Optimized Bully Algorithm

Sathesh B.M
Programmer Analyst,
Cognizant Technology Solutions,
Chennai, India

ABSTRACT
All distributed systems require one process to act as a coordinator , initiator or otherwise perform some special role .In general , it does not matter which process takes on this special responsibility , but one of them has to do it . The goal of an election algorithm is to ensure that when an election starts, it concludes with all processes agreeing on who the new coordinator is to be. Bully Algorithm by Garcia-Molina is a classic algorithm for leader election in a distributed system. Although the already existing algorithm solves the purpose, the traditional bully algorithm takes lot of message passing involved and it does not provide facilities to ensure that what will happen when dead leader recovers back again. Here we propose a slight modification in the classic bully algorithm which reduces the number of messages that are needed to elect the leader. Also we suggest methods on how to react when the dead leader recovers back again. The end result is a modified election bully algorithm which is much efficient than the existing leader election algorithms used in a distributed environment.

General Terms
Election Algorithms, Distributed Computing.

Keywords
Bully Algorithm, leader election, Message passing.

1. INTRODUCTION
A collection of independent computers, having a common goal of solving a complex problem are commonly called as distributed systems. Earlier computing of data were limited and centralized to a single processor or a single computer. The systems are interconnected via a network; capable of collaborating on a single task. Distributed computing systems each have their own memory where information is exchanged through passing messages mechanism called message passing among the various processors interconnected.

Synchronization and self-stabilization is a common challenge faced by distributed systems. In traditional computers when the input is passed a computer or a processor processes the input for a while and then it produces the output and the processing is stopped. But for some computer science problems such as the dining philosopher's problem the processing must not stop after the output is reached. Here the systems must continuously coordinate and synchronize with each other such that no deadlocks or any conflicts occur, here election algorithm come into place in electing a leader and maintaining the proper functioning of the system.

The goals or major benefits of distributed computing are given below:

•Resource sharing

•Scalability

•Fault tolerance

•Availability

1.1 Need for a coordinator
Many algorithms used in distributed systems require a coordinator to grant permission to access critical section and to manage the nodes in the distributed system .In general; all processes in the distributed system are equally suitable for the role Election algorithms are designed to choose a coordinator.

1.2 Mutual Exclusion
One of the important problems in distributed systems is mutual exclusion. The mutual exclusion problem states that only a single process is allowed to access a protected resource, also termed as a critical section (CS), at any point of time. One of the approaches for solving this problem is centralized algorithm. In this approach one process is elected as the coordinator (e.g., the one running on the machine with the highest network address). Whenever a process wants to enter a critical section, it sends a request message to the coordinator and asking for permission. If no process is currently in the critical section, the coordinator sends back a reply granting permission and when the reply arrives, the requesting process enters the critical section. Providing access to this critical section is the purpose of the coordinator node and it will be further emphasized in remainder of the paper.

1.3 Elections in Distributed Systems
In a distributed system, when the leader is crashed, other nodes must elect another leader. The election algorithm we consider here is called the bully algorithm because the node with the highest ID forces the nodes with smaller ID into accepting it as a coordinator. In bully algorithm when the node N understands the leader is crashed, sends an election message to all nodes with higher numbers. If no one responds, N wins the election and becomes the leader. If one of the higher IDs answers, it takes over. N's job is done. When such a message arrives, the receiver sends an OK message back to the sender to indicate that he is alive and will take over. The receiver then holds an election, unless it is already holding one. Eventually, all nodes give up but one and that one is the new leader. Classic example for leader election is the bully election algorithm which is prevalently used to elect leaders in a distributed system. Also for electing the leader some distributed networks make use of the Ring Election Algorithm, but we will be focusing on reducing the number of messages in the bully algorithm.

1.4 Bully Algorithm

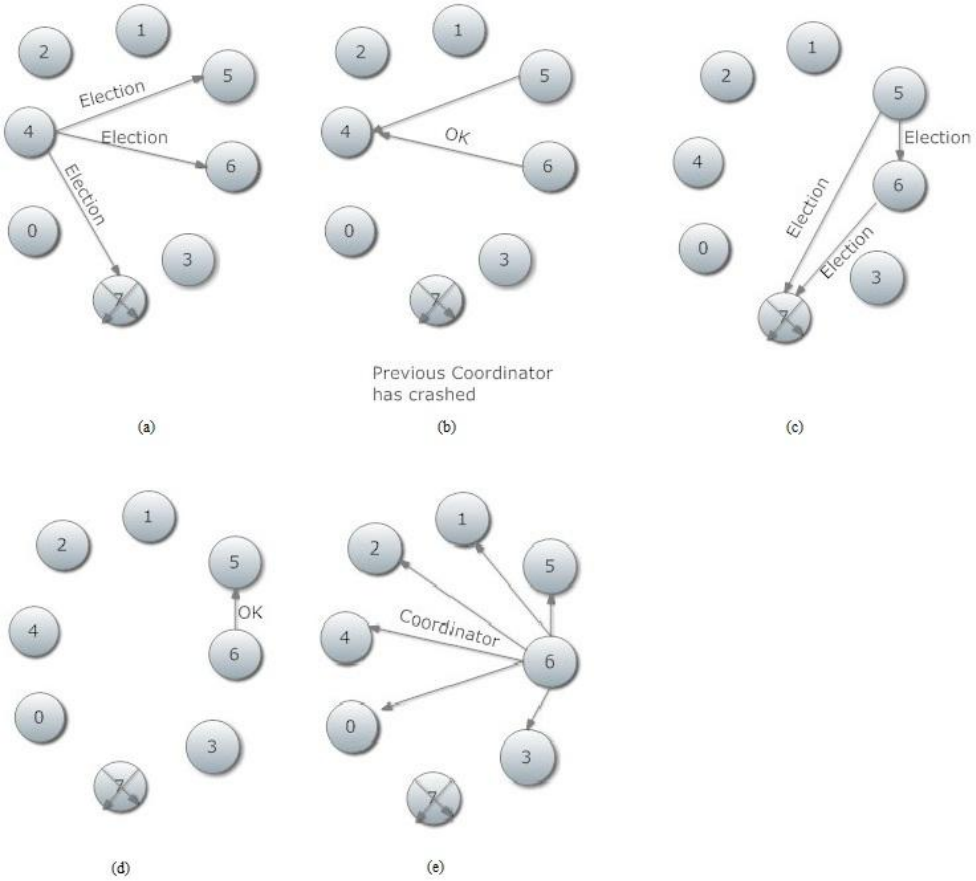


Fig 1: Working of Bully Algorithm

Process p calls an election when it notices that the coordinator is no longer responding. High-numbered processes "bully" low-numbered processes out of the election, until only one process remains. When a crashed process reboots, it holds an election. If it is now the highest-numbered live process, it will win. Process p sends an election message to all higher-numbered processes in the system. If no process responds, then p becomes the coordinator. If a higher-level process responds, it sends p a message that terminates p's role in the algorithm.

If a process receives an Election message:

Immediately sends Coordinator message if it is the process with highest ID , Otherwise, returns an OK and starts an election .If a process receives a Coordinator message, it treats sender as the coordinator

2. DRAWBACKS OF BULLY ALGORITHM

There are two major drawbacks of the bully election algorithm; they are increased message passing and no method for recovery.

2.1 Increase in message passing

This simple idea has a big problem that is the high number of messages that should be exchanged between processes. Therefore this approach imposes heavy traffic in the network. In order to solve this problem, we will present optimized method by modifying the bully algorithm that decreases the number of messages that should be exchanged between processes.

We aim to modify the bully algorithm in order to reduce the number of messages to reduce the number of messages exchanged to find a new coordinator, each process that notices the failure of the coordinator, attempts to run the bully algorithm and whatever the ID (identifier) of this process is lower, more messages should be exchanged, until a process with the largest ID is found and introduced as a new coordinator. Number of messages is calculated by the following equation, in this regard n is the number of processes in the distributed system and ID is the identifier of the process that has noticed a coordinator crash and run the bully algorithm.

2.2 No method for recovery

The bully election algorithm does not say what to do when a crashed leader recovers. Not only bully election algorithm, all the election algorithm does not care about the crashed leader, when the leader is crashed new leader is elected and the crashed leader is not considered for the new election. By some means if the crashed leader recovers itself , we need to include it also in the election .This is a major drawback in the bully election algorithm and even other election algorithms do not address this

issue. We are suggesting an alternate method to overcome this drawback.

3. PROPOSED METHOD

3.1 Improved Bully election algorithm

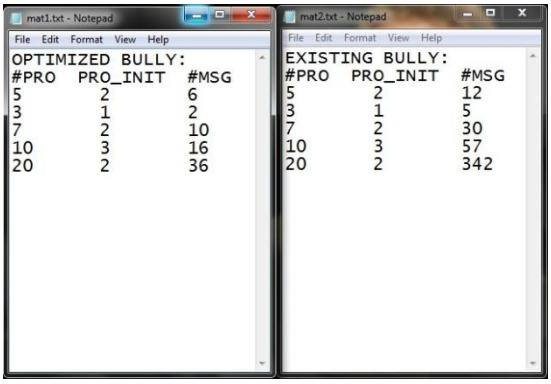
Section -1

In the proposed method, when a process demands to enter the critical region, sends a message to coordinator, the coordinator responds the applicant process based on the situation of the critical area, whether it is free or not. Here, coordinator accomplishes an addition task too, it should register the ID of the applicant process, so over time, a list of the applicants process's ID is created with the coordinator. This list represents the coordinator information from the ID of the processes presents in the system. With increasing knowledge and get the ID of applicant processes, the coordinator frequently sends a message containing the biggest ID in the list for the processes to inform them that there is a process with this ID in the list, provided that in the interval between two posts, there is an applicant process that applies to enter the critical section and its ID is bigger than the biggest ID in the list. If the coordinator is failed, each process that notices this failure compares its ID with the ID which it has received via the coordinator.

If the received coordinator id is bigger than the current coordinator, since it knows the next coordinator the node sends out election message to the next possible coordinator node. If the next possible node is alive it receives the message and it sends the coordinator message to all its other nodes bullying to accept it as the new coordinator. In other case, the received coordinator id may be less than the current coordinator; here it follows the procedure as of the traditional bully algorithm. In most of the cases, it is going to be case 1 that is the number of messages will be considerably reduced than the existing bully algorithm.

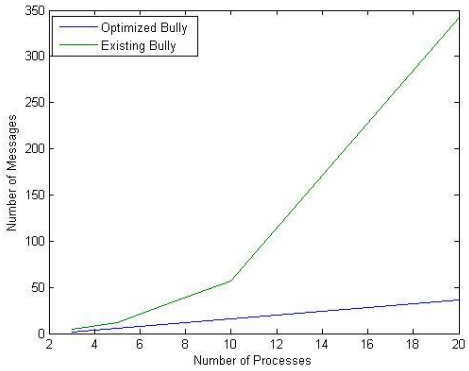
We have simulated the existing bully algorithm and our modified bully algorithm which sends out the next possible leader message to the remaining nodes in C programming language. And our results have shown a remarkable amount of decrease in the number of messages needed to elect the leader.

Below are figures that show the number of messages passed and the total number of process involved in the election.



[Output written out to a file showing the number of messages]

The output value shows the efficiency increases with the increase in the number of messages .Also we have plotted the graph for the output values for the total number of messages against the number of processes and have simulated results from matlab showing considerable increase in efficiency for our algorithm.



[Graph comparing the number of messages between the two algorithms]

Section – 2

Now we have addressed the issue of message passing, next issue to be addressed is what to do when a crashed leader recovers.

In our proposed method, the newly elected leader in spite of sending messages to the other nodes it must periodically send alive message to the crashed leader to see if it had recovered. When it finds that the crashed leader had recovered, it gives the control back to the old leader and it now takes the job of coordinator in the distributed system which is preferable than the existing leader which took over the job of the leader.

The existing bully algorithm does not satisfy this and conducts election again when the dead leader has recovered back, in spite of just transferring the control to the highest possible leader which will be achieved in the new method.

4. CONCLUSION

One of the problems in the centralized algorithms for solving mutual exclusion is finding a new coordinator when –the current coordinator crashes. Bully algorithm imposes a big message passing overhead in order to find the successor coordinator and this causes reduction in the performance. In this algorithm, after the breakdown of the coordinator, a process that first noticed the coordinator failure, attempts to send a message to all processes with bigger ID and if it does not receive an answer, it becomes the coordinator itself otherwise, the process with the bigger ID that has received this message repeats this action.

Whatever the ID of the process which notices the coordinator failure is smaller, the process ID so that the coordinator has noticed failure is smaller, amount of message passing would be more. The proposed algorithm is trying to improve the bully algorithm, so that the ID of the process which runs this algorithm be larger, therefore, in contrary to the bully algorithm that only runs when the coordinator is failed, it runs on the coordinator during its life time. In this algorithm, the coordinator during its lifetime tries to use the incoming messages to recognize

the process with the biggest ID and announces its ID to all processes. So it periodically sends the biggest known ID in the system for existing processes, then the first process that notices the coordinator failure, sends a message to a process with biggest ID and asks it to run the bully algorithm. So because the process runs the bully algorithm is probably the same process with the biggest ID, the amount the amount of message passing is minimized. Number of exchanged messaged is never greater than the number of messages which are exchanged during the traditional bully algorithm in which the messages passed is always high than the modified algorithm even after including the overhead which occurs due to the information sent by the leader to the nodes regarding the maximum available ID to the nodes of the distributed system.

5. FUTURE WORK

The algorithm is implemented for the best case, provided that there won't be any inconsistencies in the system. This can be expanded to all the cases for example, the next highest coordinator may not be in a position to accept the new coordinator request, or the new coordinator node is not willing to take on the responsibility of a coordinator, in these cases the above proposed algorithm cannot hold well. The cases must be considered and we are planning to make the algorithm run on all cases and conditions.

6. REFERENCES

[1] Arghavani . A E.Ahmadi A.T.Haghighat Improved Bully Election Algorithm in Distributed Systems

[2] Fredrickson .N and Lynch .N., 1 9 8 7 . Electing a Leader in a Synchronous Ring." J.ACM, vol.34, no.1, pp.98-115.

[3] Garcia-Molina, H., "Elections in Distributed Computing System," IEEE Transaction Computers, Vol.C-1,pp.48-59,Jan.1982.

[4] Kim, J. L. and Belford, Geneva G., 1988. A robust, distributed election protocol", Proc. of seventh IEEE Computer Soc. Symp. Reliable Distributed Systems, pp. 54-60, Columbus, Ohio.

[5] Le Lann, G., "Distributed Systems – Towards a Formal Approach", in Information Processing 77, B. Gilchrist, Ed. Amsterdam, The Netherlands: North-Holland, pp. 155-160, 1977.

[6] Park S , Y. Kim and Hwang J.S., 1999. An Efficient Algorithm for Leader-Election in Synchronous Distributed Systems, IEEE TENCON

[7] Renu Nekkanti and Aruna Kumari G.L. ELECTION ALGORITHM WHEN CRASH LEADER RECOVERS IN DISTRIBUTED SYSTEMS

[8] Singh, S., Kurose, J.F., "Electing 'good' leaders (election leader algorithm)," Journal of Parallel and Distributed Computing, Vol. 21, No. 2, pp. 184-201. May 1994.

[9] Sung-Hoon-Park, Yoon Kim, And Jeoung Sun Hwang "An Efficient Algorithm for Leader-Election in Synchronous Distributed Systems." IEEE Transaction on Computers, vol. 43, no. 7, pp.1991-1994, 1999.

[10] Tanenbaum, A.S., and Steen M.V.: "Distributed Systems Principles and Paradigms," Prentice-Hall International, Inc, 2002.