Proc. of Int. Conf. on Current Trends in Eng., Science and Technology, ICCTEST

Survey On Algorithms To Achieve Coordination Among Nodes In Distributed Systems

Vinaka Patil
DonBosco Institutue of Technology :CSE dept, Bangalore, India
vinaka.dbit@gmail.com

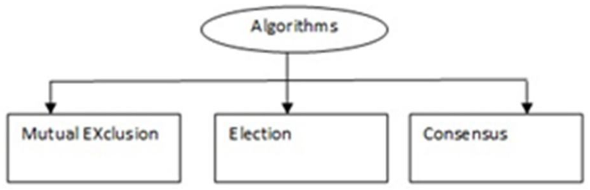
***Abstract*—A Distributed system serves as an application which executes a set of protocols to coordinate the actions of many processes on a network, such that all components cooperate together in order to perform a single or a very small set of related task. It is very difficult for the processes to cooperate with one another because of failures of any of the process during communication. In this paper a survey on all such algorithms and a comparitive studies is made.**

***Index Terms*—Distributed system;coordination; Algorithms(key words).**

I. INTRODUCTION

Distributed systems is a collection of computers that act, work, and appear as one of the large computer system. There are several distributed computing projects on the Internet that helps to work on complex problems by sharing the processing power of millions of different peoples computers. Coordination among nodes becomes very difficult when consistency is needed among all nodes. Centralized controlling nodes can be selected from the group of available nodes to reduce the complexity of decision making [2]. Many algorithms require one node to act as coordinator, initiator, or otherwise perform some special role. Leader election is one of the technique that can be used to break the symmetry of distributed systems. In order to determine a central controlling node in a distributed system, a node is usually elected from the group of nodes as the leader which serves as the centralized controller for that decentralized system. The purpose of leader election is to choose a node that coordinates activities of the system. In most of the election algorithm, a leader is chosen based on some criterion such as choosing the node with the largest identifier. Once the leader is elected, the nodes reach a particular state called as terminated state. In leader election algorithms, these states are partitioned into elected states and non-elected states respectively. When a node enters either state, it always remains in that state [3]. All leader election algorithm must be satisfied by the safety and liveness condition for an execution to be admissible. The liveness condition states that every node will eventually enter an elected state or a non-elected state. The safety state for leader election requires that only a single node can enter the elected state and eventually, becomes the leader of the distributed system. Information is exchanged between each other nodes by transmitting messages to each another until an agreement is reached. Once a decision is made, then a node is elected as the leader and all the other nodes will acknowledge the role of that node as the leader

A. Categories Of Algorithms



II. Mutual Exclusion

Two or many processes should not be allowed to access a shared resource simultaneously. In a distributed system, nodes must negotiate via message passing. The various algorithms on mutual exclusion should ensure Safety: At most one process may execute in the critical section (CS) at a time, Liveness means: Requests to enter and exit critical section eventually succeed, Causal ordering: If there is one request to enter the CS happened-before another, then entry to the CS is granted in that order only.

A. Central Server

The first algorithm uses a central server which manage access to the shared resource area. To enter a critical section, a process sends a request to the server. The server behaves as follows:If there is no one is in a critical section, then server returns a token. When the process exits the critical section, the token is returned to the server. If someone already has the token, the request is queued. Requests are serviced in FIFO order. If there is no failures occuring, then this algorithm ensures safety and liveness. However, ordering is not preserved (why?). The central server is a bottleneck and a single point of failure.

B. Token Ring

The token ring algorithm arranges al the processes in a logical ring, then a token is passed clockwise around the ring. When a particular process receives the token it can make an entry in its critical section. If it does'nt need to enter a critical section, then it immediately passes the token to the next available process. This algorithm also achieves safety and liveness, but no ordering, in the case when no failures occur. However, a significant amount of bandwidth is used because the token is circulated continuously even when no process needs to enter a CS area.

C. Multicast And Logical Clocks

Each process has a one unique identifier and maintains a logical clock. A process can be in one of the three states i-e released, waiting, or held. When a process wants to enter a CS it will do the following things:1) sets its state to waiting 2) sends the message to all other processes containing its ID and timestamp once all other processes responds, it can enter the CS.When a message is received from another process, it does the following activities :if the receiver process state is under held, then the message is queued, if the receiver process state is waiting and the timestamp of the message is after the local timestamp, the message is queued (if the timestamps are the same, the process ID is used to order messages) else replies immediately. When a process exits a Critical section , it does the following:sets its state to released, replies to available queued requests.

III. Election

An election algorithm determines which process will play the role of coordinator or server. All processes need to agree on the selected process. Any process can start and initiate an election, for ex: if it notices that the previous coordinator has failed. The requirements of an election algorithm are as follows: Safety: Only one process is chosen -- the one with the largest identifying value. The value could be load, uptime, a random number, etc. Liveness: All process eventually choose a winner or crash

A. Ring-Based

The processes are arranged in a logical ring. A process starts an election by placing its ID and value in a message and sends the message to its neighbor. When the neighour receives the message, a process does the following activities:If the value is greater than its own, it has to saves the ID and forwards the value to its neighbor. Else if its own value is greater then and if it has not yet participated in the election, then it replaces the ID with its own and forwards the message. Else if it has already participated it discards the message. If a process receives its own value and ID, it knows it has been elected. It then sends an elected message to its immediate neighbor. When an elected message is received than it is forwarded to the next neighbor.

B. Bully

The bully algorithm is capable to deal with crash failures, but not with communication failures. When a process notices that the coordinator has failed to work, then it sends an election message to all higher-numbered processes. If no processes replies, it declares itself as the coordinator and sends a new coordinator message to all available processes. If someone replies, it does nothing else. When a process receives an election message from a lower-numbered process it returns a reply and starts an election. This algorithm guarantees safety and liveness and is capable to deal with crash failures.

IV. Consensus

All of the previous algorithms are examples of the consensus problem: how can we get all processes to agree on a state? For example, we look at when the consensus problem which is solvable. The system model considers a set of processes pi (i = 1,2,.., N). Communication is reliable, but processes may fail. Failures could be crash failures or byzantine failures. The following are goals of consensus: Termination: Each and Every correct process eventually decides on a particular value. Agreement: All processes agree on a particular value. Integrity: If all correct processes propose the same value, that value is selected. We consider the Byzantine Generals problem. A set of generals must agree on whether to attack or retreat. Commanders can be faulty. This is similar to consensus, but differs in that a single process proposes a value that the others must agree on. The requirements are defined: Termination: All correct processes eventually decide on a some value. Agreement: All correct processes agree on a particular value. Integrity: If the commander is correct, all correct processes agree on what the commander gives the proposal. If communication is unreliable, consensus becomes impossible. .we can solve consensus in a synchronous system with crash failures. We can solve Byzantine Generals in a synchronous system as long as there is less than 1/3 of the processes fail. The commander sends the command to all of the generals and each general sends the commands to all other generals. If each correct process chooses the majority of all commands, the requirements are met. Note that the requirements do not specify that the processes must detect that the commander is fault. It is impossible to guarantee consensus in an asynchronous system, even in the presence of 1 crash failure. That means that we can design systems that reach consensus most of the time, but cannot guarantee that they will reach consensus every time. The techniques for reaching consensus in an asynchronous system include the following: Masking faults – Which hide failures by using persistent storage to store state and restarting processes when they crash. Failure detectors – It treats an unresponsive process (that may still be alive) as failed. Randomization - Uses randomized behavior to confuse byzantine processes.

A. Comparative Analysis Of The Algorithms

Table 1:Comparative Analysis

Algorithm	Efficiency/Drawback
Central server	Failures will be more
Token Ring	Ensures safety &liveness, but does not withstand failures
Multicast & logical locks	Ensures safety &liveness, but does not support scalability
Ring based	May lead to single point of failure
Bully	Guarantees safety and liveness and can deal with crash failures.
Consensus	Guarantees safety and liveness and can deal with crash failures. &has higher efficiency

V. Conclusion

A distributed system consists of huge collection of computers. These computers can work together efficiently, but all these systems are independent. A leader's election is a basic necessity for distributed systems. When any system is chosen as a leader, it should operate as a system’s management; make final decisions and the like. There are several election algorithms available in Distributed system. Here in this paper we discussed the concept of some existing algorithms. In wireless networks, leader election has a variety of applications such as: key distribution, routing coordination, sensor coordination, and general control.

References

[1] A prioritized distributed mutual exclusion algorithm balancing priority inversions and response time Jonathan Lejeune Luciana Arantes, Julien Sopena, and Pierre Sens

[2] Fair K Mutual Exclusion Algorithm for Peer To Peer Systems ∗ Vijay Anand Reddy, Prateek Mittal, and Indranil Gupta University of Illinois, Urbana Champaign, USA

[3] Improved Bully Election Algorithm in Distributed Systems A. Arghavani E. Ahmadi A.T. Haghighat. Proceedings of the 5th International Conference on IT & Multimedia at UNITEN (ICIMU 2011) Malaysia.

[4] Leader Election Algorithms in Distributed Systems Seema Balhara1 , Kavita Khanna2

[5] Easily rendering token-ring algorithms of distributed and parallel applications fault tolerant Luciana Arantes and Julien Sopena LIP6 - Universite de Paris 6 - CNRS - INRIA Rocquencourt