented.*

1. Introduction

The number of Internet users has been estimated to be close to 600 million persons. The dependability of the network has become increasingly important for millions of organizations and individuals. On the other hand, attacks and acts of vandalism, such as non-authorized modifications of Web pages, are becoming very common [1].

According to [2], a service specialized on recording site invasions, in 2001 about 22,400 sites were invaded and had their contents updated by unauthorized persons or processes. In 2003, this figure grew to over 137,000 sites. Therefore, there is an pressing need for efficient monitoring systems, that are able to check the integrity of data made available through the network. In this work integrity is defined as the property that data is only modified by authorized entities.

The main purpose of a monitoring system is to discover which units of the system are faulty. The monitoring system must be fault tolerant, i.e. capable of working correctly even in the presence of faulty units. This work uses distributed [3] comparison-based [4] system-level diagnosis to check the integrity of data replicated across a network, such as the Internet. An example of the practical application of this work is the diagnosis of vandalism on replicated Web servers, also called Web clusters [5].

A large number of tools and systems that deal with the detection of modifications in Web sites have been published [6, 7, 8, 9, 10]. Many of these tools even allow specific parts of a given Web page to be monitored. Most tools execute comparisons to determine that the contents have been modified, keeping a local copy or a checksum of the data to be checked. An example is WebMon [6]. Other tools work as a service within the Web, such as URL-minder [7].

The new system-level diagnosis model proposed in this work is distributed, i.e. there is no centralized component responsible for monitoring. In fact, the units, which keep the replicated data, monitor each other according to a comparison-based distributed system-level diagnosis model. Each fault-free node of the system runs an algorithm, *Hi-Dif*, which is also introduced in this paper. Units running *Hi-Dif* perform tests over other units in a distributed and hierarchical fashion. A test consists of sending a task to a pair of nodes. After receiving the two outputs for a task, the tester compares the two outputs and classifies the nodes in sets according to the result of the task. One of these sets contains the fault-free nodes of the system. Formal proofs for the latency, maximum number of tests and diagnosability of the algorithm are also presented. Experimental results obtained from the execution of a tool applied to the Web, and also results obtained through simulations are presented.

The rest of the paper is organized as follows. Section 2 presents a survey of comparison-based system-level diagnosis. Section 3 describes the new system-level diagnosis model. Section 4 includes the specification and formal proofs of *Hi-Dif*. Section 5 presents simulation results. Section 6 presents a tool for checking the integrity of Web clusters. Section 7 concludes the paper.

2. Comparison-Based Diagnosis

The main objective of system-level diagnosis is to identify which units of the system are faulty and which are fault-free [3]. Several models for system-level diagnosis

can be found in the literature. Models for comparison-based system-level diagnosis were initially proposed by Malek [11], and by Chwa and Hakimi [12]. These models assume that, in a system of N units, the comparison of the outputs produced by any pair of units is possible. These models also assume the existence of a central observer which collects and maintains information about the tasks outputs. This central observer also performs the diagnosis of the system based on comparison results, thus determining which are the faulty units of the system. The central observer is a trustful unit which cannot suffer any event. The difference between these two models is that in [12] two faulty units can produce the same output for a task, so that the comparison of these outputs results in a match.

The MM model, proposed by Maeng and Malek [13], is an extension of the model proposed by Malek [11]. This model assumes that the comparison results are sent to a central observer which performs the complete diagnosis of the system, but it allows the comparison of task outputs to be made by the units themselves. The only restriction is that the unit which performs the comparison must be different from the two units which perform the task. Sengupta and Dahbura in [14] present a generalization of the MM model, called MM*. This model allows the comparison unit to be one of the units that perform the task.

Blough and Brown in [15] present a comparison-based system-level diagnosis model based on reliable broadcast. In this model a task is assigned as input to a pair of different nodes. These two nodes execute the task and broadcasts their outputs to all nodes of the system, including themselves. The output comparisons are made by each fault-free node.

In [16] Wang proposes a comparison-based system-level diagnosis for both hypercubes and enhanced hypercubes. Araki and Shibata in [17] present the diagnosability of butterfly networks with comparison-based system-level diagnosis. Also considering comparison-based system-level diagnosis, Fan in [18] evaluates the diagnosability of crossed cubes, a variation of hypercubes, but with smaller diameter.

The Generalized Distributed Comparison-Based System-Level diagnosis model is presented by Albini and Duarte in [4]. This is the first hierarchical comparison-based diagnosis model that is not limited to crash faults. Fault-free nodes test other nodes of the system to identify their state. This model has the following assumption: when a fault-free node compares the outputs produced by two faulty nodes, the result always show a difference.

3. A General Comparison-Based Diagnosis Model

In this work we present a new general hierarchical comparison-based adaptive distributed system-level diagnosis model. We assume that the diagnosable system S consists of N nodes and is fully connected. The system is represented by graph $G=(V, E)$, where V is a set of vertices and E is a set of edges. Each vertex in the graph corresponds to a node of the system and the edges correspond to the communication links. In this model links do not become faulty. Nodes of the system can be either *faulty* or *fault-free*. A node becomes faulty by either crashing or by replying arbitrarily to a given query. A change of the state of a node is called an *event*.

Fault-free nodes test other nodes and based on test results, tested nodes are classified in sets. A test is performed by sending a task to two nodes. Task 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. If nodes are classified in more than one set, then there are faulty nodes in the system.

The new model allows the identification of crashed nodes, and also allows the identification of nodes that have not crashed but do not reply the correct, expected data.

The following assumptions are made about the system: 1) a fault-free node comparing outputs produced by two fault-free nodes always produces a match; 2) a fault-free node comparing outputs produced by a faulty node and a fault-free node always produces a mismatch; and, 3) the time interval required for a fault-free node to produce an output for a task is bounded.

One key difference of the proposed model to previously published models is that the new model allows the comparison performed by a fault-free node of the task outputs of two faulty nodes to be equal to each other.

A multi-graph, $M(S)$, is defined to represent the way tests are executed in the system. $M(S)$ is a directed multi-graph defined over graph G, when all nodes of the system are fault-free. The vertices of $M(S)$ are the nodes of system S. Each edge of $M(S)$ represents that a node is sending a task to another node.

As an example consider figure 1, where node 0 sends a task to itself and to node 1, and sends another task to itself and to node 2. In this example the edges are $(0, 0)_1$, $(0, 1)_0$, $(0, 0)_2$ and $(0, 2)_0$, and all edges are from node 0 to the other nodes or to node 0 itself. Edge $(0, 1)_0$ indicates that node 0 sent a task to node 1 and the output of this task will be compared with the output returned by node 0, therefore the edge $(0, 0)_1$ must also be in the multi-graph.

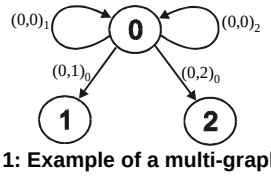

Figure 1: Example of a multi-graph *M*(*S*).

4. The Algorithm

In this section a hierarchical comparison-based adaptive distributed diagnosis algorithm, called *Hi-Dif*, is presented. The algorithm is based on the diagnosis model

presented in the previous section. This new algorithm allows distributed integrity checking for systems with replicated data. This new algorithm also allows the identification of nodes that have the same replicated data.

4.1 Description of *Hi-Dif*

The algorithm employs a testing strategy represented by a graph, called *tested fault-free graph*, $T(S)$. This graph is a directed graph defined on $M(S)$. $T(S)$ is a hypercube when all nodes in the system are fault-free and when the number of nodes is a power of two. Figure 2(a) shows $T(S)$ for a system with 8 nodes.

The *diagnostic distance* between node *i* and node *j* is defined as the shortest distance between node *i* and node *j* in $T(S)$. For example, in figure 2(a) the diagnostic distance between node 1 and node 3 is 2.

The *tested fault-free graph of node i*, $T_i(S)$, is a directed graph defined on $T(S)$ and shows how the diagnostic information flows in the system. There is an edge in $T_i(S)$ from node *a* to node *b* if there is an edge in $T(S)$ from node *a* to node *b* and the diagnostic distance between node *i* and node *a* is shorter than the diagnostic distance between node *i* and node *b*. Figure 2(b) shows $T_0(S)$ for a system of 8 nodes. Nodes with diagnostic distance 1 to node *i* are called *sons* of node *i*. In figure 2(b) the sons of node 0 are nodes 1, 2 and 4.

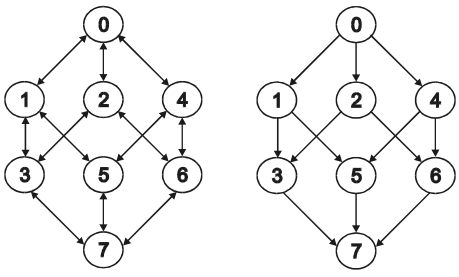

Figure 2: (a) $T(S)$ for a system with 8 nodes. (b) $T_0(S)$ for a system with 8 nodes.

A *testing round* is defined as the time interval in which all fault-free nodes obtain diagnostic information about all nodes of the system. An assumption is made that after node *i* tests node *j* in a given testing round, another event cannot occur at node *j* in the same testing round. This assumption is necessary to guarantee the correct diagnostic information propagation.

The testing strategy groups nodes in clusters [19]. A function called $c_{i,s,p}$ defines the list of nodes about which node *i* can obtain diagnostic information through a given node *p*, with diagnostic distance equal or less than *s* in $T_i(S)$. In *Hi-Dif s* is always equal to $log_2 N$, i.e. each cluster has always $N/2$ nodes. Figure 3 depicts the cluster division for a system of 8 nodes in $T_0(S)$. The clusters are: (a) $c_{0,3,1}$: nodes {1, 3, 5, 7}, (b) $c_{0,3,2}$: nodes {2, 3, 6, 7} and (c) $c_{0,3,4}$: nodes {4, 5, 6, 7}.

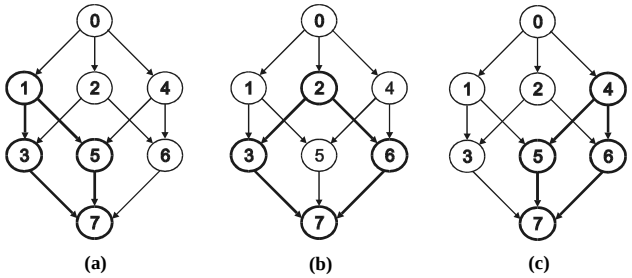

Figure 3: Cluster division for a system of 8 nodes in $T_0(S)$.

In comparison with previously published diagnostic models, the new model has a key difference: there is no more an assumption that specifies that the comparison of the outputs produced by two faulty nodes never results in a match. In order to implement the removal of this assumption in *Hi-Dif*, every node considers itself as fault-free. Furthermore, the testing strategy consists of sending tasks in pairs, and the tester always sends the task to itself and to another node, and compares the two task outputs.

If the comparison of these tasks outputs from node *i* and from node *j* produces a match, and node *i* is the tester it will obtain diagnostic information from node *j*.

Because of this reason, the algorithm has a new assumption about the external observer that is the *user* of the algorithm, and is who computes the diagnosis of the system. The observer is able to determine correctly if a given node is fault-free, so that it obtains diagnostic information from that node. In this way, this observer will never choose a faulty node from which it will obtain diagnostic information.

Besides determining if nodes have crashed or not, the algorithm also classifies all nodes that have returned task outputs in sets. These sets are constructed in a way that allows the easy identification of which nodes are returning the same output for the task. One application of this feature is to find corrupted nodes, i.e. nodes that are up and running but have had their contents modified.

In order to achieve the afore mentioned classification, the algorithm classifies nodes in sets with a sequential numeric identifier, as follows: set 0 contains nodes that have crashed; set 1 contains nodes with the correct data, i.e. nodes that are returning the same task output that the tester does; and the other sets that have identifiers greater than 1, contain nodes returning a different task outputs. If any two nodes return task outputs that are equal to each other but are different to the output of the tester, than those two nodes are classified in a set that is neither set 0 nor set 1. The algorithm is fully specified below.

4.2 Specification of the Algorithm

Algorithm *Hi-Dif* initially groups nodes in two sets: the set with faulty nodes *F*, and the set with fault-free nodes *FF*. These two sets are disjoint and the union of the two sets is equal to *V*, i.e. $F \cap FF = \varnothing$ e $F \cup FF = V$. The algorithm also employs a list of sets, called *result-set-list*,

to classify nodes in according to the tasks outputs. Each node keeps these two sets *F* and *FF* and also keeps the list *result-set-list*. By the end of a testing round every node must be in exactly one of the sets *F* of *FF*, and must be in exactly one set of the list *result-set-list*.

When node *i* compares the task output produced by itself with the output returned by another node *p* and this comparison produces a match, node *i* identifies node *p* as fault-free. Then, node *i* puts node *p* in set *FF*, removing it from set *F*, if it is the case.

The tester then puts node *p* in set 1 of *result-set-list*, i.e. puts node *p* in the set of nodes with the correct replicated data. When node *i* identifies a fault-free node, node *i* obtains diagnostic information from this fault-free node. Furthermore, as information is timestamped [19], node *i* must test if the received information is newer than its own information. The timestamps are implemented as event counters that every node keeps for all system nodes. This strategy allows the determination of the order in which events occurred and guarantees that node *i* will always keep and update only the most recent information about the state of any other node. If the received information is not new, node *i* simply rules the received information out.

When node *i* sends a task to itself and to node *p*, but node *p* doesn't reply, node *i* identifies node *p* as faulty, and puts node *p* in *F*, removing it from set *FF* if this is the case. Furthermore, node *i* puts node *p* in set 0 of *result-set-list*, i.e. puts node *p* in the set of crashed nodes of *result-set-list*.

When node *i* compares the tasks outputs produced by itself and by another node *p*, and this comparison produces a mismatch, node *i* identifies node *p* as faulty. Then, node *i* puts node *p* in the set *F* removing it from set *FF*, if this is the case. Furthermore, node *i* searches for a set in *result-set-list* in which all nodes have returned the same output returned by node *p*. If there is no such set, node *i* creates a new set in *result-set-list* and puts node *p* in this set.

Hi-Dif employs a hierarchical testing strategy, in which node *i* initially tests all it owns sons in $T_i(S)$, by sending a task always to pairs of nodes: to itself and the next son. Thus by comparing tasks results node *i* can determine the state of all nodes. When node *i* finishes testing its son's and there are still nodes about which no information about their state was obtained in the present testing round, node *i* starts to test these nodes, and obtains information from the ones that are fault-free.

Initially, node *i* tests nodes with diagnostic distance 2, i.e. the sons of its sons; then, it tests nodes with diagnostic distance 3, and so on. Figure 4(a) shows an example of a system with 8 nodes where nodes 2 and 4 are faulty. In this example, after node 0 tests all his sons, it receives information about nodes 3, 5 and 7, but no information about node 6. Node 0 then performs one more test sending a pair of tasks to itself and to node 6. In figure 4(b) node 0 tests its sons, i.e. tests nodes 1, 2 and 4 and find out that all are faulty and can't obtain any information about any other node. Then, initially node 0 tests node 3. In the sequence, it tests nodes 5 and finally it tests node 6.

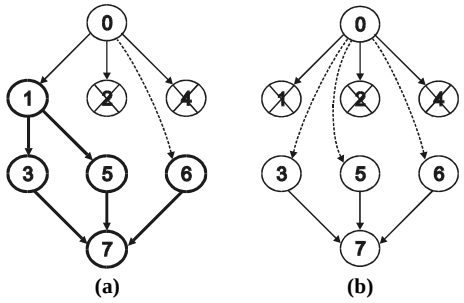

Figure 4: (a) System of 8 nodes in which 2 and 4 are faulty. (b): System of 8 nodes in which 1, 2 and 4 are faulty.

So, at the end of each testing round, every fault-free node *i* has put all other nodes in either set *F* or *FF*, i.e. $F \cup FF = V$. Furthermore every node is also in one of the sets of *result-set-list*. A node in set 0 of *result-set-list*, is necessarily also in set *F*. A node in set 1 of *result-set-list*, is necessarily also in the *FF*. Each one node in the other sets of *result-set-list*, are necessarily also in set *F*. The algorithm in pseudo-code is given below.

```
Algorithm Hi-Dif running at node i:
F = EMPTY; FF = EMPTY;
REPEAT FOREVER
   Initialize( result-set-list );
   TO_TEST = {ALL NODES};
   REPEAT
     p = next_node_to_test in TO_TEST;
     send_task( i, p );
     TO_TEST = TO_TEST – {p};
     IF ( result(p) == ∅ ) THEN
         FF = FF – {p};      F = F + {p};
         result-set-list[set 0] = result-set-list[set 0] + {p};
     ELSE IF ( result(i) == result(p) ) THEN
             F = F – {p};    FF = FF + {p};
             result-set-list[set 1] =
                         result-set-list[set 1] + {p};
             OBTAIN diagnosis information of p's cluster;
                 COMPARE timestamps of cluster's nodes;
                 UPDATE local information if necessary;
                 TO_TEST = TO_TEST – {nodes with info updated};
     ELSE IF( result(i) != result(p) ) THEN
         FF = FF – {p};      F = F + {p};    id_set = NULL;
         FOR (x starting in 2;
                 x <= result-set-list.id_biggest_set
                 AND id_set == NULL ) DO
             IF ( result(a node from result-set-list[set x])
                 == result(p) ) THEN
                 id_set = x;
             END_IF
         END_FOR
         IF ( id_set != NULL ) THEN
             result-set-list[set id_set] =
                 result-set-list[set id_set] + {p};
         ELSE
             id_new_set = result-set-list.create_new_set;
             result-set-list[set id_new_set] = {p};
         END_IF
     END_IF
   UNTIL ( the state of all nodes is obtained )
END_REPEAT_FOREVER
```

4.3 Proofs

In this section we present the formal proofs for the latency, the maximum number of tests and the diagnosability of the algorithm.

Theorem 1: All fault-free nodes running the algorithm *Hi-Dif* require, at most, log_2N testing rounds to achieve the complete diagnosis of the system.

Proof: Consider a new event at node *a*. By the definition of testing round, all nodes that are sons of node *a*, diagnose the event in the first testing round after it occurred. Considering graph $T_a(S)$, shown in figure 5, in the first testing round after the event, all sons of node *a* diagnose the event.

In the second testing round, all nodes with diagnostic distance equal to 2 diagnose the event, either by getting diagnostic information from nodes with diagnostic distance equal to 1 to node *a*, or by directly testing node *a*, if all nodes with diagnostic distance equal to 1 to node *a* are faulty. In $T_a(S)$ illustrated in figure 5, the nodes that are sons of *a* diagnose the event either by getting information from other sons of *a* or by directly testing node *a*.

Assume that fault-free node *i* with diagnostic distance *d* to node *a* diagnose the event at node *a* in at most *d* testing rounds.

Now consider node *j* with diagnostic distance *d*+1 to node *a*. By the definition of diagnostic distance, any node with diagnostic distance *d*+1 to node *a* is a son of a node with diagnostic distance *d* to node *a*. So node *j* is son of some node *i*. By the definition of testing round, a node must test all its sons in each testing round, so node *j* tests node *i* in all testing rounds, then node *j* can take at most one testing round to get new information from node *i*.

As node *i* diagnoses node *a*'s event in at most *d* testing rounds, and node *j* takes at most one testing round to get new diagnostic information from node *i*, node *j* can take at most *d*+1 testing rounds to diagnose the node *a*'s event.

Therefore, if node *j* has diagnostic distance *d*+1 to node *a*, *j* diagnoses an event that happened at node *a*, in at most *d*+1 testing rounds. Thus, if the diagnostic distance between two nodes is *x* one of these nodes may take up to *x* testing rounds to diagnose an event at the other node.

Thus, the maximum latency occurs when nodes with the largest distance in the system obtain information about each other. According to the definition of the hypercube's [16] the largest diagnostic distance between two nodes is log_2N. Therefore the algorithm's maximum latency is log_2N testing rounds. □

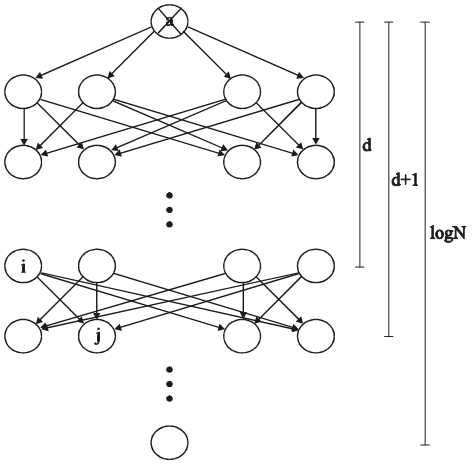

Figure 5: Graph $T_a(S)$.

Theorem 2: The maximum number of tests required by all fault-free nodes running in one testing round of *Hi-Dif* is $O(N^2)$.

Proof: Considering a system with *N* nodes, the worst case of the number of tests required in a testing round is the sum of the worst case of the number of tests for each one of the *N* nodes. In the worst case, the number of tests required by one node is when this node needs to test all of other *N*–1 nodes in the system; one example is the situation where *N*–1 nodes are faulty. In this case the fault-free node executes *N*–1 tests.

Considering this case, in which there is only one fault-free node *i* and *N*–1 faulty nodes, and also considering that none of them have crashed, and none of them produces the same output to a given task, the fault-free node sends tests to itself and each of the other faulty nodes. However each of the faulty nodes assumes itself to be fault-free, thus each executes *N*–1 tests. In conclusion, the number of tests for all nodes is $N * (N–1) = N^2–N$, that is $O(N^2)$. □

Theorem 3: A system running *Hi-Dif* is (*N*–1)-diagnosable.

Proof: Initially, consider a system with only one fault-free node and *N*–1 faulty nodes. By definition, the fault-free node tests all nodes, sending tests in pairs to itself and each other node, identifying the state of all nodes as faulty.

Now, consider a system with more than one fault-free node. Each of these fault-free nodes executes tests until it finds another fault-free node. When the tester finds a fault-free node, it obtains diagnostic information from this tested fault-free node. By getting diagnostic information from the tested fault-free node and, considering the information obtained by its own tests, the tester achieves the complete and correct diagnosis of the system.

However, if a situation such as shown in figure 6 happens, i.e. if node *a* could obtain diagnostic information about node *c* from node *b* and at the same time node *b* obtains diagnostic information about node *c* from node *a*, then both, node *a* and node *b*, would not achieve the complete diagnosis of the system.

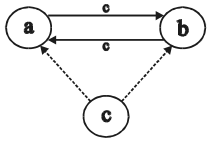

Figure 6: Nodes *a* and *b* exchange information about node *c*.

This situation never happens because if node *a* receives information about node *c* from node *b*, the diagnostic distance between nodes *a* and *c* must be larger than the diagnostic distance between nodes *b* and *c*; analogously for node *b* to receive diagnostic information about node *c* from node *a*, the diagnostic distance between nodes *b* and *c* must be larger than the distance between nodes *a* and *c*.

In conclusion, even if there is only one fault-free node, this node is capable of correctly achieving the complete diagnosis of the system, so the algorithm is (N–1)-diagnosable. □

5. Simulation Results

In this section 1,200 experiments obtained by simulating *Hi-Dif* in systems with 128 nodes are reported. Initially 600 experiments were executed in which up to 32 nodes – randomly chosen – can be faulty. After that, other 600 experiments were executed in which up to 64 nodes – also randomly chosen – can be faulty. In all experiments the fault type (crash or data modification) was randomly chosen.

The first 600 experiments considered the probability of, at most, 32 nodes – randomly chosen – becoming faulty. All nodes were initially fault-free. In order to determine which nodes were be faulty, 200 experiments were performed with a fault probability of 30%; 200 experiments were performed with a probability of 60%, and the remaining 200 experiments considered a probability of 90%. The other 600 experiments were analogously performed but considering the probability of, at most, 64 nodes – also randomly chosen – becoming faulty.

The latency was measured for all fault-free nodes to identify all faulty nodes. The number of tests was counted from the instant the faults occurred until the moment where all fault-free nodes identified the change of the status of all nodes that became faulty.

Figure 7 shows the average latency for each of the 3 groups of 200 experiments for both probabilities of faulty nodes. With up to 32 faulty nodes, it is possible to notice that with a fault probability equal to 90%, the average latency, i.e. the number of testing rounds for all nodes to complete the diagnosis, was 3.94 rounds. With up to 64 faulty nodes and fault probability equal to 90%, the average latency was 4.25 testing rounds.

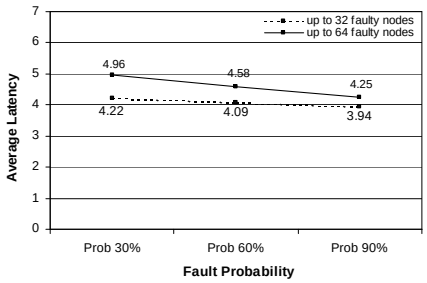

Figure 7: Average latency in a system in which up to 32 of 128 nodes are faulty.

Figure 8 shows the average number of tests performed in each of the 200 experiments also for both probabilities of faulty nodes. With up to 32 faulty nodes, it is possible to notice that with a fault probability equal to 90%, the average of number of tests realized was 2,100 tests. With up to 64 faulty nodes and fault probability equal to 90% the average was equal to 2,264 tests.

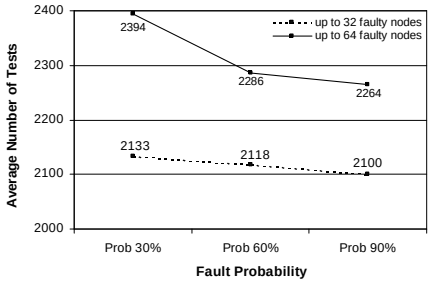

Figure 8: Average number of tests in a system in which up to 32 of 128 nodes are faulty.

Comparing the two experiments, respectively with up to 32 and 64 faulty nodes, the average latency stayed between 4 and 5, but the average number of tests had a steeper increase. This follows from the fact that the larger the number of faulty nodes, more tests are executed by the fault-free nodes.

6. Checking the Integrity of Web Clusters

Experiments were also performed with a practical tool that implements the algorithm to check the integrity of nodes with replicated data across a Web cluster. The tool was implemented with two components: a server and a client. The server is the component responsible of replying to tests and sending diagnostic information to other system nodes. The client implements *Hi-Dif*. This client is responsible for setting testing intervals, running the tests, asking for diagnostic information and completing diagnosis. These two components execute as individual processes in the system, i.e. the tool isn't executed within the Web server.

Figure 9 shows a group of 4 Web servers with replicated data executing the implemented tool. In this example, server 1 is faulty, server 2 has a modification in his data that is supposed to be replicated, and servers 3 and 4 are fault-free. The tool that executes in all servers, tests all of other configured servers to check the integrity of the replicated data among all servers.

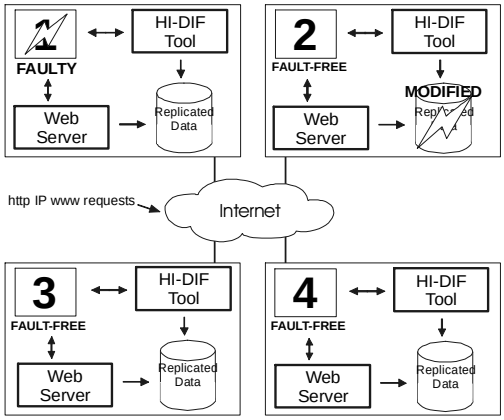

Figure 9: A Web server cluster executing the implemented tool.

One experiment was performed in a system with 32 nodes in which 8 nodes became faulty by having their data modified, simultaneously. The testing round was 10 seconds. The graph in figure 10 shows the number of nodes and the moment that all 24 fault-free nodes identified the 8 faulty nodes.

In this graph, all nodes that identified all faults in the first 10 seconds, appear in the first column, and so on. It is possible to notice that in the first 10 seconds after the 8 faults happened, 2 nodes identified all 8 faults. It is also possible to notice that most of the fault-free nodes identified all faults in between 20 and 30 seconds. And after at most 50 seconds all fault-free nodes had identified all faulty nodes.

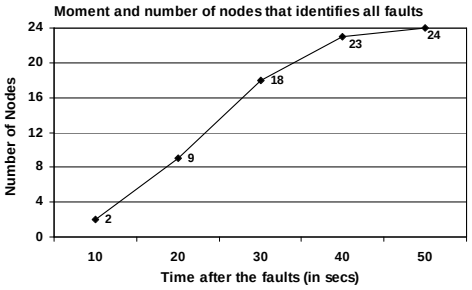

Figure 10: Experiment run in a system with 32 nodes in which 8 nodes become faulty.

7. Conclusion

This work introduced a new general model for Hierarchical Adaptive Distributed Comparison-Based System-Level Diagnosis, and also presented a new diagnosis algorithm, *Hi-Dif*, which is based on the presented model. *Hi-Dif* allows the distributed detection of integrity violations in replicated data, available for instance through Web servers. A key difference of this model to previously published ones is that the comparison, performed by a fault-free node, over outputs of faulty units can match.

A formal proof is presented showing that *Hi-Dif* has a worst-case latency of $log_2 N$ testing rounds for a system of N units. The algorithm is also $(N–1)$-diagnosable and the maximum number of tests required is $O(N^2)$. Simulation experiments of the algorithm and experiments performed with a tool that implements the algorithm applied to Web clusters were also presented. Experimental results confirm the maximum number of tests and the latency of the algorithm.

Future work includes applying the algorithm for *peer-to-peer* networks.

References

[1] CERT Coordination Center, http://www.cert.org, Accessed on 05/09/2004.

[2] ALDAS, Analytisches Labor Dr. Axel Schumann, http://www.aldas.de, Accessed on 05/10/2003.

[3] A. Subbiah, and D. M. Blough, "Distributed Diagnosis in Dynamic Fault Environments," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 15, No. 5, pp. 453-467, May 2004.

[4] L. C. P. Albini, and E. P. Duarte Jr., "Generalized Distributed Comparison-Based System-Level Diagnosis," *2nd IEEE Latin American Test Workshop*, pp. 285-290, Sep. 2001.

[5] D. Ingham, S. K. Shrivastava, and F. Panzieri, "Constructing Dependable Web Services," *IEEE Internet Computing*, Vol. 4, No. 1, pp 25-33, Jan/Feb 2000.

[6] B. Tan, S. Foo, and S. C. Hui, "Monitoring Web Information Using PBD Technique," *Proc. 2nd Intl. Conference on Internet Computing (IC'2001)*, Las Vegas, USA, pp. 666-672, Jun. 2001.

[7] Url Minder, http://www.netmind.com/URL-minder/URL-minder.html. Accessed on 22/09/2003.

[8] B. Lu, S. C. Hui, and Y. Zhang, "Personalized Information Monitoring Over the Web," *1st Intl. Conference on Information Technology & Applications (ICITA 2002)*, Nov. 2002.

[9] V. Boyapati, K. Chevrier, A. Finkel, N. Glance, T. Pierce, R. Stockton, and C. Whitmer, "ChangeDetector™: A Site-Level Monitoring Tool for the WWW," *Intl. World Wide Web Conference*, USA, pp. 570-579, May 2002.

[10] S.-J. Lim, and Y.-K. Ng, "An Automated Change-detection Algorithm for HTML documents Based on Semantic Hierachies," *Proceedings of the 17th Intl. Conference on Data Engineering (ICDE'01)*, Heidelberg, Germany, pp. 303-312, Apr. 2001.

[11] M. Malek, "A Comparison Connection Assignment for Diagnosis of Multiprocessor Systems," *Proc. 7th Intl. Symp. Computer Architecture*, pp. 31-36, 1980.

[12] K. Y. Chwa, and 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.

[13] J. Maeng, and M. Malek, "A Comparison Connection Assignment for Self-Diagnosis of Multiprocessor Systems," *Digest 11th Intl. Symp. Fault Tolerant Computing*, pp. 173-175, 1981.

[14] A. Sengupta, and A. T. Dahbura, "On Self-Diagnosable Multiprocessor Systems: Diagnosis by Comparison Approach," *IEEE Trans. on Computers*, Vol. 41, No. 11, pp. 1386-1396, 1992.

[15] D. M. Blough, and H. W. Brown, "The Broadcast Comparison Model for On-Line Fault Diagnosis in Multicomputer Systems: Theory and Implementation," *IEEE Trans. on Computers*, Vol. 48, pp. 470-493, 1999.

[16] D. Wang, "Diagnosability of Hipercubes and Enhanced Hypercubes under the Comparison Diagnosis Model," *IEEE Trans. on Computers*, Vol. 48, No. 12, pp. 1369-1374, 1999.

[17] T. Araki, and Y. Shibata, "Diagnosability of Butterfly Networks under the Comparison Approach," *IEICE Trans. Fundamentals*, Vol. E85-A, No. 5, May 2002.

[18] J. Fan, "Diagnosability of Crossed Cubes," *IEEE Trans. on Computers*, Vol. 13, No. 10, pp. 1099-1104, Oct. 2002.

[19] E. P. Duarte Jr., A. Brawerman, and L. C. P. Albini, "An Algorithm for Distributed Hierarquical Diagnosis of Dynamic Fault and Repair Events," *Proc. IEEE Intl. Conference on Parallel and Distributed Systems 2000*, pp. 299-306, 2000.