Sanyam Kaul

Theory Assignment - 2

02 April 2024

Q1

As per the question, the graph is not fully connected, but indirect paths exist between two pairs of nodes. We can keep track of the neighbours for every node, and it can help us in leader election using the Bully algorithm. Below is the explanation in detail: -

1. Every node keeps track of its neighbours. A neighbour is the node which is directly connected to the given node. Nodes can have a separate exchange of messages to update their neighbour list.
2. When a node detects that the leader has failed, it initiates the leader election locally among its neighbours. The original Bully algorithm is used here. The node with the highest ID is elected as the local leader.
3. Once the local leader is elected, it informs about its leadership to its neighbours. In this way, the latest leadership information is propagated across the network.
4. If a node detects that one of its neighbours has become a leader, then it should consider the elected neighbour as a potential global leader and evaluate its local leader accordingly. This ensures that the algorithm eventually converges to a single global leader.

The pseudo-code of the modified Bully Algorithm is below: -

```
// Variables

leader: id of the current leader

neighbors: list of direct neighbors

isLeader: boolean indicating if this node is currently the leader



// Function to initiate leader election

function initiateLeaderElection():

    // If current leader is unreachable or failed

    if leader not responding:

        // Inform neighbors of the need for leader election

        sendElectionMessage(myId)



// Function to handle incoming election message

function handleElectionMessage(senderId):

    // If sender has higher priority

    if senderId > myId:

        // Relay the election message to higher priority neighbors

        for each neighbor with higher priority than myId:

            sendElectionMessage(senderId)
```

```
    else if senderId < myId:
        // Sender has lower priority, reply with my id
        sendOkMessage(myId)

// Function to handle incoming OK message
function handleOkMessage(senderId):
    // If sender has higher priority, cancel election
    if senderId > myId:
        cancelElection()
    else:
        // Continue election process
        continueElection()

// Function to handle becoming leader
function becomeLeader():
    // Update leader status
    leader = myId
    isLeader = true
    // Inform neighbors about leadership status
    sendLeaderMessage(myId)
```

```
// Main function for processing incoming messages
function processMessage(message, senderId):
    if message == Election:
        handleElectionMessage(senderId)
    else if message == Ok:
        handleOkMessage(senderId)
    else if message == Leader:
        leader = senderId
        isLeader = false

// Main function to handle leader election process
function main():
    // Initialize variables
    leader = null
    isLeader = false

    // Main loop
    while true:
        // Periodically check leader status and initiate election if necessary
```

```
initiateLeaderElection()



// Process incoming messages

for each incoming message:

    processMessage(message, senderId)
```

Q2

In class, we studied three token ring-based leader election algorithms: Chang-Roberts Algorithm, Franklin's algorithm, and Peterson's algorithm. These are the specific implementations of the Token-Ring algorithm protocol for leader election in Synchronous systems. Franklin's Algorithm is an improvement of the Chang-Roberts algorithm as it uses bi-directional communication channels and reduces the message complexity. Peterson's algorithm is a further improvement on the Franklin's algorithm.

I did some literature review on the token-ring-based algorithms and found this paper. The paper talks about leader elections in anonymous, bi-directional, asynchronous rings. It discusses modifying Franklin's algorithm to achieve leader election in an async system. Below is a brief explanation of the modification made in Franklin's algorithm, which makes it suitable for anonymous, bidirectional, asynchronous rings: -

1. Initialisation: Each node in the ring is initially in one of two states, either "candidate" or "non-candidate."

2. Probability Distribution: The algorithm requires a specific probability distribution to be defined. Nodes need to know this distribution, determining the probability of a node becoming a candidate.
3. Message Passing: Nodes exchange messages with their neighbours asynchronously. These messages contain information about the node's current state.
4. State Transitions: Nodes may change their states based on the received messages and the defined probability distribution. The critical transition is from non-candidate to candidate state, which happens probabilistically.
5. Termination: The algorithm continues until only one node remains in the candidate state. This node is then declared the leader.

Below is the pseudo-code of the modified algorithm: -

```
1. Initialize:

   Set initial state:

      - Each node is either in the "candidate" or "non-candidate" state.

      - Initially, all nodes are in the "non-candidate" state, except for one node
(chosen arbitrarily), which is in the "candidate" state.

   Define probability distribution for state transition from non-candidate to
candidate.
```

```
2. While there are still multiple candidates:

   Each node performs the following steps:

      a. If the node is in the candidate state:

         - Continue being a candidate.

      b. If the node is in the non-candidate state:

         - Receive messages from neighbors.

         - Based on the received messages and the defined probability distribution:

            * Transition to the candidate state with a certain probability.

            * Otherwise, remain in the non-candidate state.

         - Send messages to neighbours to inform them of the node's current state.



3. Termination:

   Once only one node remains in the candidate state, it is declared the leader.
```

Below is a brief description of the correctness of this approach: -

1. Probability Distribution: The algorithm's correctness depends on the choice of the probability distribution used for state transitions. This distribution needs to be

carefully chosen to ensure that, eventually, with high probability, each node becomes a candidate.

2. Asynchronous Nature: Since the algorithm operates in an asynchronous system, nodes have no global clock or synchronisation. As a result, nodes may experience arbitrary delays in message delivery and processing. The algorithm must be designed to handle these asynchronous behaviours.
3. Convergence to a Unique Leader: The proof typically shows that, under the chosen probability distribution and asynchronous assumptions, the algorithm converges to a state where only one node remains a candidate. This last remaining candidate is then declared as the leader.
4. Probability of Correctness: The proof also analyses the likelihood of correctness, demonstrating that the probability of failure decreases exponentially with the number of iterations. This ensures that, with high probability, the algorithm elects a leader correctly.
5. Robustness to Faults: The algorithm's correctness proof may also consider various failure scenarios, such as message loss or node failures. It should show that the algorithm remains correct and eventually terminates even with such faults.

Works Cited

Leader Election in Anonymous Rings: Franklin goes Probabilistic - Link to paper