Journal of Computer Science 3 (2): 107-112, 2007
ISSN 1549-3636

Performance Improvement and Deadlock Prevention for a Distributed Fault Diagnosis Algorithm

Thabit Sultan Mohammed
Software Engineering Department, Faculty of Science and IT, Al-Zaytoonah University
P.O. Box-130, Amman (11733)–Jordan

Abstract: This research presents an overview to the issue of fault diagnosis in distributed systems and an evaluation study to some of the algorithms proposed in literature for performing distributed fault diagnosis. One algorithm was chosen and adopted for implementation in a simulator for investigation. A strategy for improving the performance of this algorithm and preventing deadlock was proposed in this research. A measure of the improvement in performance was also presented.

Key words: Distributed systems, fault diagnosis, fault-tolerance, diagnosis algorithms, deadlock, deadlock prevention

INTRODUCTION

The general increase in the use of computing has led to demands for more sophisticated facilities in terms of speed, reliability, availability, etc... Such demands are often supported by a general desire to decentralize. Fault tolerance and reliability are among the design issues that steadily gaining in importance as distributed systems are become progressively commercialized. The implementation of fault tolerance is vital in a number of applications; such as safety critical applications, highly available systems and applications in relatively inaccessible areas. Fault tolerance, refers to the ability of computers to withstand failures of some of their elements and continue to operate correctly. It includes a number of basic steps[1, 2]; fault detection, fault location (diagnosis and identifying the faulty elements) and repair and/or system reconfiguration.

The theory of fault diagnosis in distributed systems has received considerable attention over the years. The Fundamental model[3], which referred to as the PMC model, assumes that the system is partitioned into units (nodes), each of which can perform tests on a subset of remaining units. The system also includes a facility for gathering test results and performing the diagnosis. Such facility is referred to as *global observer* and is not subject to fault.

Later research has concentrated on more elaborate and more general models, where it was found that additional extensions and modifications are necessary to make the PMC model applicable to actual systems[4]. Some of the proposed models have recognized that the use of global observer contradicts the principle of distributed systems and it is unrealistic to assume that such an element is capable of observing all test results without being itself subject to faults. In the PMC model, it was assumed that faults are equiprobable. A generalized model[5] took into account the probabilistic nature of fault occurrence in the nodes of the system. Dahbura and Masson[6] proposed a diagnosis algorithm for a general case of *t-fault diagnosable* systems. In[7], a diagnosis algorithm was proposed for locating faulty and fault-free nodes in system comprising a number of processors that are being allocated similar computational tasks. The algorithm is based on a comparison approach.

Another series of diagnosis algorithms were presented, which basically depend on the following definition:

Definition: A distributed system with communication graph C and testing graph T_s is said to be t-fault diagnosable for a set of t or fewer faulty nodes, if and only if, each node in the system is capable of reliably diagnosing the condition of all other nodes in the system, by means of test results being conducted through T_s and by analyzing information contained in diagnostic messages received from neighbors.

Although these algorithms differ in detail, they are all based on the ability of a node to perform tests on some of its neighboring nodes and sending results back. In[8], two algorithms, SELF and SELF2 were proposed and in[9], algorithm SELF3 was proposed. Depending on the assumptions of these algorithms (i.e. SELF, SELF2 and SELF3), Hosseini *et al.*[10] have proposed algorithm NEW-SELF. Later it was found that it is possible for a temporary misdiagnosis of some fault-free nodes as faulty, if failures in communication links occur. For this reason, therefore a modified version of algorithm SELF3, referred to as modified SELF3, was proposed[11]. Theoretical proofs are usually difficult to be given a for algorithms like these, therefore algorithms SELF2 and NEW-SELF were implemented in a simulated distributed system [12]and[13] and the

Corresponding Author: Dr. Thabit Sultan Mohammed, Al-Zaytoonah University, Faculty of Science and IT, Software Engineering Department, P.O. Box-130, Amman (11733)–Jordan, E-mail: thabitsm@yahoo.com

simulation showed a temporary misdiagnosis of some fault-free nodes due to failures in communication links.

In this research, the modified SELF3 algorithm will be adopted for further investigation, as we have noticed that it is the most mature among the proposed algorithms. For easy reference to the details of this algorithm, section 3 will present a description of these details.

Deadlock in distributed algorithms: All the above mentioned algorithms and infact any other distributed algorithm, are considered to be composed of processes, which are executed at system nodes and exchange information with each other by message passing. Once these algorithms are applied, special attention need to be focused on the problem of *deadlock.* Deadlock refers to the case in which there exists a group of waiting processes, such that no process in this group can send message (release resource) until it receives the required message (resource) from other processes in the group. When this occurs, all these will wait permanently and the progress of their execution is halted. Hence, the execution of processes can turn out to be completely useless unless proper and careful control is executed. To handle deadlocks in distributed systems, one can try to adopt approaches known from centralized systems; i.e. prevention, avoidance and detection with recovery[14,15] and[16]. The necessary and sufficient conditions for deadlock are (mutual exclusion, no preemption, hold and wait and circular wait). Deadlock prevention is based on violating these conditions[17].

Modified SELF3 algorithm took into consideration deadlock avoidance, where "interrogation messages" are designed such that they traverse the testing graph only through acyclic paths. This is assumed by appending a set of nodes referred to as *set T* , where a node that is contained in *T* should not be re-interrogated by another node, receiving a message comprising this set, about the condition of an accused node. The handling of deadlock is a complex process due to the nature of a distributed system, where no node has accurate knowledge of the system state[18]. The stability of a deadlock handling approach greatly depends on the application and environment.

This fact has become apparent when implementing the modified SELF3 algorithm for investigation with different topologies of distributed systems. Different topologies has led to different formulations of the *set T* with some of them causing a violation to the guarantee that no node will be re-interrogated about the condition of an accused node. Such re-interrogations mean replicated actions leading to extra messages and may force messages to traverse cyclic paths. The possible variants of the formulations of the *set T* and the diagnosis scenarios are also presented. A strategy for avoiding replicated actions and a measure of the extra messages saved are discussed in them.

Description of the modified SELF3 algorithm: The modified algorithm SELF3[11] assumes that every node P_i in the system has two sets *ND-FLUR*$_i$ and *LNK-FLUR*$_i$. The elements of *ND-FLUR*$_i$ are faulty nodes in the system, while the elements of *LNK-FLUR*$_i$ are faulty communication links between P_i and the nodes with which it has direct communication links. When a node P_q is assigned to test another node P_r, they are called *tester* and *testee* respectively. The algorithm employees the following forms of messages.

1. **[P_r by P_q node]**, this message is referred to as 'broadcasting message' and it means that node P_q has determined that node P_r is faulty.
2. **[P_q – P_r link]**, this is also a 'broadcasting message' and it means that the direct communication link between P_q and P_r is faulty.
3. **[*?,T,*P_q*,*P_r*,*P_q]**, this is called ,'interrogation message'. Whenever a node P_q testes a node P_r and P_r fails the test, then P_q will interrogate all its fault-free testees P_s's (i.e the nodes that have passed the test performed on them by P_q) about the condition of P_r, by sending an interrogation message, of the form shown, to P_s. As it has been mentioned in the previous section, the *set T* is used in this type of messages to ensure that they traverse the testing graph only through acyclic paths. The initial content of this set is; $T = [P_q \cup P_s \in TESTED - BY(P_q)]$.
4. **[*YES,*P_q*,* P_r*,*P_s]**, this message is for 'transmitting test result'. When node P_s receives an interrogation message, it will conduct a test on P_r if it is a tester for this node and if P_r passes the test then this message will be sent.
5. **[*NO,* P_q*,*P_r*,* P_s]**, this message is also for 'transmitting a test result'. It is to be sent back in two cases, first if node P_s, which has received an interrogation message regarding the condition of P_r is a tester and P_r fails the test by P_s, second, if P_s is not a tester of P_r and it has no fault-free testees $P_t \in TESTED-BY(P_s)$ such that $P_t \notin T$.

   Alternatively, if P_s is not a tester of P_r but it has some fault-free nodes $P_t \in TESTED-BY(P_s)$ and $P_t \notin T$, then P_s will set $T = T \cup [P_t]$, and then interrogate each node P_t regarding the condition of P_r by sending a message [$?,T,P_q,P_r,P_s$] to P_i.

Consider a node such as P_s, which has interrogated a number of nodes (say P_t). If node P_s receives a message of the form [YES,P_q,P_r,P_t] from at least one of the nodes P_t, then it will pass a similar message [YES,P_q,P_r,P_s] to node P_q, which is its interrogator.

However, if node P_s does not receive at least a "YES" message from any of the nodes P_t, then it has to wait until it receives a message [NO,P_q,P_r,P_t] from all of them. A similar message of the form [NO,P_q,P_r,P_t] will then be sent to its interrogator P_q. These actions, which are described at node P_s will be followed by every other node, that has been interrogated.

At node P_q (the initial tester), a different set of actions are required to be taken against the receipt of a test result message. If at least one message of type "YES" has been received at node P_q from any of the nodes, that it has interrogated about the condition of P_r, then it recognizes that P_r is fault-free but the communication path between P_q and P_r is faulty. This information will be kept locally in $LNK\text{-}FLUR_q$. Otherwise, if P_q receives messages of type "NO" from all the nodes that it has interrogated about the condition of P_r then it will consider node P_r faulty, hence a message of the form [P_r *by* P_q *node*] will be broadcasted to every one of its testers.

Whenever a node P_v receives a message [P_r *by* P_q *node*] from a fault-free testee, it will consider P_r to be faulty and hence add P_r to its list of faulty nodes $ND\text{-}FLUR_v$ and it will send a message to every one of its testers.

Possible variants of *set T* formulations: When the modified SELF3 algorithm is chosen to be investigated, we find that a simulated distributed system requires a number of assumptions. Among these assumptions is a unified message format with fields that can cover and control detailed actions of the system. One of these fields holds a time stamp, representing local clocks of the nodes[19].

Earlier we mentioned that every interrogation message includes a set of nodes called *set T*, which is used to ensure that interrogation message travels only through acyclic paths. At a node P_q, which has accused another node P_r, the interrogation message(s) regarding the condition of P_r, that is(are) sent by P_q will have a *set T*, which is defined by; *T = all fault-free testees of* P_q. Consider that one of these messages is being received by node P_m, which is not a tester of node P_r. Node P_m will interrogate its fault-free testees $FT(P_m)$, provided they are included in set T of the message it has received. The interrogation messages, which node P_m will send, are provided with a *set T*, that is modified into $T = T \cup FT(P_m)$.

While the interrogation paths are branching in their search for a tester of the accused node, it is possible for a single node, especially in a graph with long paths, to be involved with more than one interrogation path. We will continue with our proposed case in which P_q accused P_r and assume that node P_t, which is not a tester of node P_r, has received messages from P_m and P_n interrogating about the condition of P_r. Let these carry sets T_m and T_n respectively. If this case is to be simulated, various situations can arise due to the differences in sets T_m and T_n and to the sequencing of execution of the messages by node P_t.

Let the fault-free testees of P_t be $FT(P_t)$ then these testees should be related to the sets T_m and T_n according to one of the following five situations:

(a) $FT(P_t) \cap \overline{T_m} \not\subset FT(P_t) \cap \overline{T_n}$ and $FT(P_t) \cap \overline{T_n} \not\subset FT(P_t) \cap \overline{T_m}$

(b) $FT(P_t) \cap \overline{T_m} = \phi$ and $FT(P_t) \cap \overline{T_n} = \phi$

(c) $FT(P_t) \cap \overline{T_m} = FT(P_t) \cap \overline{T_n}$

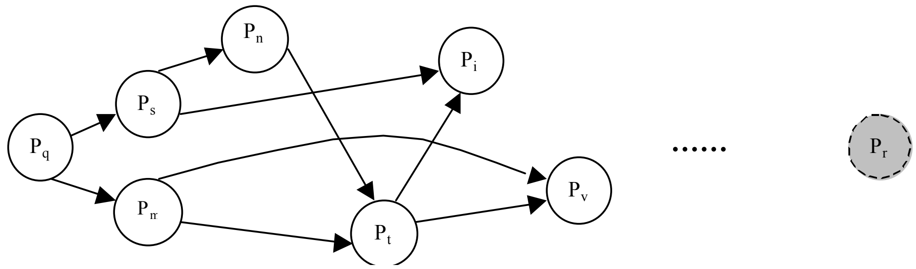
(d) $FT(P_t) \cap \overline{T_m} \supset FT(P_t) \cap \overline{T_n}$

(e) $FT(P_t) \cap \overline{T_m} \subset FT(P_t) \cap \overline{T_n}$

These five possibilities are illustrated in Fig. 1a-e. Each case (a-e) is assumed to represent part of an interrogation phase during a diagnosis procedure of node P_r. In this Fig. 1, we assume that node P_t originally has a set of fault-free testees composed of P_i and P_v and the nodes in this set are, for each case (i.e, a-e), assumed to be related differently, as testers and testees, to nodes P_m and P_n. This difference in relation between nodes P_m, P_n, P_i and P_v will cause the differences between $FT(P_t) \cap \overline{T_m}$ and $FT(P_t) \cap \overline{T_n}$.

The diagnosis scenarios: When node P_t executes the two messages, the actions that will be taken depend only on the contents of the sets T_m and T_n and not on which message is first or last to be executed. The situations for cases (a) and (b) are straightforward and therefore they will only be defined briefly in this section. In case (a), node P_t will interrogate different sets of testees when it executes the messages from P_m and P_n. Node P_i will be interrogated when the message from P_m is executed, while node P_v is interrogated when the message from P_n is executed. In case (b), however, the execution of each one of the two messages by node P_t will generate a reply of type "NO", since this node is neither a tester of node P_r nor has some fault-free testees that are not included in the sets T_m and T_n to interrogate.

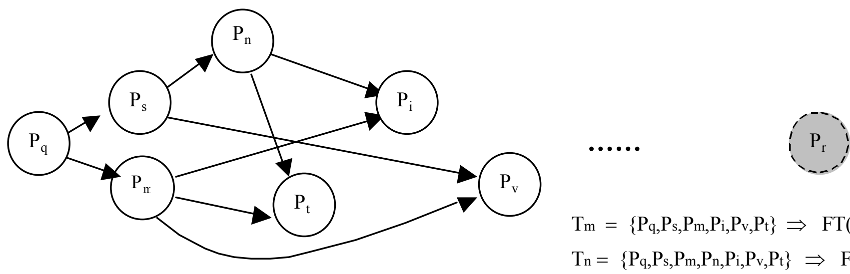
Meanwhile in cases (c-e), the situation is somehow different. In case(c), for instance, the same set of testees will be interrogated whichever message is executed first and this set will be interrogated again, when the other message is executed. In (d) and (e), however, the testees which will be interrogated twice, after executing the two messages, are those which occur in both $FT(P_t) \cap \overline{T_m}$ and $FT(P_t) \cap \overline{T_n}$. By interrogating a set of testees, or part of it, for the same reason more than once, node P_t has infact repeated similar actions. This has happened in cases (c), (d) and (e), where this node has executed all the interrogation messages it has received independently, considering only the contents of the *set T* and this infact complies with the algorithm of[11]. In this algorithm, a node like P_t is not required to consider its previous actions before sending an interrogation message to another node. Such a practice will result in generating more interrogation messages and hence the formation of a number of replicated paths, which can be considered as redundant. These extra messages have no advantage to the diagnosis process and even a deadlock. On the contrary, they may



$T_m = \{P_q,P_s,P_m,P_v,P_t\} \Rightarrow FT(P_t) \cap \overline{T_m} = \{P_i\}$.

$T_n = \{P_q,P_s,P_m,P_n,P_i,P_t\} \Rightarrow FT(P_t) \cap \overline{T_n} = \{P_v\}$.

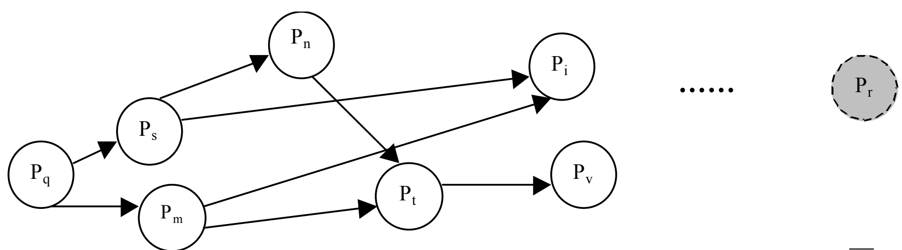
(a) $FT(P_t) \cap \overline{T_m} \not\subset FT(P_t) \cap \overline{T_n}$ and $FT(P_t) \cap \overline{T_n} \not\subset FT(P_t) \cap \overline{T_m}$



$T_m = \{P_q,P_s,P_m,P_i,P_v,P_t\} \Rightarrow FT(P_t) \cap \overline{T_m} = \phi$.

$T_n = \{P_q,P_s,P_m,P_n,P_i,P_v,P_t\} \Rightarrow FT(P_t) \cap \overline{T_n} = \phi$.

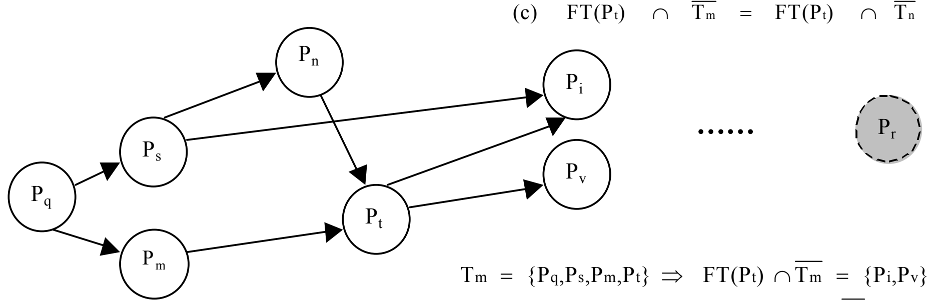
(b) $FT(P_t) \cap \overline{T_m} = \phi$ and $FT(P_t) \cap \overline{T_n} = \phi$



$T_m = \{P_q,P_s,P_m,P_i,P_t\} \Rightarrow FT(P_t) \cap \overline{T_m} = \{P_v\}$.

$T_n = \{P_q,P_s,P_m,P_n,P_i,P_t\} \Rightarrow FT(P_t) \cap \overline{T_n} = \{P_v\}$.

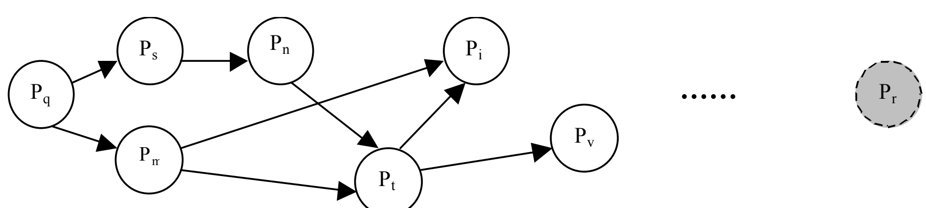
(c) $FT(P_t) \cap \overline{T_m} = FT(P_t) \cap \overline{T_n}$



$T_m = \{P_q,P_s,P_m,P_t\} \Rightarrow FT(P_t) \cap \overline{T_m} = \{P_i,P_v\}$.

$T_n = \{P_q,P_s,P_m,P_n,P_i,P_t\} \Rightarrow FT(P_t) \cap \overline{T_n} = \{P_v\}$.

(d) $FT(P_t) \cap \overline{T_m} \supset FT(P_t) \cap \overline{T_n}$



$T_m = \{P_q,P_s,P_m,P_i,P_t\} \Rightarrow FT(P_t) \cap \overline{T_m} = \{P_v\}$.

$T_n = \{P_q,P_s,P_m,P_n,P_t\} \Rightarrow FT(P_t) \cap \overline{T_n} = \{P_i,P_v\}$.

(e) $FT(P_t) \cap \overline{T_m} \subset FT(P_t) \cap \overline{T_n}$

Fig. 1: Different possibilities of $FT(P_t)$ with respect to T_m and T_n

incur additional delay in performing the diagnosis process. We consider that their prevention is of potential advantage and therefore assume that it is important for a node not to interrogate another node more than once for the same reason, when these two conditions hold:

* The first interrogation message, that has been sent is still waiting for a reply and
* The difference between the value of the time stamp of the first message, whose reply has not been received yet and the current clock value is within a specific timeout period.

The value of the timeout period may vary according to the size of the system and hence the expected length of the interrogation path. According to this assumption and the two conditions included in it, the situation at node P_t will be reassessed and this assessment, we will assume that conditions (1) and (2) above are always holding. Thus, for case (c), if the message from P_m is assumed to be executed first then all the testees in $FT(P_t) \cap \overline{T_m}$ will be interrogated. At a later instance, however, when node P_t executes the message from P_n then none of the nodes in $FT(P_t) \cap \overline{T_n}$ need to be interrogated because they have already been interrogated. In contrast, if node P_t has executed the message from P_n first, all the nodes in $FT(P_t) \cap \overline{T_n}$ will be interrogated, while none of the nodes in $FT(P_t) \cap \overline{T_m}$ need to be interrogated when executing the message from P_m. This is not the case for (d) and (e), however, where the precedence of executing the two messages makes a difference in the actions that node P_t has to take. Consider case (d) and assume that the message from P_n has been executed first, then all the testees $FT(P_t) \cap \overline{T_n}$ will be interrogated by P_t. consequently, when the message from P_m is to be executed, only part of $FT(P_t) \cap \overline{T_m}$ will be interrogated. This part includes the nodes which do not exist in $FT(P_t) \cap \overline{T_n}$ and hence have not been interrogated. For the same case (i.e. case (d)), if the message from P_m, is executed first it will result in interrogating the testees $FT(P_t) \cap \overline{T_m}$ by node P_t and because this set includes $FT(P_t) \cap \overline{T_n}$, none of its nodes need to be interrogated, when executing the message from node P_n at a later instance. After describing the situations for case (d), it can be shown how node P_t will behave towards the messages from P_m and P_n in case (e) in a rather similar way.

The number of extra interrogation messages and hence redundant replicated paths, which have been eliminated at node P_t due to the later assumption, is equal to $FT(P_t) \cap \overline{T_m}$ or $FT(P_t) \cap \overline{T_n}$, for case (c) and to $\left|FT(P_t) \cap \overline{T_m} - FT(P_t) \cap \overline{T_n}\right|$, for cases (d) and (e). These two measures represent a considerable reduction in the number of diagnosis messages and diagnosis time and hence leading to an improved performance of the algorithm.

CONCLUSION

Fault diagnosis forms an important tool in the maintenance strategy of distributed computer systems. The theory of fault diagnosis in distributed systems has received a considerable attention over the years and numbers of diagnosis algorithms were proposed in literature. Modified SELF3 algorithm is among these algorithms and it has been considered as a starting point in this study. Using a simulated distributed system, this algorithm is implemented, where all actions that were specified in the algorithm has been introduced to the simulator in a unified message format. A time stamp is appended to each message, which represents the local clock of the node from which the message is issued.

Various system topologies are used to investigate the algorithm. Originally, the algorithm includes a precaution to guard against replicating actions and assuring that messages traverse only acyclic paths. The simulation process, however, discovered that for certain cases, this precaution is violated and replicated actions may occur, which may even cause deadlock. Constraints are assumed in the study to handle this negative behavior and assure no replication in actions. These constraints have led to prevent the production of unnecessary messages and hence gaining an improvement in the performance of the algorithm. A measure of the improvement is given in the study.

REFERENCES

1. Kim, K., 1979. Error detection, reconfiguration and testing in distributed systems. Proc. 1st Intl. Conf. on Distributed Systems, pp: 284-295.
2. Randell, B. and P. Treleavan, 1978. Reliability issues in computing system design. Computing Surveys, 10(6): 123-165.
3. Preparata, F.P., G. Metze and R.T. Chien, 1967. On the connection assignment problem of diagnostic systems. IEEE Trans. on Electro. Comput.,12: 848-854.
4. Friedman, A.D. and L. Simoncini, 1980. System level fault diagnosis. Computer, 3: 47-53.
5. Maheshwari, S.N. and S.L. Hakmi, 1976. On modules for diagnosable systems and probabilistic fault diagnosis. IEEE Trans. on Computers, 3: 228-236.
6. Dahbura, A. and G.M. Masson, 1984. An O($n^{2.5}$) fault identification algorithm for diagnosable systems. IEEE Trans. on Computers, 6: 486-492.

7. Rangarajan, S. and D. Fussell, 1988. A probabilistic method for fault diagnosis of multiprocessor systems. 18th Intl. Symp. on Fault Tolerant Computing. Tokyo, 6: 278-283.
8. Kuhl, J.G., 1980. Fault Diagnosis in Computing Networks. PhD Thesis, University of Iowa.
9. Kuhl, J.G. and S.M. Reddy, 1981. Fault diagnosis in fully distributed systems. IEEE Symp. on Fault Tolerant Computing, pp: 100-105.
10. Hossieni, S.H, J.G. Kuhl and S.M. Reddy, 1984. A distributed algorithm for distributed computing systems with dynamic failure and repair. IEEE Trans. on Computers, 3: 223-233.
11. Hossieni, S.H, J.G. Kuhl and S.M. Reddy, 1988. On self-fault diagnosis on distributed systems. IEEE Trans. on Computers, 2: 248-251.
12. Griffith, G.W., 1986. A fault-tolerant distributed computer system for automotive applications. M. Sc. Thesis, Cranfield Institute of Technology, UK.
13. Bianchini, R., K. Goodwin and D.S. Nydick, 1990. Practical application and implementation of distributed system-level diagnosis theory. Proc. 20th Symp. on Fault Tolerant Computing, pp: 332-338.
14. Holt, R.C., 1972. Some deadlock properties of computer systems. ACM Computing Surveys, 4: 179-196.
15. Chandy, K.M., J. Misra, and L.M. Hass, 1983. Distributed deadlock detection. ACM Trans. Computer Systems, 1: 144-156.
16. Wu, H., W.N. Chen and J. Jaffer, 2002. An efficient distributed deadlock avoidance algorithm for the AND model. IEEE Trans. Software Engg., 1: 18 - 29.
17. Barbosa, V.C., 1990. Strategies for the prevention of communication deadlocks in distributed parallel programs. IEEE Trans. Software Engg., 11: 1311-1316.
18. Singhal, M., 1989. Deadlock detection in distributed systems. IEEE Computer, 22: 37-48.
19. Mohammed, T.S., 1992. Fault diagnosis of distributed systems: Analysis, simulation and performance measurement. Ph. D. Thesis, Cranfield Institute of Technology. UK.