International Journal on Emerging Technologies **1**(2): 10-14(2010) **ISSN : 0975-8364**

A new approach of bully election algorithm for distributed computing

Pawan Kumar Thakur, Ram Kumar, Roohi Ali and Rajendra Kumar Malviya

Department of M.C.A., Govt. Geetanjali Girls. P.G. College, Bhopal (M.P.) INDIA

ABSTRACT : Many distributed algorithms require one process to act as a unique process to play a particular role in distributed systems. Election algorithms are meant for electing such a process called coordinator from among the currently running processes of distributed systems in such a manner that at any instance of time there will be a single coordinator for all the processes in the system. So, election algorithms are extremely crucial in any distributed system. Bully algorithm is one of the classical approaches for electing the coordinator in distributed systems. In this paper, we have presented an efficient version of bully algorithm to minimize the redundancy in electing the coordinator, to reduce the recovery problem of a crashed process in distributed systems and thus to maximize the effectiveness of traditional bully algorithm.

Keywords : Bully Algorithm, Election, Coordinator, Process Number, Turnaround Time, Election Message, Ok Message, Coordinator Message, Query Message, Answer Message.

I. INTRODUCTION

Several distributed algorithms require that there be a unique coordinator process in the entire system that performs some type of coordination activity needed for the smooth running of other processes in the system. In general, it does not matter which process takes on this special responsibility, but one of them has to do it. If all the processes are exactly the same, with no distinguishing characteristics, there is no way to select one of them to be special [1]. Coordinator is one of the processes with a distinguishable process number. So, electing coordinator plays a vital role in distributed systems.

II. ELECTION ALGORITHM

An algorithm for choosing a coordinator to play a distinct role is called an election algorithm. For example, in a variant of 'central-server' algorithm for mutual exclusion, the 'server' is chosen from among the processes pi, i = 1, 2..., N that need to use the critical section. An election algorithm is needed to choose which of the processes will play the role of server. Since all other processes in the system have to interact with the coordinator, it is essential that they all must unanimously agree on who the coordinator is. If the coordinator process wishes to retire or fails due to the failure of the site on which it is located, then another election is required to choose a replacement and a new process must be elected as coordinator to take up the job of the failed coordinator [3].

We say that a process calls the election if it takes an action that initiates a particular run of the election algorithm. An individual process does not call more than one election at a time, but in principle the N processes could call N concurrent elections. At any point in time, a process pi, is either a participant meaning that it is engaged in some run of the election algorithm — or a non-participant — meaning that it is not currently engaged in any election [3].

An important requirement is for the choice of elected process to be unique, even if several processes call elections concurrently. For example, two decide independently that a coordinator process has failed, and both call elections [3].

Without loss of generality, we require that the elected process be chosen with the largest process number. The 'process number' may be any useful value, process numbers are unique and totally ordered.

III. BULLY ALGORITHM

The bully algorithm [Garcia-Molina 1982] allows processes to crash during an election, although it assumes that message delivery between processes is reliable.

Algorithm: This algorithm is based on the following assumptions:

(*a*) The system is synchronous: it uses timeouts to detect a process failure [3].

(*b*) Each process in the system has a unique process number [1].

(*c*) Every process knows the process number of every other process and which processes have higher numbers and communicate with all such processes [3]. What the processes do not know is which ones are currently up and which ones are down currently.

(*d*) Whenever an election is held, the process having the highest process number among the currently processes alive is elected as the coordinator [2].

(*e*) On recovery, a failed process can take some actions to rejoin the set of active processes [2].

(*f*) There are three types of message in this algorithm. An election message is sent to announce an election; an answer message is sent in response to an election message; and a coordinator message is

sent to announce the number of the elected process – the new 'coordinator'. A process begins an election when it notices, through timeouts, that the coordinator has failed. Several processes may discover this concurrently [3].

Operation: The operation of this algorithm is shown in Fig. 1.

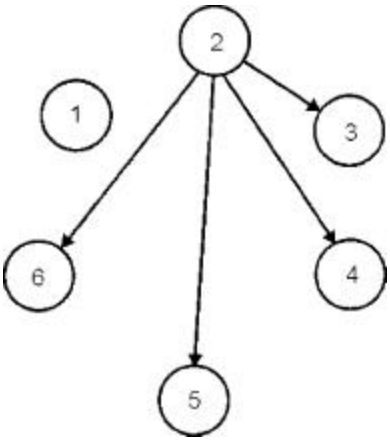

Fig. 1(*a*)

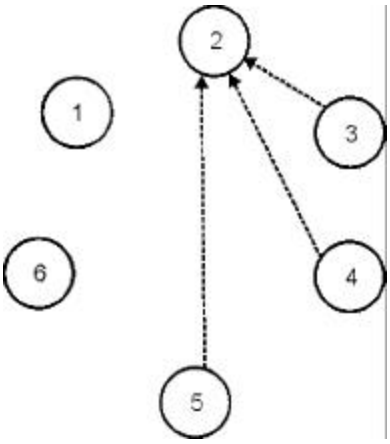

Fig. 1(*b*)

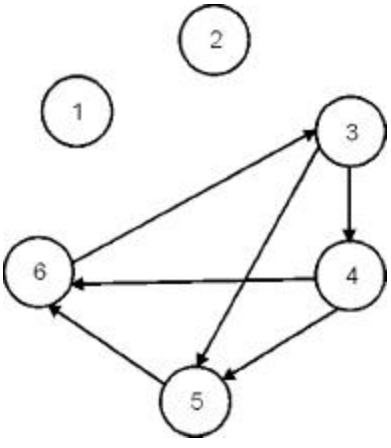

Fig. 1(*c*)

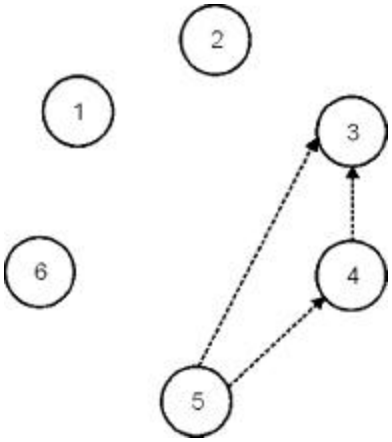

Fig. 1(*d*)

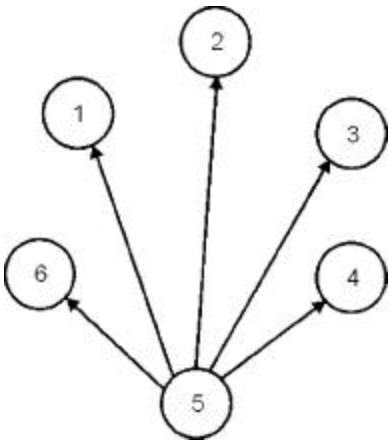

Fig. 1(*e*)

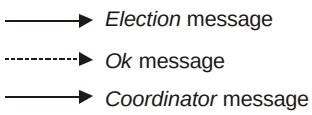

Fig. 1 : The bully election algorithm

Here, the system consists of six processes, numbered 1 to 6. Suppose all of the six processes are alive except process 1. Now process 6 is the coordinator, but it has just crashed. Process 2 is the first one to notice this, so it sends election messages to all the processes higher than it, namely processes 3, 4, 5 and 6, as shown in Fig. 1(a). Processes 3, 4 and 5 all response with ok message, as shown in Fig. 1(b). Upon getting the first of these responses, process 2 knows that its job is over. It just sits back and waits to see who the coordinator will be. In Fig. 1(c), processes 3, 4 and 5 hold elections, each one only sending message to those processes higher than itself. In Fig. 1(d), process 4 sends ok message to process 3 and similarly process 5 sends ok message to process 3 and process 4. Process 5 will not receive any ok message from any other processes. Thus process 5 will be the new coordinator and sends coordinator messages to all

processes as shown in Fig. 1(e). Suppose now process 5 is crashed.

Process 2 has noticed this and initiated the election. Thus process 4 will be the new coordinator in a similar way as depicted in Fig. 1. At this moment of time, process 1 recovers from failure and initiates an election by sending messages to all other processes. Thus process 4 will be elected as coordinator again. Now process 6 recovers from failure and will send coordinator messages to all processes and will bully them into submission. So process 6 is the new coordinator. After this, if process 5 recovers from failure, it will also send coordinator messages to all processes and will bully them into submission.

Advantages: This algorithm has following advantages:

(*a*) This algorithm clearly meets the live ness condition E2, by the assumption of message delivery. And if no process is replaced, then the algorithm meets condition E1.

(*b*) It is impossible for two processes to decide that they are the coordinator, since the process with the lower number will discover that the other exists and defer to it.

Limitations: The bully algorithm suffers from the following shortcomings:

(*a*) This algorithm is not guaranteed to meet the safety condition El if processes that have crashed are replaced by processes with the same number. A process that replaces a crashed process p may decide that it has the highest number just as another process (which has detected p's crash) has decided that it has the highest number. Two processes will announce themselves as the coordinator concurrently. Unfortunately, there are no guarantees on message delivery order, and the recipients of these messages may reach different conclusions on which is the coordinator process [3].

(*b*) Condition El may be broken if the assumed timeout values turn out to be accurate — that is, if the processes' failure detector is unreliable [3].

(*c*) Taking the example just given, suppose that process 5 either had not failed bit running unusually slowly (that is, the assumption that the system is synchronous incorrect) or that process5 had failed but is then replaced. Just as process 4 sends its coordinator message, process 5 (or its replacement) does the same. Process 4 receives process 5's coordinator message after it sent its own and so sets elected4 = process 5. Due to variable message transmission delays, process 3 receives process 4's coordinator message after process 5's and so eventually elected3 = process 4. Condition El has been broken.

(*d*) From the above example, we can find out some other drawbacks:

- When process 5 recovers from failure, it becomes new coordinator whereas process 6 with the highest process number is still alive. This is a violation of the assumption.
- At the time, when process 4 is coordinator, process 1 recovers from failure and it initiates an election. Then process 4 becomes coordinator again. This election is redundant.
- When process 2 notices the failure of the coordinator process 6, it sends election messages to processes 3, 4, 5 and 6. In reply of the election messages, it gets ok messages from processes alive among processes 3, 4, 5 and 6. Process 2 can elect the coordinator itself instead of holding elections by processes 3, 4 and 5. These elections are also redundant.

IV. PROPOSED BULLY ALGORITHM NEW APPROACH

In this section, we have presented a modified version of existing bully algorithm.

Algorithm: Our algorithm is based on the assumptions of the existing bully algorithm.

(*a*) There are five types of message in this algorithm. An election message is sent to announce an election; an ok message is sent in response to an election message on recovery, a process sends a query message to the processes with process number higher than it to know who the new coordinator is; a process gets an answer message from any process numbered higher than it in response to a query message; and a coordinator message is sent to announce the number of the elected process – the new 'coordinator'.

(*b*) Since the system is assumed to be synchronous, we will construct a reliable failure detector and calculate a time T as upper bound on the total elapsed time from sending a message to another process to receiving a response as described in [3].

(*c*) When a process notices that the coordinator is no longer responding, it initiates an election. A process P, holds an election as follows:

- P sends an election message to all processes with higher numbers.
- If no one response, P wins the election and becomes the coordinator.
- If process P gets responses (ok message along with process number of the responder) from the

processes containing higher process number than that of process P, it selects the process with the highest process number as new coordinator and sends coordination messages to all other processes.

At any moment, a process can get election message from one of its lowered numbered colleagues. When such a message arrives, the receiver sends an ok message along with process number of itself back to the sender to indicate that it is alive and is a potential candidate for the coordinator.

If a process that was previously down comes back up, it sends a query message to the processes with process number higher than it to know who the new coordinator is. If it gets any answer message from any processes numbered higher than it, it knows the new coordinator. If it gets no answer message from any processes numbered higher than it within time T, it knows there is no other higher numbered process alive. The process then sends coordination message to all processes with lower numbers and becomes the new coordinator.

Operation: The operation of this algorithm is shown in Fig. 2 :

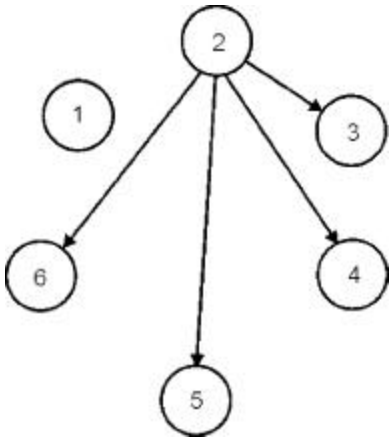

Fig. 2(*a*)

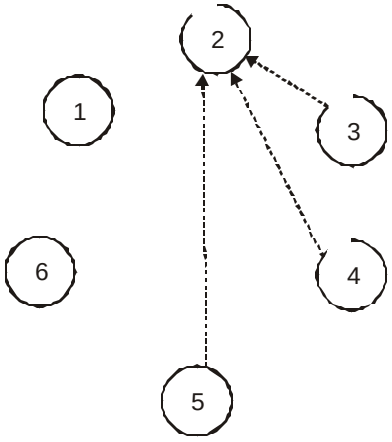

Fig. 2(*b*)

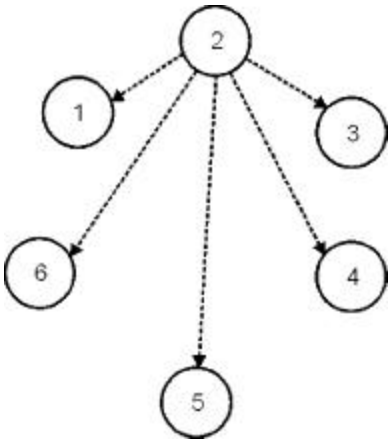

Fig. 2(*c*)

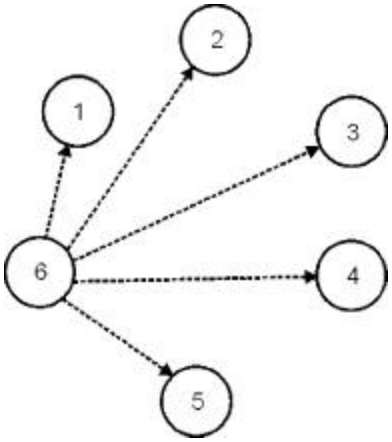

Fig. 2(*d*)

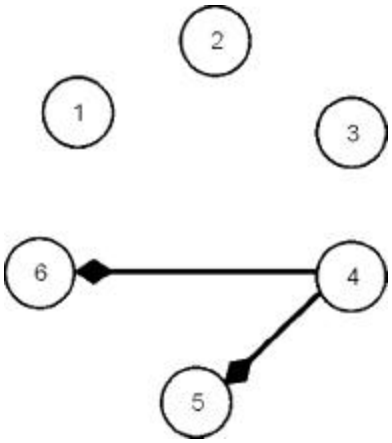

Fig. 2(*e*)

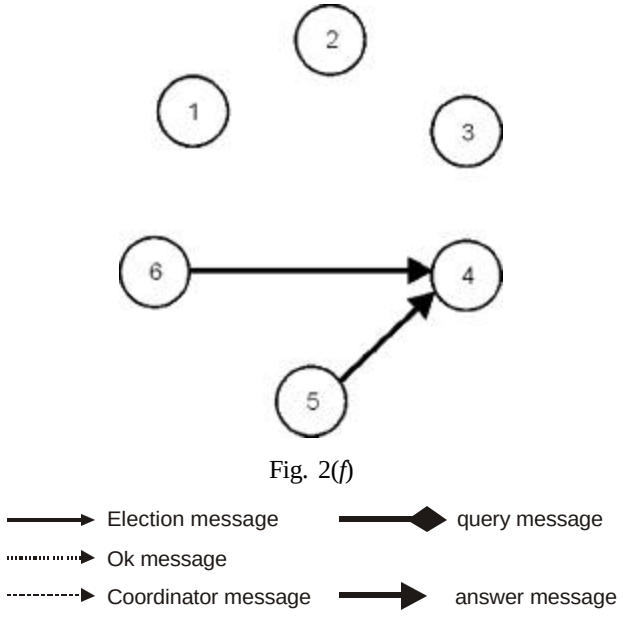

Fig. 2(*f*)

Fig. 2 : The modified bully algorithm

Here also, the system consists of six processes, numbered 1 to 6. Now process 6 is the coordinator, but it has just crashed. Process 2 is the first one to notice this, so it sends election messages to all the processes higher than it, namely processes 3, 4, 5 and 6, as shown in Fig. 2(a). Processes 3, 4 and 5 all response with ok message, as shown in Fig. 2(b). Process 2 now knows who the alive process with highest process number is. So it elects process 5 as the new coordinator and sends coordinator messages to all other processes, as shown in Fig. 2(c). The election is finished at this point. Every process knows process 5 as the new coordinator. Now process 4 has just crashed and process 6 has recovered from failure. As process 6 knows that it is the process with highest process number, it just sends coordinator message to all processes as shown in Fig. 2(d) and becomes the new coordinator. Suppose process 4 has recovered from failure and sends query messages to process 5 and process 6 instead of holding an election, as shown in Fig. 2(e). In Fig. 2(f), process 4 gets answer message from process 5 and process 6 in response of its query message and process 4 comes to know process 6 as the new coordinator.

V. CONCLUSION

We measure the performance of an election algorithm by its total bandwidth utilization and by the turnaround time for the algorithm: the number of serialized message transmission times between the initiation and termination of a single run [3]. From this context, our modified bully algorithm as compared the existing one is efficient and easier to implement in all cases. But still there is a future scope to improve this algorithm so that it can be guaranteed to meet the safety condition E1 with out any compromise with the efficiency of electing the coordinator in distributed system.

VI. ACKNOWLEDGEMENT

The authors wish to acknowledge, HOD ***Dr. Sanjay Bhatt***, Department of MCA for providing the distributed computing facilities.

REFERENCES

[1] Tanenbaum, A. S., *Distributed Operating Systems*, Pearson Education (Singapore) Pte. Ltd., 140– 142 (2002).

[2] Sinha, P. K., *Distributed Operating Systems Concepts and Design*, Prentice Hall of India Private Limited, 332–334 (March 2002).

[3] Coulouris, G., Dollimore, J., Kindberg, T., *Distributed Systems Concepts and Design*, Pearson Education, 431–436 (2003).

[4] Tanenbaum, A. S., Steen, M. V., *Distributed Systems Principles and Paradigms*, Prentice-Hall of India Private Limited, 262–263 (July 2003).

[5] Garg, V. K., *Principles of Distributed System*, Kluwer Academic, Norwell, MA (1996).

[6] Garcia-Molina, H., "Elections in a Distributed Computing Systems", *IEEE Transactions on Computers*, **C–13**(1)**:** 48 – 59 (1982)