**Distributed Algorithms** **M.Tech., CSE, 2016**

# Lecture 2: Leader election algorithms.

*Faculty: K.R. Chowdhary* *: Professor of CS*

## 2.1 Leader Election

In distributed algorithms, a leader election is a process of designating a single process as the organizer, coordinator, initiator or sequence of some task distributed among several computers (nodes).

In other words, leader election is a process of determining that a process will be manager of some task distributed among several processes or nodes.

Following are the reason to elect a leader:

- A centralize control simplifies process synchronization, but it is single point failure, and can limit the service.
- To provide solution to choose a new controller (leader) upon failure of existing controller.
- There are many algorithms for leader election, and one of those can be chosen based on specific requirements.

When to elect a leader?

- During the system initiation or when the existing leader fails.
- A process that gets no response from the current leader for a predetermined time. This time out suspects a leader failure and hence should initiate a leader election.

There are two election criteria:

- *Extrema finding:* It is based on global priority. Every process is characterized by fixed evaluation value.
- *Preference-based:* Processes in the group can vote for a leader based on a personal preference (e.g. locality, reliability estimation, etc).

*Electing a unique leader process from among the processes in a network:* For this we consider the digraph as a ring. The communication is achieved by token passing. Some times when the token is lost, then it is required to execute an algorithm to regenerate the token again. The regeneration amounts to electing a leader. The problem can be sated as follows: We will consider a network of $n$ processors circularly placed on a ring. In unidirectional (clockwise), each process $p_i$ sends messages to process $p_{i+1}$ and receives messages from the process $p_{i-1}$. We assume modulo $n$ arithmetics). Accordingly, process $2n+1$ will receive from $2n$, and will send it to $2n+1$, and so son. We take processor number node as $n = 0, 1, 2, ..., n-1$. When a message has taken a round, the numbers are $n, n+1, n+2, ..., 2n-1$. Also, the node 0 after consecutive rounds becomes $n, 2n, 3n, ...$. Similarly, node 1 becomes $n+1$, $2n+1$, and so on (see fig. 2.1).

It is assumed that each process has a $UID$ (unique identifier) and knows its neighbors, and their uids. In addition, only one process should announce that it is leader.

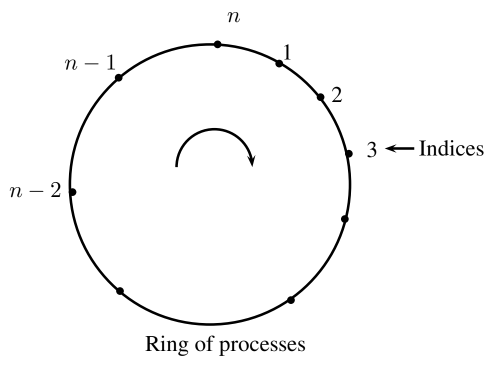


Figure 2.1: A ring of processes.

The basic algorithm is LCR (Lehann-Chang Robots) after its inventors' names. It uses unidirectional communication, and only the leader performs an output.

The leader election is a different process than an already known scenario called mutual exclusion. The following table 2.1 explains the differences.

Table 2.1: Leader election v/s Mutual exclusion.

| Leader election | Mutual exclusion |
|---|---|
| 1. Processes may execute normally as long as a leader remains selected | 1. Process competes until it succeeds. |
| 2. Concerned with fast and successful termination of election process. | 2. Must ensure that no process is starved. |
| 3. Result of leader election must be known to all other processes. | 3. Does not care which process is running in the critical section. |

The design topologies for networks can be: complete topology, logical ring topology, or tree topology.

## 2.2 LCR (LeLann-Chang-Roberts) Algorithm

Each process sends an identifier (uid) around the ring, when a process receives a *uid*, it compares with its own. If greater, it keeps passing to next process, and if less it discards, and if equal, then declares itself as a leader.

In the algorithm, the process with the largest UID is the only one that outputs the leader.

1. Assume clockwise unidirectional ring, like shown in figure 2.1.
2. One or more processes $p_i$s can take the initiative and start an election, by sending an election message containing their *uid* to $p_{i+1}$.

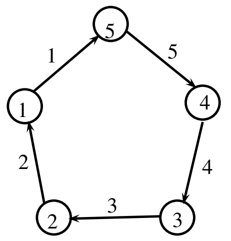
3. When a process $p_i$ spontaneously or upon receiving a message goes in an election, it marks itself as a participant.
4. If the $p_i$ receiving an election message has a greater id and is not already a participant, then it sends an election message with its own id to $p_{i+1}$.
5. If its own id is smaller, it forwards the message with the *uid* it has received.
6. If it receives a message with its own *uid* then it declares itself as the leader.

Algorithm for it can be given as:

```
boolean participant=false;
int leader-id=null;
% Initiate an election:
send(ELECTION(my-id));
participant:=true;
Upon receiving a message ELECTION(j):
if (j $>$ my-id) then send(ELECTION(j));
if (my-id = $j$) then send(LEADER(my-id));
if (my-id $>$ j) $\wedge$ ($\neg$ participant) then
   send(ELECTION(my-id));
participant:=true; % propagate leader uid
Upon receiving a message LEADER(j):
leader-id:=j;
if (my-id $\neq$ j) then send(LEADER(j));
end.
```

Note the following points in the algorithm:

- Only the message with the largest identity completes the round trip and returns to its originator, which becomes the leader.
- Time complexity: $O(n)$
- The leader has to announce itself to all $p_i$s through the leader messages, so that termination is guaranteed and everybody knows who the leader is.
- The algorithm verifies the safety and liveness conditions with:
  - done(i) $\equiv$ (leader-id(i) $\neq$ null)
  - is-leader(i) $\equiv$ (leader-id(i) = i)



Message transmitted:

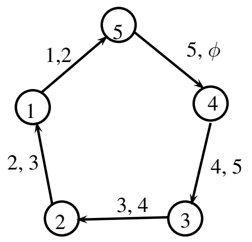
| | |
|---|---|
| <1> | 1 time |
| <2> | 1 time |
| <3> | 1 time |
| <4> | 1 time |
| <5> | 1 time |
| Total | 5 times |

Figure 2.2: Demonstration of LCR Algo. stage 1.

**Example 2.1** *An example of running LCR algorithm.*

Assume that all the $p_i$s are initiators (see fig. 2.2).

The stage-2 of message transmission by each process $p_i$ in a ring is shown in fig. 2.3.

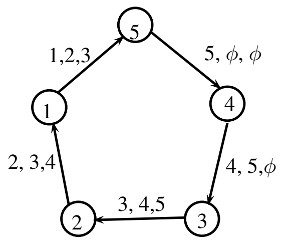


Message transmitted:

| | |
|---|---|
| <1> | 2 time |
| <2> | 2 time |
| <3> | 2 time |
| <4> | 2 time |
| <5> | 1 time |
| Total | 9 times |

Figure 2.3: Demonstration of LCR Algo. stage 2.

The stage-3 of message transmission by each process $p_i$ in a ring is shown in fig. 2.4.



Message transmitted:

| | |
|---|---|
| <1> | 3 time |
| <2> | 3 time |
| <3> | 3 time |
| <4> | 2 time |
| <5> | 1 time |
| Total | 12 times |

Figure 2.4: Demonstration of LCR Algo. stage 3.

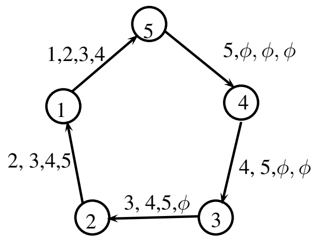
The stage-4 of message transmission by each process $p_i$ in a ring is shown in fig. 2.5.

The stage-5 of message transmission by each process $p_i$ in a ring is shown in fig. 2.6.

Now, the leader $uid = \langle 5 \rangle$ has to be announced to all nodes with 5 more messages. So, in total 15+5=20 messages are transmitted. Note that each identifier $i$ is sent $i$ times.

### 2.2.1 LCR algorithm: Message complexity

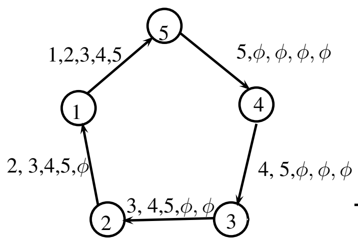
We are interested in message complexity, which depends on how the *ids* are arranged.



Message transmitted:

| | |
|---|---|
| <1> | 4 time |
| <2> | 4 time |
| <3> | 3 time |
| <4> | 2 time |
| <5> | 1 time |
| Total | 14 times |

Figure 2.5: Demonstration of LCR Algo. stage 4.



Message transmitted:

| | |
|---|---|
| <1> | 5 time |
| <2> | 4 time |
| <3> | 3 time |
| <4> | 2 time |
| <5> | 1 time |
| Total | 15 times |

Figure 2.6: Demonstration of LCR Algo. stage 5.

- The largest *id* always travels all around the ring ($n$ msgs).
- 2nd largest id travels until reaching the largest.
- 3rd largest id travels until reaching largest or second largest, and so on.

Worst way to arrange the *ids* is in decreasing order (and all $p_i$s are initiators): 2nd largest causes $n-1$ messages, 3rd largest $n-2$ messages etc.

Number of $msgs = (n + (n-1) + ... + 1) + n = \frac{n(n+1)}{2} + n$ (including the $n$ leader messages at the end). Hence, the worst case complexity is $O(n^2)$.

## 2.3 Complete topology

In the complete topology, each process can communicate any other process in the same group in one message hop, providing minimum time-complexity for communication. Following are the assumption in complete topology:

- All process IDs are unique and known to others,
- Network is reliable and only processes may fail.
- A process takes a known finite amount of time to handle a message.

### 2.3.1 Bully Algorithm

It was introduced by Garcia-Molina, and is an extrema-finding algorithm. Process with the highest priority is elected as a leader, hence name, Bully algorithm.

The Bully algorithm is based on complete topology network. It is an extreme finding algorithm, i.e., process with the highest priority is elected as a leader, hence Bully algorithm. The Bully algorithm works as follows:

**Algorithm:**

1. Process $P$ starts a leader election if it suspects the failure of existing leader.
2. $P$ sends inquiry message to nodes (processes) with higher priority.
3. If any response then, $P$ gives up the election and waits for higher priority node to elect itself leader.
4. If no response then $P$ becomes a leader.

The algorithm is as follows:

```
1. send := null ; cleans up from previous message
2. if incoming message is $v$, a UID, then
3. case
4.    $v > u$ : send := $v$
5.    $v = u$ : status := leader
6.    $v < u$ : do nothing
7. endcase
```

Correctness of algorithm means exactly one process performs as leader. The entire code for $trans_i$ is supposed to be executed individually. Let $i_{max}$ is index of process with maximum UID, and $u_{maxx}$ is its UID. It is enough to show that process $i_{max}$ outputs leader by the end of round $n$, and no other process ever performs such an output.

In complete topology, each process can reach any other process in the same group in one message hop. It is assumed that: All process ids are unique and each process knows to every other process, communication network is reliable and only the communicating processes may fail, and process takes a known finite amount of time to handle a message.

The figure 2.7 demonstrates the application of bully algorithm for leader election in complete topology.

The figure 2.7(a) shows that process 4 detected leader failure (process 7 was leader and it has failed) and initiated an election. Next, figure 2.7(b) shows that processes 5 and 6 respond telling to process 4 to stop. Now, 5 and 6 each hold an election (figure 2.7(c)).

The figure 2.8(d) shows that Process 6 responds to the call from 5 and tells it to stop. Finally, the figure 2.8(e) shows that process 6 wins and tells everyone.

## 2.4 Logical Ring topology

It is simplest and easy to construct topology. The unique property is - message initiated by a node will return to the same node ultimately. It eliminates the need of acknowledgement. Following is the algorithm for leader election for ring topology.

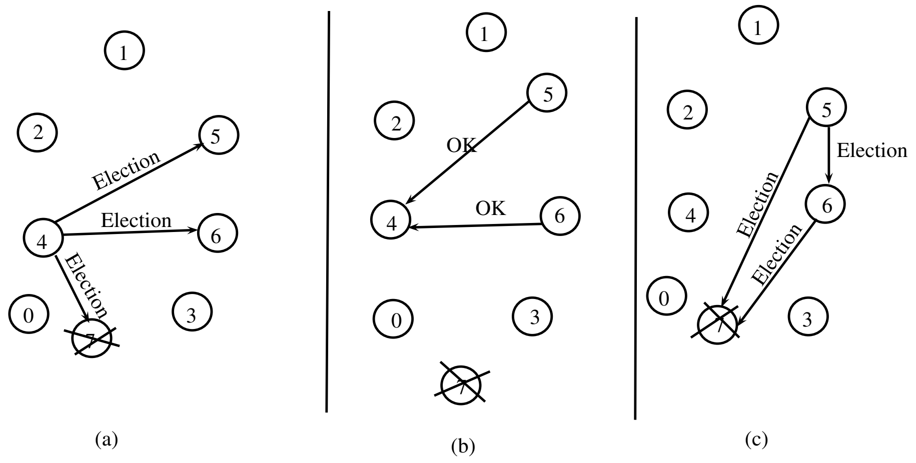


Figure 2.7: Bully algorithm steps for leader elect in complete topology.

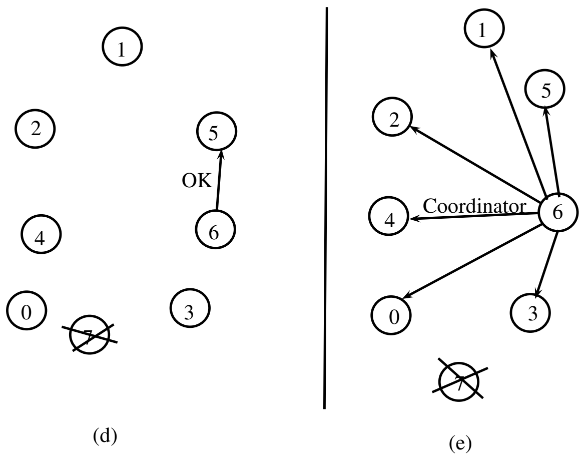


Figure 2.8: Bully algorithm steps for leader elect in complete topology.

**Leader election for Ring topology:**

1. On detection of a leader failure, process starts election by circulating a message with priorities appended to the message by each node along the ring.

2. Message comes back to initiator, it chooses the highest priority and broadcast the new leader identity to all nodes.

## Exercises

1. The *mutual exclusion* allows a privileged execution of an algorithms or its part; and the *leader* in the algorithm is coordinator as well as initiator of actions. In spite of similar nature of both, explain, why these two things are totally different.

2. Consider that there are number of algorithms executing not as parallel algorithms, but concurrent, i.e., only one processor executes the processes $p_1, \dots, p_n$ in round robin fashion.
   (a) Explain the concept for implementing these
   (b) Modify the LCR algorithm to be executed for single processor system, which executes all algorithms in sequential manner.
3. Answer the following for LCR algorithm:
   (a) Is it synchronous or anynchronous?
   (b) Is there any situation, where it fails to execute?
4. For leader election in an anonynous Ring,
   (a) Is deterministic leader election possible in a synchronous Ring where all but one process have same uid? Give an algorithm if yes, else prove the contradiction.
   (b) Same as (a) above, but in place of one process, consider two processes.
5. For Bully algorithm:
   (a) Give a diagram and explain 3 messages - election, vote, and coordinator.
   (b) What happens if two processes notice at the same time, that a process has crashed?
6. Which of the following distributed algorithm function only in a synchronous system?
   (a) Ring-based leader election
   (b) Bully algorithm.

## References

[1] ALFRED V. AHO, JOHN E HOPCROFT, AND JEFFREY D. ULLMAN, "Data Structures and Algorithms," *Pearson Education, India*, 2002.

[2] ALLEN B. TUCKER, JR., "The computer Science and Engineering Handbook," *CRC Press*, 1997.