ISSN (Online): 2230-7893
www.IJCEM.org

A General Model for Distributed Termination Detection in Dynamic System for Non-Diffusing Computation

Vishal Garg[1], Swati Malhotra[2], Chandan Kapoor[3]

[1] A.P.,Department of Computer Science and Engg., JMIT
Radaur, Yamunanagar, Haryana, India
vishalgarg.9@gmail.com

[2] A.P., Department of Computer Science and Engg., APIIT
Panipat, Haryana, India
swatimalhotra87@gmail.com

[3] A.P., Department of Computer Science and Engg., JMIT
Radaur, Yamunanagar, Haryana, India
chandankapoor.123@gmail.com

Abstract

A symmetric algorithm for detecting the termination of a distributed computation is presented. The algorithm works for dynamic systems allowing creation and destruction of processes during the course of computation. The algorithm does not require global information concerning the system and does not assume any communication features barring finite delays in the delivery of messages. The algorithm requires acknowledgement for basic messages received. The algorithm is discussed with a suitable example.

Keywords: *Distributed system, Dynamic system, Distributed termination, Termination detection*

1 Introduction

Since it was first introduced in 1980 by Francez ([2]) and by Dijkstra and Scholten ([1]), the termination detection problem, i.e. to find out whether a distributed computation being performed in a system has terminated, is a very vital problem in distributed programming.

Taking into account distributed execution of an application program which is usually called the basic computation to be performed, the motive is to construct a superimposed control program, which will detect the termination of the underlying computation. In distributed systems, processes communicate solely by message passing, the detection of termination of a distributed computation is non trivial as no process has complete knowledge of the global state of the computation, and as global time or common memory do not exist in distributed setting.

According to Mattern: "A distributed computation is considered globally terminated if every process is locally terminated and no messages are in transit. Locally terminated can be understood to be a state in which a process has finished its computation and will not restart unless it receives a message" (see Section 2 of Introduction of [3]).

This definition is implicitly based on the following model for distributed computations:

- an active process can execute statements modifying its internal state, send messages or become spontaneously passive (i.e. locally terminated),
- a passive process can only receive messages (one at a time) and then instantaneously it becomes active.

In this context termination means that all processes are passive and all messages sent have been received and interpreted (consumed) by their destination processes. Many distributed algorithms have been designed to detect such a termination.

2. Related work

The DTD problem has been studied extensively since last two decades. One of the earliest works in this area is an algorithm by Dijkstra and Scholten, based on the diffusing computation model [1]. This approach was later extended by Misra and Chandy [8] and Cohen and Lehmann [5]. The algorithms in this area can be broadly classified as “symmetric” and “asymmetric” algorithms.

In the symmetric algorithms [9] all the processes execute identical code and any process can detect the termination. The asymmetric algorithms [1, 3,7,8] rely on a pre-designated process for termination detection which may be

called as the controlling agent. Termination detection algorithms may also be divided into three classes depending upon the topology employed for the control communication, viz. Hamiltonian cycle [7], spanning tree [2,3,4,5] and general networks[1,8, 9].

3. Overview of Distributed Systems

In the field of distributed systems, the problem of detecting the termination of a distributed computation has been well studied. Distributed termination detection (DTD) was introduced in 1980 independently by Francez (Francez, 1980) and Dijkstra and Scholten (Dijkstra and Scholten, 1980).
Since its introduction, and especially during the mid 1980s, DTD has been a popular problem to research. Although it is not popular as a research topic today, one or two new algorithms a year continue to be presented. DTD is difficult to achieve efficiently as it is a global state detection problem. The global state of a distributed system consists of states of each of the processes in the system and the states of the channels in the system. Because processes in a distributed system do not share clocks or memory, synchronization problems make the detection of global state difficult. An algorithm which accomplishes global state determination has been presented by Chandy and Lamport (Chandy and Lamport, 1985), and termination detection has been solved using this global state approach (Misra and Chandy, 1982; Misra, 1983; Chandy and Misra, 1985, 1986a).

3.1 Process and the communication model

A distributed system is defined as a set of n processes, $P=\{p_1,p_2,\ldots,p_n\}$, which are distributed across a network, and a set of communication channels, E, by which the processes can transfer information. The communication channels are sometimes considered as directed channels in DTD algorithms, but are most often bidirectional. This set theoretic definition of a distributed system can be naturally represented as a graph. Thus, processes are often discussed as nodes or vertices and communication channels are often discussed as edges.

We make the following assumptions about any distributed system.

- There is no shared memory. Hence, information that is to be shared must be transmitted from one process to another by a message along some channel in E.
- There is no common clock. For example, there is no way to ensure that all processes perform some action at exactly three o'clock.
- Communication takes arbitrary, but finite, time. In other words, a process cannot predict when a message is to be delivered and, therefore, the sender does not know when the receiving process is cognizant of the information within the message.

The computation that is being executed by the distributed system is called the basic computation. Messages that are involved in this basic computation are called basic messages. We define M to be the number of basic messages used in the computation. Processes in the system can be in either of two states: active, in which the process is currently working on the basic computation, or passive, in which the process is currently waiting to be either activated or terminated. We define a model of a distributed system to constrain the action of active and passive processes.

Processes behave according to the following rules.

- Initially, each process in the system is either active or passive.
- An active process may become passive at any time.
- Only an active process may send a basic message to another process.
- A passive process can only become active if it receives a basic message.

The above assumptions ensure that the system behaves in a predictable manner. If one can determine that all processes in the system are passive and there are no basic messages in transit, one has detected termination. These two conditions are the necessary and sufficient conditions for detecting termination given the four rules of distributed systems behavior above.

4. Overview of Dynamic Systems

Most of the algorithms mentioned above work only for static systems, i.e. for systems comprising of a fixed set of processes. There is relatively less work on dynamic

systems, where processes may be created as well as destroyed while the computation is in progress.

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. Also, since processes may destroy themselves, the algorithm has to ensure that:
(i) the computation does not get partitioned, and
(ii) a process capable of detecting termination always exists in the system.

5. Overview of the paper

In the paper we present a symmetric termination detection algorithm for dynamic systems which allows unrestricted creation and destruction of processes. A distributed computation is assumed to be composed of a set of concurrent processes {P*i*}, with each process allotted a specific computational task to be performed. Processes can be created or destroyed while the computation is still in progress.

Once the task allotted to the process is completed, the process becomes passive and awaits one of the following events:

- assignment of a new computational task that happens through the receipt of a basic message from some other process,
- receipt of a message killing it, or
- declaration of global termination.

The process becomes active again after receiving a basic message, it informs other processes of its destruction and dies after encountering a kill message, while in case of declaration of global termination, it simply terminates. The interprocess communication is assumed to be asynchronous with arbitrary but finite delays and possible non-FIFO delivery of messages.

Each process maintains a data structure called the neighbor set (NS) to store the identities of its neighbors. Whenever two processes P*i* and P*j* communicate with one another, their NS's are updated with each other's id's. Similar actions are performed when process P*i* creates process P*j* . When a process P*i* kills process P*j* , or P*j* kills itself, P*j* informs each P*k* belonging to set NS*j* of its destruction. This leads to the deletion of P*j* from NS*k*. Certain new processes may be added to NS*k* to avoid network partitioning. This ensures that all existing processes participate in the termination detection.

Termination detection is performed in a symmetric manner. The basic model is that of a diffusing computation [1]. Every time a process becomes passive, it initiates termination detection by sending a termination detection (***detect***) message to each process in its NS. To limit the communication overheads, each ***detect*** message contains a broadcast set (BS) containing the id's of all processes to which the ***detect*** message has been (or is being) sent. A passive process receiving the message propagates it to those of its neighbors which are not already in the BS. It replies with a ready to terminate (***ready***) message only when each such neighbor replies with a ***ready***, or when it has no neighbours to whom it should send the termination detection message. The process initiating the termination wave concludes that termination has occurred when it receives the ***ready*** messages from all its neighbours. Each process has a unique process identification number (PID) which is an ordered pair (sequence number, process id). Sequence numbers are used analogous to synchronized logical clocks. The sequence number of a process is initially zero, gets incremented by one every time the process initiates termination detection and is updated whenever a control message with a higher sequence number is received. A process ignores a ***detect*** message if it is active, or has a PID larger than that of the initiator of the ***detect*** message. This ensures correctness of termination detection and also minimizes the control message traffic. Termination is thus detected by the process with the highest identification number. In our algorithm the receiver of a basic message is required to send an acknowledgement to the sender. Each process Pi maintains a counter (COUNTi) to record the number of basic messages sent by it for which no acknowledgements have been received. A process cannot become passive, or be killed, as long as its COUNT is non-zero.

6. Representation of the System and Features of Processes

A distributed computation is a pair (PS, CG), where PS is the non-empty set of permanent processes with which the system begins its execution. A permanent process exists during the entire lifetime of the computation, i.e. it cannot be killed. CG = (P, E) is an undirected graph, wherein P is a finite set of processes {P*i* | P*i* is live}, and E is a set of edges {(P*i*,P*j*) | P*i* $\in$ NS*j* , P*j* $\in$ NSi}.

At system initiation, CG = (PS,E*o*), where
E*o*={(P*i*,P*j*) $\forall$ P*i*,P*j* $\in$ PS}, i.e. all permanent processes are connected to each other. A new edge is added to CG when two processes communicate for the first time. When a new process Pi is created, a vertex Pi and an edge between Pi and its creator are added to CG. When a process is destroyed, the corresponding vertex and all

edges connected to it are removed from CG and some new edges are added to avoid system partitioning.

The physical network for inter-process communication is a connected single component graph with bi-directional links.

6.1 Features of Processes

STATE*i* : Status of the processes is represented by this, which can have the following values:

- **White**: active process
- **Grey**: passive process, has propagated the ***detect*** message and is waiting for replies.
- **Black**: passive process and has sent a ***ready*** reply to a ***detect*** message.

Sequence*i*: Current sequence number of P*i*.
Static: ID of some permanent process.
NS*i*: Set of IDs of neighbors of process P*i*.
COUNT*i*: Count of basic messages sent by Pi for which no acknowledgement has been received.

6.2 Main Control Messages in the Algorithm

- *ack*: Acknowledgement sent for basic message received.
- *detect*: Sender propagates its passive condition and maintains a broadcast set.
- *ready*: It indicates that the sender and all its neighbors are ready to terminate.
- *kill_p*: Sender wants receiver to kill itself.
- *kill_m*: It indicates that sender has killed itself.
- *Terminate*: It indicates that the processes involved in the basic computation may terminate.

7. Discussion of the Algorithm with Example

The example presented in this section illustrates the following features of the system's functioning:

a) How activation of a black process (i.e. a process which has replied to a ***detect*** message with ***ready***) does not lead to a false termination detection.

b) How the system graph is transformed to preserve connectivity when a process is killed, and

c) How distributed termination is concluded by the algorithm.

Figure 1 shows a system consisting of four processes. P1 is the only permanent process, hence it is the static for P2, P3 and P4. Let processes P1, P2 and P3 be white, process P4 be grey and let no basic or control messages be in transit. Let process P1 turn grey and let Seq$_{P1}$ = x such that x is the largest sequence number in the system and PI$_{P1}$ = (x, P1) is the maximum of all PI's in the system. The detect message from P1 is received by P4 which replies with a ready message and turns black. Hence PID$_{P4}$.Seq = PID$_{P1}$.Seq = x.

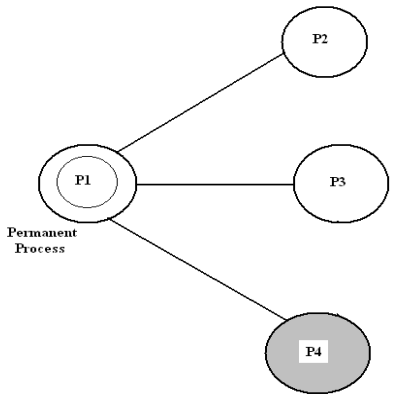

Figure 1: Initial configuration

Now, consider the following sequence of events:

(i) P4 receives a basic message from P3, sends ***ack*** and becomes active.

(ii) P4 creates a new process P5, and sets Seq$_{P5}$ = Seq$_{P4}$. The system now looks as shown in figure 2.

(iii) P3 receives the ***ack*** for the basic message sent to P4 and sets Seq$_{P3}$ = Seq$_{P4}$ = x.

(iv) P2 being active ignores the detect message and later kills itself by sending ***kill_m*** message to P1, which removes P2 from its NS.

(v) Node corresponding to P2 and edge connecting P2 with P1 is removed from the graph. The system now looks as shown in figure 3.

(vi) P3 becomes grey and sends its own ***detect*** messages with PID = (x + 1, P3) to P4 and P1.

(vii) P3 receives the ***detect*** message initiated by P1 and ignores it because its own PID is higher. Hence P1 cannot conclude termination, thus illustrating feature (a) mentioned above.

(viii) P4 kills itself sending ***kill_m*** messages to P1, P3 and P5.

(ix) P1 receives the ***kill_m*** message from P4, and adds P5 to NS_A. It sets Seq_{P1} = x + 1, initiates new detect messages and sends them to P3 and P5.

(x) P3 receives the ***kill_m*** message from P4. P4 is now removed from $TDSET_{P3}$. No processes are added to NS_{P3}, since S_{P4} = P1 is already in NS_{P3}.

(xi) P1 receives the ***detect*** sent by P3 (with PID = (x + 1, P3)), and ignores it because its own PID is higher.

(xii) P5 receives ***kill_m*** from P4 with PID = (x, P4). It sets PID_{P5}.Seq = x + 1 and adds P1 to NS_{P5}. The new configuration of the system is the connected graph shown in figure 4. This illustrates feature (b) mentioned above.

(xiii) P5 becomes grey and sends a ***detect*** message with PID = (x + 2, P5) to P1.

(xiv) P5 receives P1's ***detect*** with PID = (x + 1, P1) and ignores it because its own PID is higher.

(xv) P3 receives P1's ***detect*** message and replies with a ***ready.***

(xvi) P1 receives P5's ***detect***, and propagates it to P3.

(xvii) P3 receives P5's ***detect*** from P1 and replies with a ***ready*** message.

(xviii) Finally, P1 returns ***ready*** message to P5, and P5 concludes termination, thus illustrating feature (c) mentioned at the start of this section.

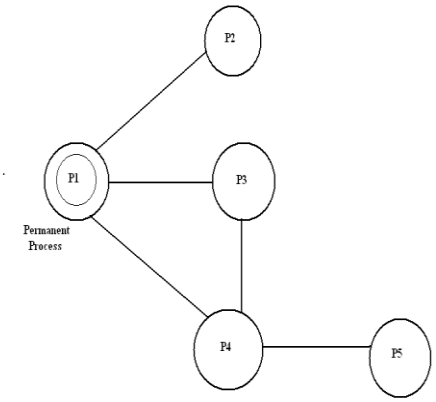

Figure 2: Activation of process P4

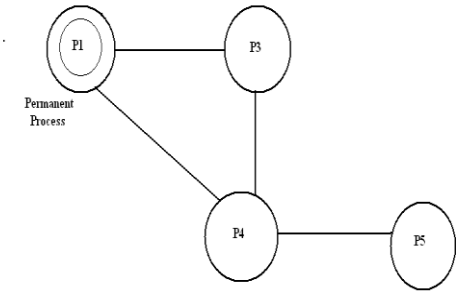

Figure 3: After process P2 kills itself

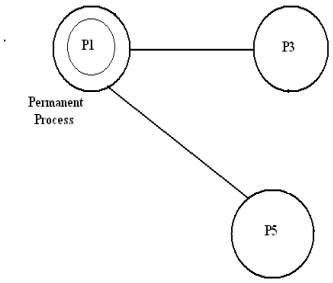

Figure 4: After process P4 kills itself

8. Conclusion

The algorithm discussed in our paper works well for distributed dynamic systems in which the processes may be created or destroyed during the course of the computation. The algorithm is symmetric and based on diffusing computational model. The algorithm covers various key features like it results in correct termination even if a black process is reactivated, connectivity is maintained even if a process kills itself and there is always a process in the system to detect termination. In the algorithm discussed the receiver is required to send acknowledgement for the basic message received. The algorithm can be studied further and enhanced by reducing the communication overhead by eliminating the acknowledgements to the basic messages involved in the computation. Also the case of destruction of the process by external process can be discussed. The algorithm assumed that all the processes work reliably. But in actual processes may crash or become faulty during the course of the computation. All these cases can be discussed in future.

9. References

[1] E. W. Dijkstra and C. S. Scholten, *Termination detection for distributed computations*, Information Processing Letters, vol. 11, no. 1, pp. 1-4, 1980.

[2] N. Francez, *Distributed termination*, ACM Transactions on Programming Languages and Systems, vol. 2, no. 1, pp. 42-55, 1980.

[3] F. Mattern, *Algorithms for distributed termination detection*, Distributed Computing, vol. 2, pp. 167-175, 1987.

[4] S. Chandrasekaran and S. Venkatesan, A message-optimal algorithm for distributed termination detection, Journal of Parallel and Distributed Computing, vol. 8, no. 3, pp. 245-252, 1990.

[5] S. Cohen and D. Lehmann, Dynamic systems and their distributed termination, Proceedings of the first Annual ACM Symp. on Principles of distributed computing, Ottawa, 1982, pp. 29-33.

[6] D. M. Dhamdhere, E. K. K. Reddy and S. R. Iyer, Distributed termination detection for dynamic systems, TR-081-92, IIT Bombay

[7] E. W. Dijkstra, W. H. J. Feijen and A. J. M. van Gasteren, Derivation of a termination algorithm for distributed computations, Information Processing Letters, vol. 16, pp. 217-219,1983.

[8] J. Misra and K. M. Chandy, Termination detection of diffusing computations in communicating sequential processes, ACM Transactions on Programming Languages and Systems, vol. 4, no.1, pp. 37-43, 1982.

[9] S. T. Huang, A fully distributed termination detection scheme, Information Processing Letters, vol. 29, no. 1, pp. 13-18, 1988.

Vishal Garg received his degree of B.Tech and M.Tech and have 5 years teaching experience and also 1 year of industrial experience. He is presently working as Asst. Professor at JMIT, Radaur. He is a member of Computer Society of India and has published paper in IEEE conference. His main area of research is Wireless Communication and Advanced Databases.

Swati Malhotra received her degree of B.Tech and now pursuing M.Tech and also working as Assistant Professor at APIIT, Panipat. Her main area of interest are data structures and databases.

Chandan Kapoor received his degree of B.Tech and now pursuing M.Tech and presently working as Asst. Professor at JMIT, Radaur. His main area of interest are Mobile Communication and Operating Systems.