Reliable Termination Detection Algorithm in Dynamic Distributed System

Archana Nair S1, Philip Samuel2
1Dept of computer science, 2Dept of Information Technology, School of engineering
Cochin University of Science and Technology, Kerala, India- 682022
1archa85@gmail.com, 2philips@cusat.ac.in

Abstract—In this paper, we present a symmetric, fault tolerant algorithm for detecting termination in dynamic distributed system. An important problem in the area of distributed computing is distributed termination detection (DTD). The problem has been well studied in static systems. Anyhow not much work has been done in dynamic systems. When dynamic systems are considered active processes may create other processes. In that situation our new algorithm will detect termination. The algorithm detects a terminated status by checking the status of the root node. The algorithm will detect a weak termination in the presence of faulty nodes. Even when some of the nodes fail, then the non faulty nodes can continue their local computation and inform the root about their terminated status. Basic idea adopted is diffusing computation. Another significance of our algorithm is no restriction to network topology.

Index Terms— Dynamic systems, Termination, diffusing computation, general networks, fault tolerance.

I. INTRODUCTION

An important problem in the field of distributed computing is determining termination of a distributed computation. As a basic problem in operating system design, termination detection refers to the necessity of determining whether a set of distributed processes has entered a terminated status and no more messages are in transit. Both termination and deadlock fit under the heading of quiescence. Quiescence means there are no messages in the system and all processes are waiting. The problems are similar since they both concern sensing when the system is in a state where there is no way to continue the computation.

The algorithms so far developed in the literature focuses on static system only, less work have been done on dynamic system. The task of termination detection in dynamic systems is more difficult because the exact number of processes participating in the computation is not known at any instant of time. Most of the algorithms found in the literature are trying to detect termination with the precondition that no processes are allowed to leave the system before the computation terminates.

This paper presents a termination detection algorithm for dynamic systems. It is reliable in the sense that it handles node failures and detect a weak terminated in the presence of faulty nodes. A diffusion based scheme is applied here.

II. RELATED WORKS

After its introduction in 1980 many works have been done in distributed termination detection[1,4,9,]. Some algorithms take the advantage of a global time base in the system [17]. Some algorithms assumes synchronous communication [11], while others work equally well for asynchronous systems [5]. Some algorithms require FIFO channels [14], and some do not [1,7]. Most algorithms are intended for static systems only [6,13,16,17]; algorithms for dynamic systems are rare [2,14,7]. Most of the algorithms handle termination detection only; they are not considering the issues when node failure occurs. Some fault tolerant algorithms can be found in [3,8,12,15]. A good review of the existing algorithms can be found in [10]. This paper examines the fundamental problem, the termination detection problem in dynamic systems and tries to detect a terminated state even some of the nodes in the computation fail.

The algorithms in [14] are concerned with special cases of dynamic systems, systems with synchronous communication in which processes may be created but not destroyed. In [7] a termination detection algorithm for dynamic system is given, where processes may be created and destroyed, the communication can be synchronous or asynchronous and message need not be delivered in the FIFO order. The algorithm is asymmetric. It also requires a process to participate in termination detection even after it has been destroyed.

Several works have been done to detect termination in presence of faulty node, but the works are for static systems. In [8] a fault tolerant algorithm based on the DS algorithm is presented. In that algorithm when a process detects a fault in the system, let it adopt the coordinator as its parent and it send a signal to the latter when it is done. The signal carries a set of processes that contains all faults the process has so far detected. The set of detected faults intended to help the coordinator know whether every child has signaled its completion of processing. The algorithm assumed that the underlying communication network provides reliable end to end communication, failure detection and fail flush.

In [15] another fault tolerant algorithm is presented. In that algorithm each node is associated with k other representative. When a node fails at least one of its representatives is still alive and can do the computation.
III. THE PROBLEM AND THE MODEL

Termination is detected in a distributed system when a process determines that the distributed computation has completed and there are no pending messages. We define several terms in order to describe distributed systems and distributed computations in this section.

In this paper we consider the three states of a process, active, passive and terminated. Active processes are those currently working on a computation. In passive state, a process does not perform any arithmetic operation and cannot send any messages; but it can receive an incoming message. The terminated process can no longer receive or send any messages. The messages for computation are called basic messages.

In dynamic distributed systems processes are created dynamically. The computer network is represented by an undirected graph G= (V, E) where V is the set of nodes and E is the set of communication channels.

Our algorithm made the following assumptions.

- Root node has the least process id.
- There is always a path from all the nodes in the computation to the root node.
- In the time of failure, the network is not partitioned.
- Undelivered messages are discarded.

IV. ALGORITHM

A. Messages and variables in the algorithm

Basic message: Message sent by one active node to an idle node. After sending basic message each node will start a timer. Within this time it has to receive a passive message from the child node.

Passive message: When a node completes its part of the local computation it will send a passive message to all the nodes from which it had received a basic message.

Terminated message: A message sent by terminated process to its parent nodes.

Hello message: Message sent from a parent process to a child process in the time when no passive message is obtained after a predefined time.

ACK message: Message sent in response to the hello message.

Each node will maintain two data structures

childList: Process id of all the nodes to which node had sent basic message.

parentList: Process id of the messages from which the node receives basic message. The active root node will send basic message to an idle node. Both nodes will update their parentList and childList. The newly activated node will activate other idle nodes. And in this way the computation network grows. When a node completes its local computation it will send a passive message to all the nodes in its parentList within a predefined time. If a node becomes faulty it failed to send the passive message in time. Then the parent will send a hello message to the child. If the child is active then it will respond with an ACK signal. Then the parent will start a new timer and wait for the passive message from the child. If there is no reply it means the child node become faulty. Then the parent will remove the faulty node from the childList and send the faulty message to the root node so that root will set its faulty flag which indicate a weak terminated status. After sending a passive message a node changes its state to the passive state. Then the node checks the childList. If the list is empty the process change its state to terminated and send a terminated message to all the parents. If not it will remain in the passive state and can receive messages from the child nodes. If the parent node fails the child detect it while sending the passive message. Then the child will adopt the root as its parent by sending a joining report to the root. After getting the joining report the root consider that nodes as its child and update the childList and set the faulty flag. Then the node will send the passive message to the root node. In this way, each node in the computation network becomes terminated. When the root itself becomes terminated, it checks the flag. If the flag is set a weak termination is detected otherwise root detect a terminated status.

B. Pseudo code

Root
Send basic message.
Start the timer.
Update the childList.
If a fault message is received
Set the faulty flag
When the computation completes
Check the childList
If (empty ()
Check the flag
If not set
Weak termination is detected
Else
Detect termination
Else
Remain in passive state
All nodes
Send basic message.
Start the timer & Update the childList.
If the time up
Send hello message to child.
If get ACK
Start the timer
Else
Update the childList
Send the fault message to the root node.
Receive the basic message and Update parentList
If (passive)
Send passive message to parent.
If parent is faulty, Send passive message to root node.
Check the childList
If (empty ()
Change state to terminated and send terminated message to the parent.
In figure 1 node 0 is the root node which starts the computation. Nodes 1 and 2 become the child nodes of root node by sending basic message to the nodes by the root node. Figure 2 shows the computation network at a particular time. After sending the basic message each node will start the timer as stated in the algorithm. Each node will keep the list of its parent nodes as well as the child nodes.

If node 6 complete the computation then the following events takes place.

- Send passive message to the node 3
- Change state passive.
- Check the childList.
- It is empty and sends the terminated message to the node 3.

Consider the situation in which one node fails. Suppose node 2 fails. Then node 3 and node 4 takes node 0(root) as their parent by sending a joining message to the root node and root update its childList. Then they send the passive report to the root node. Then the algorithm can detect the termination.

VI. CONCLUSION

This paper examines the distributed termination in dynamic distributed systems and presents a symmetric algorithm for termination detection. When studying distributed termination detection in a dynamic distributed system we need to consider the frequent creation and destruction of processes. When all the processes in the terminated state the algorithm will announces the strong termination. This algorithm detects termination in the presence of faulty nodes also. If some of the nodes fail, the algorithm detects weak termination. It takes the advantage diffusing computation. There is no restriction to network topology. A node can send basic message to more than one nodes and a node can receive message from more than one active nodes.

VII. REFERENCES