The Reduction of Number Messages in Election Bully Algorithm

Qamil Kabashi, Arbnor Zeqiri and Milaim Zabeli

***Abstract*—**In distributed systems, communication networks, and general communication between processes, it is required to have a process that synchronizes all other system processes and communication between them.
If the chosen coordinator crashes or becomes isolated, a new coordinator is elected. All active processes at any given point of election get together to choose a coordinator.
Nowadays, many algorithms have been developed for the election of the coordinator. These algorithms differ among themselves for the way in which they select the coordinator, the criteria that are taken into account for selection, and the number of messages required for selection. Among the best known of such algorithms is the Bully Algorithm and its modifications.
The proposed algorithm is also based on the Bully Algorithm, but unlike similar algorithms, it will select the process with the smallest identifier as a coordinator, assuming that the minimal identifier that a process can take is known. This algorithm reduces the number of messages (in the best case only one message is required), as well as network traffic, and it ensures that the system has only one coordinator at any given time.

***Keywords*—**Coordinator, Distributed Systems, Election Algorithms, Election, Liveness, Synchronous Distributed Systems.

I. Introduction

IN distributed systems, if two or more processes send messages at the same time, their messages will collide and they will not arrive at their destination. Therefore, among these processes, there must exist a process that enables coordination between them and initiates certain tasks in the system. This process is known as the coordinator (leader). In general, it does not matter which process takes on this special responsibility, but one of them has to do it [1].

Designating a process as the coordinator in distributed systems is a challenging issue that requires special algorithms. To determine which process will take the role of coordinator, different algorithms, known as election algorithms, have been developed. These election algorithms are needed in two cases:

- When the system starts.
- When the current coordinator fails or leaves the system [1], [2].

In every election algorithm, the coordinator is selected based on two basic criteria:

- Process identifier.
- Process availability.

Each election algorithm must satisfy safety and liveness conditions.

When the status of any process *i* is set to leader/coordinator, it is understandable that the certain process (*i*) has fulfilled the liveness condition, and also that there is no another process with the same status (which satisfies the safety condition) [3], [4].

When an election is initiated, every process enters the election procedure in either a state of non-participation or one of active participation. Once a process enters into a particular state, it remains in that state until the end of the election.

Information is exchanged between processes by transmitting messages to one another until an agreement is reached. Once a decision is made, a process is elected as the coordinator and all the other processes will acknowledge the role of that process as the coordinator [5].

Once the coordinator is selected, the processes reach a state known as a terminated state [3].

Nowadays, there are many election algorithms, such as: the Bully Algorithm - designed by Hector Garcia Molina in 1982 [6], the Enhanced Bully Algorithm for leader process election in synchronous distributed systems [7], the Ring Based Algorithm, proposed by Silberschatz and Gavin [8], [9], [10] the Modified Bully Algorithm using election commission [11], the Change-Roberts Algorithm [12], the Peterson Algorithm [13], the Franklin Algorithm [14], etc.

In this paper, we have proposed an algorithm, which is a modification of the Bully Algorithm. The proposed algorithm selects the process with the smallest identifier as a coordinator and reduces the number of required messages.

II. Bully Algorithm

When any process notices that the coordinator is no longer responding to requests, it initiates an election [1].
This algorithm is based on these assumptions:

- Each process knows the priority of other processes in the system.
- The communication subsystem does not fail [6], which means that the communication infrastructure is stable.
- A process never pauses and always responds to incoming messages with no delays [1], [6].
- There are no transmission errors [6].
- Whenever the selection of the coordinator is made, it is ensured that the process with the highest identifier will be selected as the coordinator [1].
- A new process or one that failed earlier may join in the system.

The algorithm follows the following procedure:

1) The process *P* sends an *ELECTION* message to all processes with higher numbers and waits for responses.
2) If no one responds, *P* wins the election and becomes coordinator.

3) If one process, *Q*, with a higher number answers with *OK*, *P*'s job is done and the process *Q* will continue the election procedure.

The number of required messages for determining the coordinator process, is calculated by [15], [16], using the following formula:

$$T_m = (N - P + 1)(N - P) + N - 1 \quad (1)$$

where:
T_m – the number of exchanged messages between processes when process *P* detects failure of the coordinator,
N – the total number of processes,
P – the ID of the process that detects the failure of the coordinator.

If a process that was previously down comes back up, it initiates an election procedure. If it happens to be the highest-numbered process currently running, it will win the election and take over the coordinator's job. Thus the biggest guy in town always wins, hence the name "bully algorithm" [1].

A. Disadvantages of Bully Algorithm

Some of the disadvantages of the Bully Algorithm are:

- Large number of messages:
  - Best case: $O(n) = n - 1$
  - Worst case: $O(n^2) = n^2 - 1$
  - When a new process joins the system: $O(n^2) = n^2 - 1$
- There is no mechanism that ensures that the system has only one coordinator.
- When a new process joins a new election, the procedure must start.

III. Improved Bully Election Algorithm for Synchronous Distributed Systems

Md. Golam Murshed and Alastair R. Allen in [7] have made some improvements to the Bully Algorithm.

In their algorithm the processes are divided in two sets: N/2 *Candidate* processes and N/2 *Ordinary* processes, where N is the total number of processes. Any *Candidate* process has a higher identifier than any *Ordinary* process.

The algorithm follows the following procedure:

1) A process detects the failure of the coordinator:
   a. If it belongs to the *Ordinary* set it sends the *election* message to *Candidate* processes and waits to receive an *OK* message. If it does not receive any answer from any *Candidate* process, then it sends the *election* message to the *Ordinary* processes that have higher IDs.
   b. If it belongs to the *Candidate* set it sends an *election* message to the *Candidate* processes that have higher IDs.

The *election* message contains the ID of the failure detector process and the ID of the failure coordinator. The respective *answer* message contains the process IDs of the leader and *Candidate* set.

2) When a process receives the *election* message, it answers with an *OK* message and attaches its ID to it.
3) The electioneer process, after receiving all *OK* messages, selects the process with the highest ID as the coordinator and sends the *coordinator* message, to which it attaches the ID of the coordinator and the *Candidate* set, to all processes.
4) A new process joins the system:
   a. If it belongs to the *Ordinary* set it sends a *query* message to the *Candidate* processes; otherwise, if it belongs to the *Candidate* set, it sends a *query* message to the *Candidate* processes that have higher IDs.
   b. When a process receives a *query* message, it answers with an *answer* message, to which it attaches the ID of the coordinator and the *Candidate* set.
   c. If the process which receives the *answer* message has a higher ID than the ID of the current coordinator, it initiates a new election procedure.

A. Disadvantages of Improved Bully Algorithm

As the original Bully Algorithm, this improved algorithm also has some disadvantages:

- It is a complex algorithm
- Large number of required messages:
  - Best case: $O(n)$
  - Worst case: $O(n^2)$
- When a new process joins the system the set cardinality must be rearranged.
- When a new process joins a new election, the election procedure must start.

IV. The Proposed Algorithm

The algorithms discussed above choose the process with the highest ID as the coordinator. The algorithm that we propose chooses the process with the smallest ID as the coordinator.

Our proposed algorithm significantly improves the number of messages required to elect a coordinator and also ensures that the system has only one coordinator at any given time.

A. The Algorithm Procedure

Our proposed algorithm assumes that the minimum ID that a process may take is known.

The algorithm follows the following procedure:

1) When process P_i notices a failure of the coordinator, then:
   a. If P_i has the second minimum ID, it sends the *COORDINATORi* message (where *i* is the ID of P_i) to notify the other processes that P_i is now the coordinator (Fig. 1.a).
   b. If P_i does not have the second minimum ID, it sends the *ELECTION* message to all processes (broadcasts) and waits for responses.
2) If process P_i does not receive any responses, then it becomes the coordinator and sends *COORDINATORi* message, where *i* is the ID of the P_i.
3) If after P_j receives the *ELECTION* message from P_i, it determines that it has a smaller ID than the P_i, then:
   a. If it has the second minimum ID, it sends a $COORDINATOR_j$ message (Fig. 1.b).
   b. If it does not have the second minimum ID, it sends an OK_j message (Fig. 1.c).

   Where *j* is ID of the process P_j.

4) The process *Pi*, after receiving the responses from all processes with smaller ID, selects the process with the smallest ID P_k as the coordinator and sends a $COORDINATOR_k$ message to all processes, where k is the ID of the coordinator (Fig. 1.c).

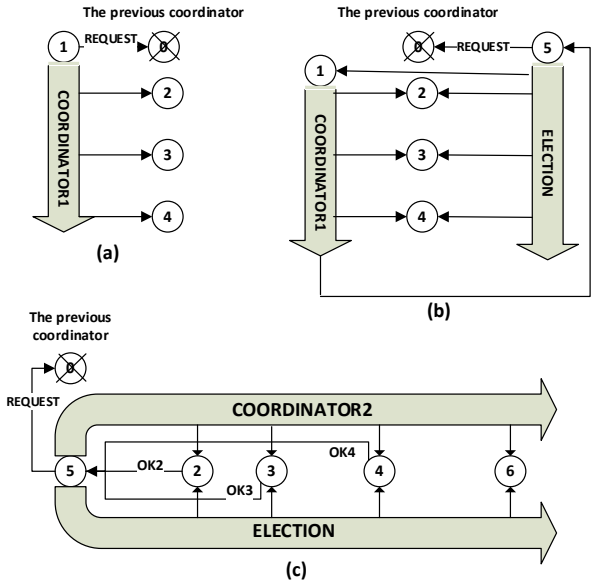

Fig. 1. Electing the new coordinator in case of failure of the previous coordinator: (a) The case when the process with the second minimum ID detects the failure of the coordinator, (b) The case when the process with the second minimum ID receives the *ELECTION* message and sends a *COORDINATORi* message, (c) The case when process 5 selects the new coordinator.

5) If a process P_x joins the system, then:
   a. It has the minimum ID, it sends a *COORDINATORx* message to all processes to notify them that it is the new coordinator, where x is the ID of P_x (Fig. 2.a).
   b. Otherwise, it sends a *QUERY* message to all active processes (broadcasts). The coordinator answers with *CIDc* message, where c is the ID of the coordinator. If the ID of P_x is smaller than c, then it becomes the coordinator and it sends the *COORDINATORx* message (Fig. 2.b).

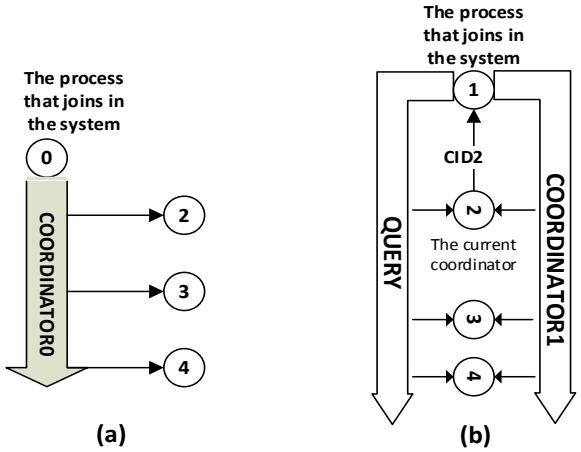

Fig. 2. The case when a new process joins the system: (a) The case when it has the minimum ID, (b) The case when it has a smaller ID than the current coordinator.

6) When process P_w receives the *COORDINATORk* message, it checks if the ID of the new coordinator is higher than its own ID. If it is true, then it sends a *COORDINATORw* message to the all processes to notify them that now P_w is the new coordinator, where w is the ID of P_w (Fig. 3). This ensures that we have only one coordinator at any given time.

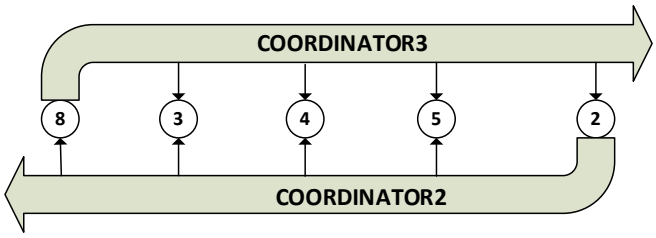

Fig. 3. The case when process 2 receives the *COORDINATOR3* message with a higher ID.

V. Pseudocode

The pseudocode for every step of proposed algorithm is shown below.

```
//The process Pi with ID i detects the
//failure of the coordinator
procedure failureDetection
   if (Pi has the second minimum id)
then
      broadcast(coordinatori)
   else
      broadcast(election)
      wait for responses
      if (any ok message received)then
         find smallest id c from the
      responses
         broadcast(coordinatorc)
      else
      broadcast(coordinatori)
  end if
   end if
end procedure

//The process Pw with ID w receives
//COORDINATORk message
procedure coordinatorMessageReceived
   if(w is lower than k) then
      broadcast(coordinatorw)
   end if
end procedure
//The process Pj with ID j receives an
//election message from the process Pi
//with ID i
procedure electionMessageReceived
   if(j is the second minimum id) then
      broadcast(coordinatorj)
   else
      if(j is lower than i) then
         send(OKi)
      end if
   end if
end procedure
```

```
//The new process Px with ID x joins the
//system
procedure joiningTheSystem
   broadcast(query)
   wait for response
   if(x is lower than id of the
coordinator) then
      broadcast(coordinatorx)
   end if
end procedure

//The process receives query message
procedure queryMessageReceived
   if(The process with ID c is the
coordinator)then
      answer with (CiDc) message
   end if
end procedure
```

VI. Findings and comparison

Based on the proposed algorithm, we see that processes with the smallest identifier are more favorable to be elected as the coordinator, because:

- The process with the minimum ID does not start an election procedure, but automatically becomes the coordinator.
- When the coordinator fails, the process with the second minimum ID does not start an election procedure, but automatically becomes the coordinator.
- It reduces the number of messages required:
  - Best case: 1
  - Worst case:

$$2b + (n-1) \quad (2)$$

where, b is the number of broadcast messages, n is the total number of processes

  - In the case when a new process joins the system:
    - Best case: 1
    - Worst case: 3
- The processes do not need to know the ID of any process.
- It is suitable for different process topologies.
- There is no need to start a new election when a new process joins the system.
- There is no need to update the minimum ID that the process may take.

Table I and Table II show a comparison between the number of messages required for the proposed algorithm and the number of messages required for the other two algorithms reviewed before.

Table I. Comparison of the number of messages required for the coordinator election in the worst case.

Number of processes	Number of messages required in the Bully Algorithm	Number of messages required in the Improved Bully Election Algorithm for Synchronous Distributed Systems [7]	Number of messages required in the proposed algorithm
5	24	9	6
10	99	18	11
20	399	38	21

Table II. Comparison of the number of messages required for the coordinator election in the best case.

Number of processes	Number of messages required in the Bully Algorithm	Number of messages required in the Improved Bully Election Algorithm for Synchronous Distributed Systems [7]	Number of messages required in the proposed algorithm
5	4	4	1
10	9	9	1
20	19	19	1

VII. Conclusion

If a process has the minimum ID, it does not need to initiate an election procedure. According to the rules of the proposed algorithm, it is the process which will be the coordinator.

If a process that has the second minimum ID detects the failure of the coordinator or receives an *ELECTION* message, it immediately becomes the coordinator and sends a *COORDINATOR* message, with its ID attached, to all other processes.

Our proposed algorithm is:

- Simple
  - The first two processes do not initiate an election process.
  - Unlike the Bully Algorithm, only one process initiates an election procedure and determines the coordinator.
  - Unlike the Improved Bully Election Algorithm for Coordinator in the Synchronous Distributed Systems, there is no need to divide processes into sets.
- Efficient–Reduces the number of required messages.
- Fast–Reduction of the number of messages speeds up the election procedure.
- Safe–It ensures that the system has only one coordinator at any given time.

VIII. References

[1] A.S. Tanenbaum, M.V. Steen "Distributed systems: Principles and Paradigms." 2nd Ed, Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[2] M.R. EffatParvar, N. Yazdani, M.EffatParvar, A. Dadlani, and A. Khonsari. "Improved Algorithms for Leader Election in Distributed Systems." The 2nd International Conference on Computer Engineering and Technology (ICCET 2010), Vol. 2, pp. 6-10, April, 2010

[3] J. Villadangos, A. Córdoba, R Fariña , M. Prieto. "Efficient leader election in complete networks." In Proc. 13th Euromicro Conf. Parallel, Distributed and Network-based Processing, IEEE Computer Society, pp. 136-143, 2005.

[4] G. Singh. "Efficient Distributed Algorithms for Leader Election in Complete Networks." Distributed Computing Systems, 11th International Conference, pp. 472-479, on 20-24 May 1991.

[5] S. H. Park, Y. Kim, and J. S. Hwang. "An efficient algorithm for leader-election in synchronous distributed systems." IEEE TENCON, pp. 1091-1094, 1999.

[6] H. Garcia-Molina. “Elections in Distributed Computing System.” IEEE Transaction Computer, Vol.C-31, pp. 48-59, 1982.
[7] Md. G. Murshed and A. R. Allen. “Enhanced Bully Algorithm for Leader Node Election in Synchronous Distributed Systems.” Computers (2012), Vol. 1, pp. 3-23.
[8] M. Zargarnataj. "New Election Algorithm Based on Assistant in Distributed Systems." Proc. IEEE AICCSA, pp. 324-331, 2007.
[9] G. N. Frederickson, N. A. Lynch. “Electing a leader in a synchronous ring.” Journal of the ACM (1987), v.34 n.1, pp.98-115.
[10] G.N. Frederickson, N. Lynch. "The impact of synchronous communication on the problem of electing a leader in a ring." Proc. 16th ACM Symp. Theory of Computing, pp. 493–503, 1984.
[11] M. Rahman and A. Nahar. "Modified bully algorithm using election commission." MASAUM Journal of Computing (2009), vol.1 no.3, pp. 88-96.
[12] G.L. Peterson. “An O(nlog n) Unidirectional Algorithm for the Circular Extrema Problem.” ACM Transactions on Programming Languages and Systems (TOPLAS), v.4 n.4, pp. 758-762, 1982.
[13] W. R. Franklin. "On an improved algorithm for decentralized extrema finding in circular configurations of processors." Commun. Ass. Comput. Mach., vol. 25, pp. 336-337, 1982.
[14] B. Awerbuch, “Optimal Distributed Algorithm for Minimum weight spanning tree, Leader Election and related problems.” ACM STOC, pp. 230-240, 1987.
[15] P B. Soundarabai, R. Sahai, J. Thriveni, K R Venugopal and L M Patnaik. “Improved Bully Election Algorithm for Distributed Systems.” International Journal of Information Processing (2013), 7(4), pp. 43-54.
[16] A. Arghavani, E. Ahmadi and A.T. Haghighat. "Improved bully election algorithm in distributed systems." Information Technology and Multimedia (ICIM), International Conference, 2011.

Qamil Kabashi is professor of Department of Informatics Engineering and vice dean at Faculty of Mechanical and Computer Engineering, University of Mitrovica, 40000 Mitrovica, Republic of Kosovo (e-mail: qamil.kabashi@uni-pr.edu).
Arbnor Zeqiri is Software Backend Engineer/Team Leader and actually is working on master thesis: “Application of RFID and GPRS in Access Control System.” at Faculty of Mechanical and Computer Engineering, University of Mitrovica, 40000 Mitrovica, Republic of Kosovo (e-mail: arbnor.zeqiri@uni-pr.edu).
Milaim Zabeli is professor and head of Department of Informatics Engineering, Faculty of Mechanical and Computer Engineering, University of Mitrovica, 40000 Mitrovica, Republic of Kosovo (e-mail: milaim.zabeli@uni-pr.edu).