SYSTEM LEVEL FAULT DIAGNOSIS IN DISTRIBUTED SYSTEM

[1]Manmath Narayan Sahoo, [2]Pabitra Mohan Khilar

[1] NIT Rourkela/CSE, Rourkela, India, Email: sahoo.manmath@gmail.com
[2] NIT Rourkela/CSE, Rourkela, India, Email: pmkhilar@nitrkl.ac.in

ABSTRACT

With the increasing need for efficient means of automatic fault diagnosis in large distributed computing systems, system-level fault diagnosis has been a fertile research area for the last few years. The increasing complexity of the multiprocessor systems leads to different types of faults which in turn degrades the performance of the system. This paper includes different types of faults that might occur in a distributed environment. It discusses traditional mechanisms for hardware and software fault-tolerance such as – Built In Self Test (BIST), Triple Modular Redundancy (TMR), N-Version Programming, Recovery Block Scheme and Check-pointing and Rollback Recovery. BIST, TMR, N-Version Programming are basically redundancy techniques. Along with these, the comparison methods such as PMC model, Simple Comparison Model and the General Comparison Model for diagnosis of multiprocessor systems are also discussed.

INDEX TERMS : Fault Tolerance, Diagnosability, System Level Diagnosis, Comparison Models, Multiprocessor System.

1. INTRODUCTION

It has long been the goal of system designers to connect independent computer resources together to create a network with greater power and availability than any of its parts. Unfortunately the reverse can happen if faulty resources are allowed to corrupt the network. In the distributed environment, there are many opportunities for failure. Any component in any computing node could fail. A hardware or software failure on a management node could affect the entire system, as scheduling and synchronization data is lost. Failures external to individual nodes are also possible. Many possible failures could remove a large number of nodes from the system, such as a network switch failure. Any of these failures will cause the applications running on the affected nodes to crash or produce incorrect results. So the motivation towards the fault-tolerant computing is to form an agreement among the fault-free members of the resource population on a quantum of information in order to maintain the performance and the integrity of the system. System reconfiguration means after any failure of a node the whole system will not collapse rather the rest of the fault-free nodes will function correctly but in a gracefully degraded manner.

The most widely accepted definition of a fault-tolerant computing system [9] is that, it is a system, which has the built-in capability to preserve the continued correct execution of its programs and I/O functions in the presence of a certain set of operational faults.

There are two fundamental classes of faults that can occur in cluster systems [2]. First, a centralized component such as storage node or management software can fail as a result of a software bug or a hardware fault. Protecting against these failures typically involves redundancy. Critical functionality is replicated over several nodes such that if one node fails, a backup can step in to take over the responsibilities of the primary. The second class of failure is a crash or hang of software on one of the many computation nodes in the cluster. This can result from a software bug in an application, a hardware fault on the node, or a problem in the operating system local to the node. All of these failures have the same end result: the application running on the node can no longer function properly, but the other nodes participating in the same computation can continue unaffected excepting that they will no longer receive output from the failed node. The standard technique for handling application failures is to periodically checkpoint the computational state of the application such that it can be restored in the event of a system failure. Other nodes participating in the computation may need to be rolled back to earlier checkpoints in order to make them consistent with the recovery state of the failed node.

Diagnosing each unit of the distributed system to check whether a particular node is faulty or fault-free is called as system level diagnosis [3]. In order to perform diagnosis, a system is usually partitioned into a set of well-defined units and one or more tests are applied to each unit. A test may be any kind of a pass/fail check on the operation of a unit. So a fault diagnosis algorithm deals with the problem of accurately locating faults in a system, given a system description and a set of test results.

The reminder of the paper is organized as follows: section 2 presents the system model of a multiprocessor system. Section 3 outlines different types of faults that

could occur in a distributed environment. In section 4 presents the traditional methods for tolerating hardware and software faults. Section 5 discusses different system level fault diagnosis techniques and Section 6 concludes our research work.

2. SYSTEM MODEL

A multiprocessor system is modeled by a graph G(V,E,C) where V is a set of n vertices and E and C are sets of edges [4]. Each element of the set V corresponds to a single computer, processor or unit in the system and each element of the set E represents the communication connection between a pair of units (processors, computers). Set C corresponds to the comparison connections and it may or may not be a subset of E. The comparison connection (comparison edge), represented by an edge, denotes that it is possible to compare the behavior of two units, which are the endpoints of the considered edge. It also means that there is some hardware and/or software mechanism making the comparison feasible.

3. FAULT MODEL

There are countless ways in which computing systems and applications may fail. These failures can be categorized by abstract models that describe how a system will behave in the presence of faults [6]. A fault tolerance technique will assume a certain model of failure when making claims about the types of faults it can handle.

3.1 **Fail-Stop Fault:** The fault that occurs when a processor ceases operation and alerts other processors of this fault [Schlichting and Schneider 1983].

3.2 **Crash Fault:** The fault that occurs when a processor loses its internal state or halts. For example, a Processing Element that has had the contents of its instruction pipeline corrupted or has lost all power has suffered a crash fault.

3.3 **Omission Fault:** The fault that occurs when a processor fails to meet a deadling or begin a task [Cristian et al.1986]. In particular, a *send omission* fault occurs when a processor fails to send a required message on time or at all, and a *receive omission* fault occurs when a processor fails to receive a required message and behaves

3.4 **Timing Fault:** The fault that occurs when a processor completes a task either before or after its specified time frame or never [Cristian et al.1986]. This is sometimes called performance fault.

3.5 **Incorrect Computation Fault:** The fault that occurs when a processor fails to produce the correct result in response to the correct inputs [Laranjeira et al.1991].

3.6 **Authenticated Byzantine Fault:** An arbitrary or malicious fault, such as when one processor sends differing messages during a broadcast to its neighbors that cannot imperceptibly alter an authenticated message [Lamport et al.1982].

3.7 **Byzantine Fault:** Every fault possible in the system model [Lamport et al.1982]. This fault class can be considered the universal fault set.

4. TRADITIONAL FAULT TOLERANCE MECHANISMS

Failures are the manifestation of the errors latent in the system. Therefore even in the situations where errors are present the system should be able to tolerate the faults and compute the correct results. This is called fault-tolerance [10]. Fault-tolerance can be achieved by carefully incorporating redundancy.

There are 2 types of fault-tolerance mechanisms [10].

i. Hardware fault-tolerance.
ii. Software fault-tolerance.

4.1 Hardware fault-tolerance

The following two methods are popularly used to achieve hardware fault-tolerance.

4.1.1 Built In Self Test (BIST): In BIST, the system periodically performs self-tests of its components. Upon detection of a failure, the system automatically reconfigures itself by switching out the faulty components and switching in one of the redundant good components.

4.1.2 Triple Modular Redundancy (TMR): In TMR [6], as the name suggests, three redundant copies of all critical components are made to run concurrently. The system performs voting of the results produced by the redundant components to select the majority result. TMR can help tolerate occurrence of only a single failure at any time. So at least (2n+1) redundant components are required to tolerate simultaneous failures of n components.

4.2 Software fault-tolerance

The following methods [10] are popularly used to achieve software fault-tolerance

i. N-version programming
ii. Recovery block technique
iii. Check-pointing and Rollback recovery.

4.2.1 N-Version programming:

In this technique, independent teams develop N different versions of a software component. The redundant

modules are run concurrently (possibly on redundant hardware). The results produced by the different versions of the module are subjected to voting at run time and the result on which majority of the components agree is accepted.

4.2.2 Recovery Block Scheme:

In this scheme, the redundant components are called *try blocks*. Each try block computes the same end result as the others but are written using different algorithms. In N-version programming different versions of a component are written by different teams of programmers, whereas in recovery block different algorithms are used in different try blocks. Also in contrast to N-version programming approach where the redundant copies are run concurrently, in the recovery block approach they are run one after another. The results produced by a try block are subjected to an *acceptance test*. If the test fails, then the next block is tried. This is repeated in a sequence until the result produced by a try block successfully passes the acceptance test.

3.2.3 Check-pointing and Rollback recovery

In this technique [2] as the computation proceeds, the system state is tested each time after some meaningful progress in computation is made. Immediately after a state-check test succeeds, the state of the system is backed up on a stable storage. In case the next test does not succeed, the system can be made to rollback to the last check-pointed state. After a rollback, from a check-pointed state a fresh computation can be initiated.

5. SYSTEM LEVEL FAULT DIAGNOSIS TECHNIQUES

5.1 The PMC Model

In 1967, Preparata, Metze, and Chien (PMC) formed the framework for system level fault diagnosis [6], in which a processing element (PE) can test other. However the test result is not reliable if the testing PE is faulty.

The PMC model uses a graph G (V, E) to model the system's testing convention. Where V represents the set of PEs, and directed edges in E represent one processor applying a test to another processor. Each edge is labeled with a O(1) if the corresponding test produces a passing (failing) result. The set of results is known as a *syndrome* [5].

Table I: Invalidation Rules for PMC Model

Status of Tester PE	Status of Tested PE	Test Result
Fault-Free	Fault-Free	0
Fault-Free	Faulty	1
Faulty	Fault-Free	X
Faulty	Faulty	X

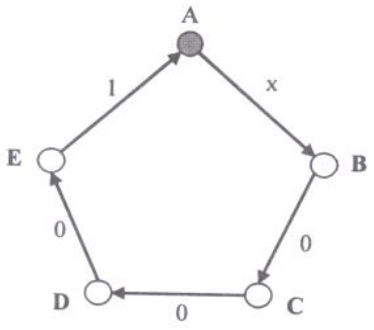

Figure 1: Example (PMC Model)

Table I depicts the invalidation rules for the PMC model and figure 1 shows an example of system level diagnosis using the PMC model. A through E are processors where an edge (v1, v2) represents a test by v1 on v2. A label on an edge is the result of that test. The table in the figure relates the status of v1 and v2 to the result of the test of v1 on v2. In the figure, A is faulty as denoted by its gray color. The "X" on the edge (A, B) means that this result may be a one or a zero without affecting the result of the diagnosis. If it is assumed that it is a 1-diagnosable system, then it is possible to identify A as being faulty. First notice that edge (E, A) is labeled "1" meaning A is faulty if E is fault free. If E were faulty, then it would be the single faulty member of the system. So diagnosis depends on deducing the condition of E. Either E is faulty, or A is. Assume E is faulty in which case D must be fault free since 1-diagnosable system. But this leads to a contradiction because the label "0" on edge (D, E) implies that fault-free D misdiagnosed faulty E. Thus, E is fault free, and A is faulty regardless of the actual value of "X."

5.2 The Simple Comparison Model (SCM)

In the simple comparison model [1] there is a central observer (comparator), which performs comparisons between pairs of processors by assigning them some tasks from the set of tasks T = {T1, T2, ...}. Each pair of processors v_i and v_j is assigned a task T_l e T. Once both processors complete the task T_l, their results are compared.

The comparison graph in this case, is an undirected graph G = (V,C), where V denotes the set of processors and C={(vi, vj) : (vi, vj) is a pair of processors performing the same task T_l e T }.The processor pair (vi, vj) or (vj, vi) are denoted as c_{ij}.

Let F be the fault set i.e. the set of faulty processor, and W^{ij} is the comparison outcome of the processor pair c_{ij}. The set of all comparison outcomes is called as *comparison syndrome* [7]. According to SCM a comparison syndrome W, is said to be consistent or compatible [1] with a fault set F, if for any c_{ij} e C, such that v_i is fault- free, i.e. v_i e V-F, W^{ij} = 1 iff v_j e F.

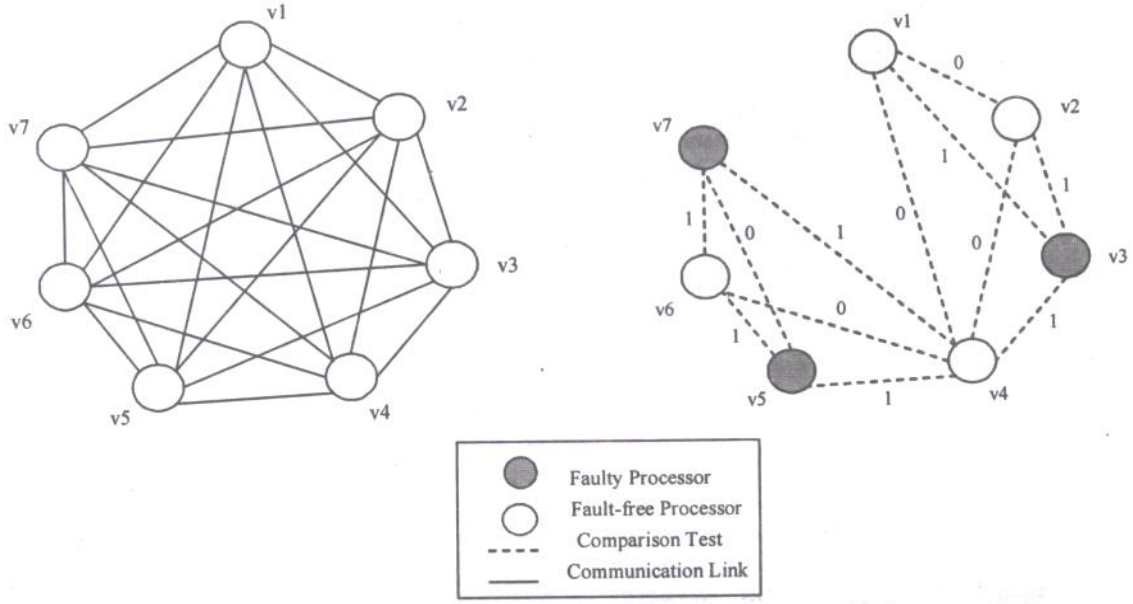

Figure-2: (a) communication graph (b) comparison assignment

Figure 2(a) shows a small system of seven fully connected processor and figure 2(b) shows a comparison assignment with dashed lines under the fault set F= $\{v_3, v_5, v_7\}$

5.3 The General Comparison Model (GCM)

Unlike SCM, in case of GCM [8], the comparator node is one of the processors under comparison. Figure 4 depicts the invalidation rules for GCM. According to GCM, if the comparator node is fault-free, then the comparison outcome is 0 if none of the compared nodes is faulty, and it is 1 if one of them is faulty. However, if the comparator itself is faulty, then the comparison outcome is unreliable, and hence, may be 0 or 1.

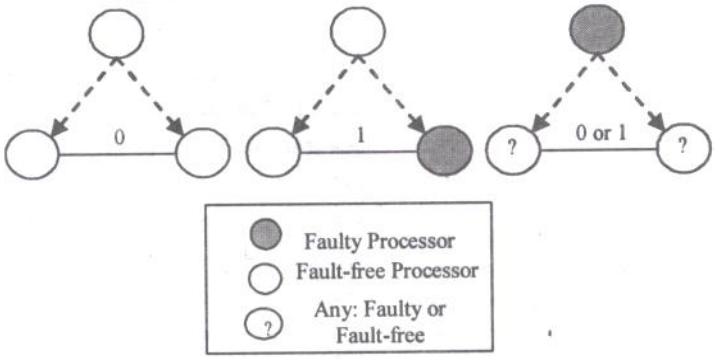

Figure 3: Invalidation rules for GCM

In this model, the communication graph and the comparison graph are defined as follows. The communication graph is represented by the graph G(V,E), where V = {v1, v2, ···, vN} denotes the set of N processing units and E refers to the set of communication links. The comparison graph is a multigraph whose edges represent comparison tests performed by the processing units on pairs of processors.

Figure 4(a) shows a five units interconnection graph and Figure 4(b) depicts an example of a comparison graph.

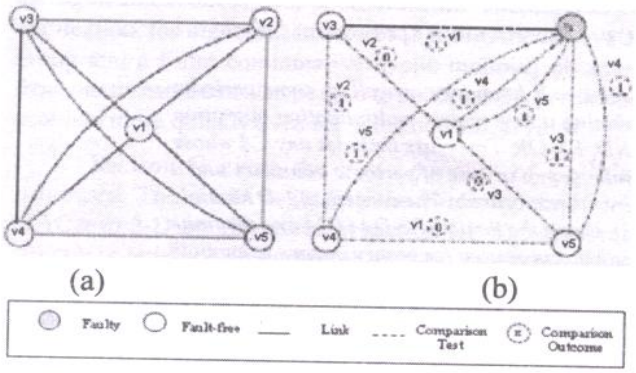

Figure 4: (a) An interconnection graph (b) A comparison multigraph

Each processing unit is assigned a subset of the other units to test. Testing is based on assigning a set of tasks T = {T1, T2, ···} to the system's processors. A pair of processors (v_i, v_j) is assigned a task T_i e T. Once both processing units execute the task T_i, the results are compared. The comparison multigraph is represented by an undirected graph M(V,C), where V and C denote, respectively, the vertices (processors) and the edges (comparison tests). For every v_i, v_j, v_k e V, (vi, vj, vk) e C, or simply c_k^{ij} e C, iff processor v_k tests processors v_i and v_j by assigning them the same task. Each edge c has a label associated with it. The binary value assigned to the label depends on the GCM's invalidation rules as discussed previously.

6. CONCLUSION

Distributed management hardware and software fault-tolerance typically makes use of redundancy, due to the relatively small number of components that need to be duplicated for this approach. When a component fails, the redundant components take over the responsibilities of the failed parts. Apart from this some other techniques such as check-pointing and rollback recovery, N-version programming and Recovery block scheme are also used. A comparison method can be used for fast and reliable detection and/or location of a faulty unit in multiprocessor systems. This method is primarily applicable for the re-configurable multiprocessor systems where the simultaneous running of the same programs on different processors and diagnostic comparisons are easily feasible. Simplicity and ease of implementation as well as the smaller number of tests than those used in classical methods make comparison connection assignment an attractive alternative in the diagnosis of the multiprocessor systems.

Reference

[1] *Mourad Elhadef, Shantanu Das, and Amiya Nayak, "System-Level Fault Diagnosis Using Comparison Models: An Artificial-Immune-Systems-Based Approach," JOURNAL OF NETWORKS, VOL. 1, NO. 5, pp. 43-53, Sep/Oct 2006*

[2] *Michael Treaster , "A Survey of Fault-Tolerance and Fault-Recovery Techniques in Parallel Systems," arXiv:cs.DC/0501002 v1 1 Jan 2005*

[3] *Jinghao Xu and Leszek Lilien, **"A survey of methods for system-level fault diagnosis,"** ACM-IEEE Computer Society Fall Jolnt Computer Conference, DallaS, Texas, October 1987*

[4] *M. Malek, "A comparison connection assignment for diagnosis of multiprocessor systems," in Proc. 7th Symp. Comput. Architecture, May 1980. pp 31-35*

[5] *F. P. Preparata, G. Metze and R. T. Chien, "On the connection assignment problem of diagnosable systems," IEEE Trans. Electron. Comput., vol. 16, pp. 848-854, Dec. 1967*

[6] *M. Barborak, M. Malek, and A. Dahbura, "The Consensus Problem in Fault-Tolerant Computing," ACM Computing Surveys, vol. 25, no. 2, pp. 171–220, June 1993.*

[7] *J. Maeng and M. Malek, "A Comparison Connection Assignment for Self-Diagnosis of Multiprocessor Systems," in Proc. 11th Int. Symp. on Fault-Tolerant Comput., 1981, pp. 173–175.*

[8] *A.Sengupta and A. Dahbura, "On Self-Diagnosable Multiprocessor Systems: Diagnosis by the Comparison Approach," IEEE Trans. on Computers, vol. 41, no. 11, pp. 1386–1395, Nov. 1992.*

[9] *Algirdas Avizienis," Fault-Tolerant Systems," IEEE Trans. on Computers Vol. C-25, NO. 12,pp. 1304-1312, December 1976*

[10] *Rajib Mall,"Real -Time System, Theory and Practice," Pearson Education, 2007*