Chapter 6

Multi-attribute based self-stabilizing algorithm for leader election in distributed systems

Nowadays, several areas such as artificial intelligence, distributed machine learning, distributed deep learning expert systems, natural language processing, cloud computing, edge computing and robotics realise the requirement of a truly distributed system and try to adopt it. In distributed systems, leader election is a fundamental problem because the elected leader node coordinates all the nodes (resources). A good quality (suitable) leader improves the system's resource utility and overall performance, simplifies its management procedure, reduces coordination and operational complexity, and makes the system more fault-tolerant. The existing well-known leader election algorithms talked about electing a good quality leader [104] [20] [102]. However, no precise and rigorous leader election method has been proposed to elect a good quality leader based on the system requirements. Different distributed systems are designed for different purposes. So the definition of a good quality leader may vary from system to system. On the other hand, a distributed system may consist of heterogeneous nodes. A node can have multiple attributes, and different nodes can have different values of those attributes. So it is challenging to determine which type of quality a leader of a distributed system should have and which attributes

are responsible for that quality of the leader. It is also difficult to determine how much priority has to be given to those pertinent attributes. Hence electing a good quality leader for a system considering multiple attributes is a pretty intricate task.

This chapter proposes a self-stabilizing leader election method to elect a good quality leader (according to the system requirements) for a dynamic distributed system. The proposed leader election algorithm can be used in replicated distributed databases, cloud computing, distributed edge computing and new technologies such as distributed machine learning, distributed artificial intelligence, e-learning, etc. Seeing the popularity of replicated cloud servers in distributed databases, we consider the scenario of replicated cloud servers like Cloud Spanner [32], and conduct an experiment to assess the efficiency of the elected leader using the proposed election method.

Outline: Section 6.1 describes the considered system model. We describe the proposed leader election method in section 6.2. Proof of self-stabilization, complexity analysis and an illustrative example are also given in this section. The experimental results are presented in section 6.3 and section 6.4 summarizes the chapter.

6.1 System Model

This section describes the system model. Various definitions and assumptions are given in this section that will be used in the following sections. For this work, we consider a distributed system consisting of N nodes (N is a finite integer and $N \geq 2$). These nodes are uniquely identifiable by their Ids which are distinct integers, and they are connected through an arbitrary network topology. The nodes can be heterogeneous, and they communicate with one another using message passing. Each node has an adequately large receive buffer to avoid buffer overflow. The links are bidirectional and a message gets transmitted from one node to its adjacent node through a correct link within a bounded time. Each node has a local clock. The local clocks need not be synchronized, and they are used only to implement timers. A link cannot create, corrupt, or duplicate the messages. Formally, $D = (\Pi, L)$ represents our considered distributed system, where Π refers to the set of node Ids ($\Pi = \{N\_Id_i\}$, $i = 0, 1, 2, \cdots, N-1$) and L refers to the set of

network connectivity (links) of the nodes. The nodes and the network connections may fail and recover independently. Each node has a failure detection module. By invoking this module, a node can detect the link and node failures in the system.

6.1.1 Definitions

Definition 1 **(Stable node):** A node is called a stable node if it does not fail for a long time. Especially during the selection, a stable node does not fail until the election process ends.

Definition 2 **(Correct link):** A link is said to be a correct link if it can send a message from one node to another node (connected through this link) within a bounded time without any modification of the message.

Definition 3 **(Network regions):** Due to link and/or node failures, a network may get divided into smaller sub-networks. These sub-networks are called network regions.

Definition 4 **(Status of a node):** A node can have one of the three statuses, i.e., $NULL$, *Leader* and *Non-leader*. The variable $Status_i$ represents the status of a node i. The status of a node i is set to *Leader* if and only if the node is elected as the leader. Otherwise the status is set to *Non-leader*. During the election, the status of every node is set to $NULL$ until a node gets elected as the leader.

6.1.2 Assumptions

- Every network region has some stable nodes (at least one stable node). The nodes of a region may not know who are the stable nodes.
- After recovering from the failure state, a node does not fail again instantly, which means it exists in the system for a certain amount of time, and then it can fail.
- Every node knows the Id of its all adjacent nodes and also knows the minimum and maximum value of every attribute.

6.2 The Proposed Leader Election Method

A good quality leader can coordinate the nodes for completing various tasks while maintaining consistency, efficiently managing the system's resources, reducing communication overhead, etc. The quality of a leader depends on multiple quality attributes. The proposed leader election method considers the multiple quality attributes that determine the quality of the node. CPU capacity, available memory, failure rate, security level, degree, eccentricity, closeness centrality, etc., are some quality attributes that define the node in a distributed system. Based on the system's goal, it is challenging to identify multiple quality attributes that precisely define a suitable system leader where the importance of attributes changes accordingly. Thus, it is required to specify a proper set of quality attributes that can distinctly characterize the nodes participating in the leader election process. And in the light of multiple attributes, we need to assess the worth of each identified attribute utilized to elect a leader. Since the system itself cannot recognize the attributes, we require system experts to understand the system. Hence, we attempt to propose a multiple attribute based decision making approach [58] to solve the leader election problem. The proposed process consists of two parts: *i*) appropriate attributes identification and their prioritization *ii*) designing of the leader election algorithm. We employ the group decision-making approach in the first part, where a group of system experts is invited to express their knowledge over the set of attributes through preference relations. And in the second part, the decision result obtained from the first part is provided as input for initiating the election process.

6.2.1 Identification and prioritization of the quality attributes

In distributed system environment, it is required to identify all the possible quality attributes that characterize a node. Each expert provides different sets of quality parameters. Let $E = \{e_1, e_2, e_3, \ldots, e_\lambda\}$ where $\lambda \geq 2$ be the finite set of experts and each expert provides its own set of attributes for the decision analysis. Let η_k be the number of attributes suggested by the expert $e_k \in E$ then $Q_k = \{q_{kl}\}$, $\forall\ l \in \{1, 2, \ldots, \eta_k\}$ and $k \in \{1, 2, \ldots, \lambda\}$, be the set of attributes provided by some expert e_k. Based on the requirement of the system and the experts understanding and knowledge, the set of attributes provided by

each expert may or may not be the same. So, we need to decide on a global finite set of attributes that will be communicated to all experts to obtain their preferences. For this, we apply a union operator over the attribute set provided by each expert to obtain the global set of attributes Q.

$$Q = \bigcup_{k=1}^{\lambda} Q_k$$

Now, the global set of attributes Q of size say m, where m is the number of unique attributes, is suggested by the experts. For this obtained set Q, each expert expresses their opinions in the form of a pairwise comparison matrix using fuzzy preference relation [98]. Let $P^k = (p^k_{ij})_{(m\times m)}\ \forall\ i,j \in \{1,2,\ldots,m\}$ be the preference matrix provided by expert e_k over the set of attributes say $Q = \{q_1, q_2, \ldots, q_m\}$, where $m \geq 2$. Once the evaluation matrix P^k of each individual expert is obtained, the score of each attribute q_i is computed in two phases [58]: 1) aggregation and 2) Exploitation

Aggregation Phase: This phase is used to combine experts' preferences. It defines the collective preference, $P^c = (p^c_{ij})_(m \times m)$, obtained by the aggregation of the individual preference relations $\{P^1, P^2, \ldots, P^\lambda\}$ as shown: $P^c = \varphi(P^k), k \in \{1,2,\ldots,\lambda\}$ and $\lambda \geq 2$ Where P^c is the global preference between every pair of attributes from the opinions obtained from the majority of experts and φ is an aggregation operator such as ordered weighted averaging (OWA) or weighted averaging (WA) operator [119] used to aggregate the individuals preference.

Exploitation Phase: This phase is used to obtain the solution set of attributes from the global preference information obtained from the aggregation phase. The acquired global preference is transformed to get the scores of the attributes, defining their priorities. Choice functions [97] are used to obtain the solution set, and therefore we apply the dominance degree for each attribute q_i. The following function is used to obtain the dominance degree:

$$D(q_i) = \frac{1}{m-1} \sum_{j=1, j\neq i}^{m} p^c_{ij}$$

where m is the number of global quality attributes and p_{ij}^{c} is the preference value of attribute i over attribute j according to the group of experts. And finally, the importance level of an attribute defined by the weight assigned w_i, for $i \in \{1, 2, \ldots, m\}$ where $w_i \geq 0$ and $\sum_{i=1}^{m} w_i = 1$, we normalize the obtained dominance degree as shown:

$$w_i = \frac{D(q_i)}{(\sum_{i=1}^{m} D(q_i)}$$

where w_i is the weight assigned to the quality attribute q_i. And this obtained weight value will be treated as input to the leader election algorithm. For a particular system, the quality attributes identification and prioritization is done only once. Further, it does not need to re-identify and re-prioritize the quality attributes until the system's requirements get changed.

6.2.2 The leader election algorithm

In the first part of the election method, with the help of some experts' opinions, we identify the appropriate attributes and assign their weight that is considered at the time of leader election. Since we consider multiple attributes of the nodes, we can use the multi-criteria decision-making (MCDM) method to elect the leader node. However, a node may or may not have information about all other system nodes in a distributed system. In general, a node does have some nodes' information. Therefore we cannot use an MCDM method directly to elect the leader. On the other hand, we should choose an MCDM method with minimum time complexity and implementable in a distributed system. TOPSIS [121] [110] is a well-known and widely accepted MCDM method with adoptable time complexity. Here we modify the concept of the TOPSIS method so that it can work in a distributed scenario to elect a leader. We assume that for each attribute, a node knows the minimum and maximum value of the attribute. Before taking part in the election, a node calculates its quality factor. The system leader is elected based on the quality factor of the nodes. The node with the highest quality factor is elected as the system leader. A node calculates its quality factor using the following four steps.

Step 1: Normalization

Different attributes of a node are usually measured in different units. The normalization transforms different dimensional attributes into dimensionless attributes that allow comparisons across criteria. A node i normalizes its attributes' values using the following equation.

$$n_{ij} = \frac{a_{ij} - min(q_j)}{max(q_j) - min(q_j)}$$

Here, a_{ij} is the value of attribute q_j of a node i. $min(q_j)$ and $max(q_j)$ are the possible minimum and maximum value of the attribute q_j respectively.

Step 2: Weighted normalization

A node i calculates the weighted normalize value (v_{ij}) using the following equation. $v_{ij} = w_j \cdot n_{ij}$

Step 3: Distance calculation

In this step, a node i calculates the euclidean distance from the ideal best and ideal worst. Here, d_{ib} and d_{iw} represent the euclidean distance from the ideal best and ideal worst respectively.

$$d_{ib} = \sqrt{\sum_{j=1}^{m} (v_{ij} - v_b)^2}$$

where, $v_b = 1$ if q_j is a benefit attribute and $v_b = 0$ if q_j is a cost attribute. Benefit attributes are those attributes whose higher values are preferred while cost attributes are those attributes whose lower values are preferred during leader election.

$$d_{iw} = \sqrt{\sum_{j=1}^{m} (v_{ij} - v_w)^2}$$

where, $v_w = 0$ if q_j is a benefit attribute and $v_w = 1$ if q_j is a cost attribute.

Step 4: Quality factor calculation

In this step, the quality factor (qf) of a node i is calculated and the leader is elected based on the quality factor of the nodes.

$$Qf_i = \frac{d_{iw}}{d_{ib} + d_{iw}}$$

In the proposed leader election method, we use five types of messages to conduct an election. They are *i*) $EIM[Emc\_Id, S\_eim]$, *ii*) $ACK[Emc\_Id, C\_ack]$, *iii*) $AGM[Emc\_Id, C\_agm]$, *iv*) $MQFM[Mqf, Mqf\_Id, S\_mqfm]$, *v*) $LDM[El\_Id]$.

$EIM[Emc\_Id, S\_eim]$ is the election-initiating message, it is used to initiate an election, and it has two fields: the Id of the election initiating node (Emc_Id) and the Id of the EIM message sender (S_eim). $ACK[Emc\_Id, C\_ack]$ is an acknowledgement message. In general, when a node gets an election-initiating message (EIM) from an adjacent node, it creates an $ACK[Emc\_Id, C\_ack]$ message and sends it to the adjacent node that had sent the EIM message. It has two fields: the Id of the election initiating node (Emc_Id) and the Id of the acknowledgement message creator. If a node gets the same EIM message from another adjacent node(s), then the node creates an $AGM[Emc\_Id, C\_agm]$ message and sends it to the EIM message sender node(s) to inform them that the EIM message is already received. It also has two fields: the Id of the election initiating node (Emc_Id) and the Id of the AGM message creator. $MQFM[Mqf, Mqf\_Id, S\_mqfm]$ is the maximum quality factor message. It is used to pass the information of a node with the leading quality factor. It has three fields: the maximum quality factor (Mqf), the Id of the node with maximum quality factor (Mqf_Id) and the Id of the $MQFM$ message sender (S_mqfm). $LDM[El\_Id]$ is used to declare the Id of the elected leader to all the nodes in the system. It has only one field i.e., the Id of the elected leader (El_Id).

In our proposed algorithm, a node can be one of the three types of nodes i.e., the parent node, the co-parent node and the child node. Here, we introduce the concept of a co-parent node to increase the number of link and node failures tolerability of the proposed election algorithm. This algorithm is divided into three phases: Initiation of the election, Intermediate steps and Declaration of the leader, which are given in Algorithm 11, Algorithm 12 and Algorithm 13 respectively. We explain each phase as follows.

I. Initiation of the election: When a node identifies that the system leader is failed, it creates an election-initiating message ($EIM[Emc\_Id, S\_eim]$ where Emc_Id contains the Id of the election initiator and S_eim contains the Id of the EIM message sender. In the case of the election initiating node, both fields of the EIM message hold the same node Id (the Id of the election initiator) because the node that creates the EIM message also

sends it) and starts the election by sending it to all the adjacent nodes. After sending the EIM message, the node expects an acknowledgment message (ACK) from every adjacent node. If it does not get an ACK message from an adjacent node, then it does not consider that node as its adjacent node anymore. The node then waits $Tf_{e\text{-}m}$ amount of time to get an $MQFM$ message from each of its adjacent nodes. If multiple nodes initiate the election simultaneously, the node with the minimum Id among those election initiating nodes gets the scope to conduct the election. That means the EIM message created by the node with the minimum Id gets spread over the network, and the EIM messages created by the other nodes get discarded.

Algorithm 11: Initiation of the election

1 **if** *(A node i identifies that the leader is failed)* **then**
2 $Emc\_Id \leftarrow N\_Id_i$, $S\_eim \leftarrow N\_Id_i$, $Ein\_Id_i \leftarrow N\_Id_i$, $Pnode_i \leftarrow NULL$, $L\_Id_i \leftarrow NULL$, $Status_i \leftarrow NULL$.
3 Create the $EIM[Emc\_Id, S\_eim]$ and send it to all the adjacent nodes and wait for $Tf_{e\text{-}m}$ time to get an $MQFM$ message from each adjacent node.
4 **end**

II. Intermediate steps: When a node receives an election-initiating message (EIM), it checks whether it is first received EIM message. If it is its first received EIM message, the node considers the EIM message sender as its parent node and knows about the election initiator. The node stores the Id of the EIM message sender and the Id of the election initiator in its *pnode* and *ein_id*, respectively. If the node itself is a leaf node, it creates a maximum-quality-factor message ($MQFM$) using its self-information, sends it to the parent node and waits for $Tf_{e\text{-}m}$ time to get a LDM message. If the node is not a leaf node, it creates an acknowledgment (ACK) regarding the received EIM message and sends it to its parent node. Besides, the node replaces the second field of the received EIM message with its own Id and sends it to all the adjacent nodes except the parent node. Then the node waits for $Tf_{e\text{-}m}$ time to get the $MQFM$ messages from the adjacent nodes whom it sent the EIM messages.

On the other hand, if the received EIM message is the second or onward EIM message, the node compares the first field of the EIM message (Emc_Id) with its own *ein_id* because if multiple nodes initiate the election simultaneously, the EIM message created by the minimum Id needs to be spread out in the network, and other EIM messages need to be discarded. If both are the same that means the node already got the same

EIM message, the node creates an AGM message regarding the received EIM message and sends it to the EIM message sender node and includes the S_eim in its *cpnode*. If the Emc_Id is less than the ein_id, the node removes all the Ids stored in its *cnode* and *cpnode* and acts the same way as it did when it got the EIM message for the first time. If Emc_Id is greater than ein_id, the node discards the received EIM message.

When a node receives an acknowledgment message (ACK) from an adjacent node, it considers that adjacent node as its child node and includes the child node Id into its $Cnode$.

When a node receives an (AGM) message from an adjacent node, it considers that adjacent node as its co-parent node and includes the co-parent node Id in its *cpnode*. Then it checks whether all the adjacent nodes are co-parent nodes except the parent node. If so, the node creates an $MQFM$ message using self-information and sends it to all the co-parent nodes. When a node gets all the expected $MQFM$ messages from its child nodes, it selects the maximum quality factor from among the received quality factors and its own quality factor. The node then makes a $MQFM$ message using the information of the node with the maximum quality factor, sends it to all the co-parent nodes and waits to get the $MQFM$ message from the co-parent nodes. As a node passes the information to its co-parent nodes about the node that has the higher quality factor among itself and its child nodes, the information of the node with the highest quality factor among all the nodes get several paths to reach the election conducting node. That increases the number of link and node failure tolerability of the proposed algorithm. Afterwards, when the node gets all the expected $MQFM$ message from the co-parent nodes, it selects the maximum quality factor among all the received quality factors (from all the child nodes and co-parent nodes) and its own quality factor, makes an $MQFM$ message using the information of the node with the maximum quality factor, sends it to its parent node and waits for an LDM message. If a node does not have any co-parent node, then after getting all the expected $MQFM$ messages from the child nodes, the node chooses the maximum quality factor among all the received quality factors (from all the child nodes) and its own quality factor, creates an $MQFM$ message using the information of the node with the maximum quality factor, sends it to its parent node and waits for a LDM message. If a node does not get the expected $MQFM$ messages from its adjacent nodes during a certain amount of time,

it sends a request message to those adjacent nodes to send the $MQFM$ message. If the node does not get any response regarding the request message from an adjacent node, it does not consider that node as its adjacent node anymore.

III. Declaration of the leader: When the election initiating node gets all the expected $MQFM$ messages from its adjacent nodes, it chooses the node with highest quality factor among the received quality factors and its own quality factor as the leader. If the election initiating node finds that multiple nodes have the highest quality factor, then it chooses the node with minimum Id among them as the leader. Thus the node with the highest quality factor is elected as the new system leader. To declare the newly elected leader, the election initiating node creates a leader declaration message ($LDM[El\_Id]$ where El_Id is the elected leader's Id) and sends it to all the adjacent nodes. When a node gets an LDM message, it gets to know about the newly elected leader and it sends the LDM to the adjacent node except the node that had sent it the LDM message. Further if the node gets the same LDM message, it discards the message.

If a node recovers from its failure state, it performs the following actions.

- The node collects all the required system information from its adjacent nodes.
- If the node gets back from its failure state during the election and its all adjacent nodes have sent the $MQFM$ message to their parent nodes, the node does not participate in the election. It only waits for the LDM message. Otherwise, the node randomly selects its parent node among the adjacent nodes that did not send a $MQFM$ message to their parent node. Then the node participates in the election by creating an $MQFM$ message using self-information and sending it to its parent node.

6.2.3 Proof of Self-stabilization

Lemma 6.1. *Let the system consists of N connected nodes and Π represents the set of all nodes ($|\Pi| = N$) and S_p is the set of participating nodes in the election. A proper execution*

Algorithm 12: Intermediate processing

if *(A node j gets an* $EIM[Emc\_Id, S\_eim]$ *message)* **then**
  if *(*$Ein\_Id_j == NULL$*)* **then**
    $Pnode_j \leftarrow S\_eim$, $Ein\_Id_j \leftarrow Emc\_Id$, $L\_Id_j \leftarrow NULL$, $Status_j \leftarrow NULL$
    if *(Node j is a leaf node)* **then**
      $Mqf \leftarrow Qf_j, Mqf\_Id \leftarrow N\_Id_j, S\_mqfm \leftarrow N\_Id_j$
      Create an $MQFM[Mqf, Mqf\_Id, S\_mqfm]$ and send it to the parent node.
      Wait for $Tf_{e\text{-}m}$ time to get a LDM message.
    else
      $S\_eim \leftarrow N\_Id_j$, $C\_ack \leftarrow N\_Id_j$
      Create an $ACK[Emc\_Id, C\_ack]$ message and send it to the sender of the EIM message.
      Send the $EIM[Emc\_Id, S\_eim]$ to all the adjacent nodes except the sender of EIM.
      Wait for $Tf_{e\text{-}m}$ time to get the $MQFM$ messages.
    end
  else
    if *(*$Ein\_Id_j == Emc\_Id$*)* **then**
      Include S_eim in $Cpnode_j$, $C\_agm \leftarrow N\_Id_j$
      Create an $AGM[Emc\_Id, C\_agm]$ and send it to the sender of the EIM message.
    else
      if *(*$Ein\_Id_j > Emc\_Id$*)* **then**
        $Pnode_j \leftarrow S\_eim$, $Ein\_Id_j \leftarrow Emc\_Id$
        Remove all the Ids from $Cnode_j$ and $Cpnode_j$.
        Follow step 4 to step 11.
      else
        Discard the received EIM message.
      end
    end
  end
end
if *(A node j gets an ACK message)* **then**
  Include C_ack in $Cnode_j$.
end
if *(A node j gets an AGM message)* **then**
  Include C_agm in $Cpnode_j$.
  if *(All the adjacent nodes are co-parent nodes except the parent node)* **then**
    $Mqf \leftarrow Qf_j$, $mqf_{\leftarrow} N\_Id_j$, $S\_mqfm \leftarrow N\_Id_j$
    Create an $MQFM[Mqf, Mqf\_Id, S\_mqfm$ message and send it to all the co-parent nodes.
  end
end
if *(A node j gets an MQFM message)* **then**
  Store the received $MQFM[Mqf, Mqf\_Id, S\_mqfm]$ message.
  if *(The expected MQFM messages are received from all the child nodes only but not from the co-parent nodes)* **then**
    Select the maximum quality factor from among the received quality factors and self quality factor.
    Make a $MQFM$ message using the information of the node with maximum quality factor and send it to all the co-parent nodes.
  end
  if *(The expected MQFM messages are received from all the child nodes and co-parent nodes or the expected MQFM messages are received from all the child nodes and* $Cpnode_j == NULL$*)* **then**
    Select the maximum quality factor from among the received quality factors and self quality factor.
    Make a $MQFM$ message using the information of the node with maximum quality factor and send it to the parent node.
    Wait for LDM message.
  end
end

Algorithm 13: Declaration of the leader

```
1  if (Node i is the election initiating node) then
2      if (Node i gets all the expected MQFM messages within the Tf_e-m time) then
3          Find the highest quality factor from among received quality factors and self quality factor.
4          Declare the node with highest quality factor as the elected leader.
5          If multiple nodes have the highest quality factor, then choose the node with minimum Id
             among them as the leader.
6          L_Id_i ← the elected leader Id
7          El_Id ← L_Id_i
8          if (El_Id == N_Id_i) then
9              Status_i ← leader
10         else
11             Status_i ← Non-leader
12         end
13         Create an LDM[El_Id] message and send it to all the adjacent nodes.
14         Ein_Id_i ← NULL
15     else
16         Check the aliveness of the adjacent nodes which did not send the MQFM message.
17         Collect the MQFM from the alive adjacent nodes.
18         Follow step 3 to step 14.
19     end
20 else
21     if (A node j gets an LDM[El_Id] message for the first time) then
22         L_Id_j ← El_Id
23         Ein_Id_j ← NULL
24         if (El_Id == N_Id_j) then
25             Status_j ← leader
26         else
27             Status_j ← Non-leader
28         end
29         Send the received LDM[El_Id] message to all the adjacent nodes except its sender node.
30     else
31         Discard the received LDM[El_Id] message.
32     end
33     if (A node j does not get a LDM message within the Tf_e-m time) then
34         The node j re-initiates the election.
35     end
36 end
```

of the proposed election method ensures that if node $x \in \Pi$ *and all* x *are connected through a network, then* $x \in S_p$.

Proof: Initially, $S_p = \phi$. A node participates in the election either by creating an EIM message or by receiving an EIM message and get a chance to become the leader. When a node i initiates an election by creating EIM and sends it to its all the adjacent nodes, the node i becomes the first participant of the election and an element of S_p. When the adjacent nodes of i receive the EIM, they participate in the election and each of them sends the EIM message to its adjacent nodes except the parent node and becomes an

element of S_p. In this way, the EIM message gets spread over the network. That means the broadcasting method is used to spread the EIM message over the network. On the other hand, according to the system model, a message gets transmitted from one node to another node within a bounded time period and all N nodes are connected through a network. So all the nodes in the network get the EIM message within a certain time period and become the elements of set S_p. After a certain time of election initiation, if $x \in \Pi$ then $x \in S_p$ and $|\Pi| = |S_p| = N$. That means all the nodes in the network participate in the election. ■

Lemma 6.2. *If S_{ec} represents the set of election conducting node Id, then a proper execution of the proposed election algorithm in a system consisting of n connected node ensures that $|S_{ec}| = 1$.*

Proof: Suppose S_{ei} is the set of election initiating node Id. Node $x \in S_{ec}$ if and only if $x \in S_{ei}$ and the node x declares the elected leader by creating a LDM message. When a node realizes the leader's failure, it creates an EIM message and initiates the election. Leader's failure can be detected by a single node or multiple nodes simultaneously. When only one node (suppose node i) detects the leader's failure, then only that node creates the EIM message and initiates the election by sending the EIM message to its all adjacent nodes. So, node $i \in S_{ei}$ and $|S_{ei}| = 1$. According to the proposed algorithm, the EIM message gets spread all over the network through a broadcasting method and a message gets transmitted from one node to another node within a bounded time period. Hence within a certain amount of time, the node i gets the $MQFM$ messages from its adjacent nodes and declares the node with the highest quality factor as the leader. So node $i \in S_{ec}$. In this case, only node i initiates the election and conducts the whole election. Hence $|S_{ec}| = 1$. Let's consider the scenario where multiple nodes initiate the election simultaneously. Suppose r number of nodes initiate the election simultaneously. So, $S_{ei} = \{x : x \in \Pi \textit{ and } x \textit{ creates an } EIM \textit{ message}\}$ and $|S_{ei}| = r$. In the proposed algorithm, if multiple nodes create EIM messages and initiate election simultaneously, the EIM message created by the node with minimum Id gets spread all over the network and others get discarded. According to the system model, every node has a distinct integer Id. So among all the election initiating nodes, there will be only one node with minimum Id. Suppose the node i has the minimum Id in S_{ei}. So the only node i will get the $MQFM$

messages from its all adjacent nodes and declare the leader by creating an LDM message. Hence node $i \in S_{ec}$ and $|S_{ec}| = 1$. That means only one node conducts an entire election process. ■

Lemma 6.3. *A proper execution of the proposed election algorithm in a distributed system consisting of N connected nodes elects only one node as the system leader.*

Proof: In the proposed algorithm, only one node conducts the whole election process (cf. Lemma 6.2). If a node initiates an election or gets an EIM message, it sets its *status* to $NULL$. Later on, when the election conducting node gets all the expected $MQFM$ messages from its adjacent nodes, it chooses the node with the highest quality factor among the received quality factors and its own quality factor as the leader. Here two cases may occur, 1) all the quality factors are distinct 2) multiple nodes have the same quality factor. In case 1, the election conductor chooses the node with highest quality factor and declares it as the leader by broadcasting the $LDM[El\_Id]$. When a node gets the $LDM[El\_Id]$ message it compares self Id with the El_Id. If both are same, the node sets its *status* to *leader*. Otherwise, it sets its *status* to *Non-leader*. In this case, all the node have distinct quality factor so among all the nodes only one node's *status* gets set to *leader* and each of the other nodes *status* gets set to *Non-leader*. In case 2, if multiple nodes have the highest quality factor, the node with the minimum Id among them is chosen and declared as the leader. As the node Ids are distinct integer and only one node conducts the election so a single node gets elected as the leader and like case 1 (among all the nodes only one node's *status* gets set to *leader* and each of the other nodes *status* gets set to *Non-leader*). Hence the proposed algorithm always elects only one node as the system leader. ■

Lemma 6.4. *Let the system consists of N connected nodes and Π represents the set of all nodes ($|\Pi| = N$) and S_{arg} represents the set of nodes that agree the elected leader. At the end of the election, if node $x \in \Pi$ then $x \in S_{arg}$.*

Proof: In Lemma 6.1, we proved that all the connected nodes of a distributed system participate in the election. According to the proposed algorithm, when a node initiates an election or gets an EIM message, it sets its L_Id to $NULL$. Later on every node shares

its quality factor through a $MQFM$ message. Finally, when the election conducting node gets higher quality factor nodes information through its adjacent nodes, the node with the highest quality factor gets elected as the leader. The election conducting node then declares the elected leader by broadcasting an $LDM[El\_Id]$ message. When a node i gets the $LDM[El\_Id]$ message, it accepts the elected leader by setting its L_Id to the Id of the elected leader and becomes an element of S_{arg}. As all the node are connected and the $LDM[El\_Id]$ message is broadcast over the network through a flooding method so all the nodes get the $LDM[El\_Id]$ message and become the elements of S_{arg}. Hence if node $x \in \Pi$ then $x \in S_{arg}$. That means all the nodes get to know about the elected leader and they agree with the elected leader. ■

Lemma 6.5. *A proper execution of the proposed election algorithm in a distributed system consisting of N connected nodes elects a leader in a finite time.*

Proof: Suppose the proposed algorithm takes δ_t time to elect a leader. We have already proved that at a time, only one node conducts the election (cf. Lemma 6.2). So election termination depends on the election conducting node. Here two cases may occur: 1) the election conducting node does not fail during the election 2) the election conducting node fails during the election. In case 1, the election gets terminated in $O(D)$ that means $\delta_t = O(D)$ where D is the diameter of the network. The maximum diameter of our considered network can be $N - 1$. Our system consists of N nodes, where N is a finite integer, so D is also finite. Hence in the first case, the algorithm elects the leader and gets terminated in a finite time. In the second case, if the election conducting node fails before creating the leader declaration message (LDM), then election re-initiation happens. The system has some stable nodes, and when one of these stable nodes gets the chance to conduct the election, one node is elected as the system leader because the stable node does not fail during the election. In the worst scenario, all the unstable nodes get the chance to conduct the election first then the stable nodes get the chance to conduct the election. The system is consisting of finite number of nodes so the number of unstable nodes is also finite. In this scenario, $\delta_t = O(N \cdot D)$ that means the proposed algorithm also elect the leader in finite time. ■

Theorem 6.6. *The proposed leader election algorithm is a self-stabilizing leader election algorithm.*

Proof: Lemma 6.3 proves that the proposed algorithm always elects a unique leader for the system. Lemma 6.4 proves that the algorithm satisfies the agreement condition and lemma 6.5 proves the termination condition of the algorithm. Hence the proposed algorithm is a self-stabilizing leader election algorithm. ■

6.2.4 Complexity analysis

The efficiency assessment of a distributed algorithm is done by three complexity measures, i.e., message complexity, time complexity, and bit complexity. In this section, first, we calculate these three complexity measures of the proposed algorithm for arbitrary network topology. Next, we calculate these complexity measures for the several regular network topologies and present them in a tabular form.

6.2.4.1 Message complexity

In the leader election algorithm, the message complexity refers to the number of transmitted messages to elect a leader for the system.

Best case: The proposed leader election algorithm is designed considering an arbitrary network topology consisting of N nodes and l links. Here, in the best-case, only one node initiates the election and conducts the whole election process. In this algorithm we use five types of messages i.e., EIM, ACK, AGM, $MQFM$ and LDM. Here, $O(l)$ EIM messages need to transmit to initiate the election. After that, at most $O(l)$ ACK messages and $O(l)$ AGM messages get transmitted as the response of the EIM messages. The proposed algorithm finds the relation between a node and its adjacent nodes by transmitting these three messages (EIM, ACK and AGM). On the other hand, $O(l)$ $MQFM$ messages are transmitted to send the information of the higher quality factors' nodes to the election initiator. Finally, $O(l)$ LDM messages are exchanged to declare the elected leader. Hence, in the best case, the message complexity of the proposed algorithm is $O(l)$.

Worst case: Suppose the election conducting node is not a stable node and it fails before declaring the elected leader. The considered distributed system is consisting of

at least one stable node. In the worst case, each of the $N-1$ unstable nodes gets the chance to conduct the election first then the stable node gets the chance to conduct the election. Unlike unstable node when a stable node conducts the election, it can complete the entire election process and elect a leader. When one node initiates the election, the message complexity is $O(l)$. In this scenario, after $N-1$ incomplete elections, one complete election happens. So maximum $O(N \cdot l)$ messages are required to elect the leader. Hence in the worst case the message complexity of the proposed algorithm is $O(N \cdot l)$.

6.2.4.2 Time complexity

The time complexity of a leader election algorithm refers to the time required by the algorithm to elect a leader for the system.

Best case: In this case, only one node initiates and conducts the whole election process. the EIM message created by the election conducting node gets spread all over the network. After that, every node sends a $MQFM$ message to its parent node. Finally, when the election conducting node gets the $MQFM$ messages from adjacent nodes, it declares the new system leader by creating and sending a LDM message. The EIM message takes $O(D)$ time to get spread all over the network, where D is the diameter of the network. After initiating the election, the election initiator gets the $MQFM$ messages from its all the adjacent nodes in $O(D)$ time. The LDM message also takes $O(D)$ time to get spread all over the network. So total $O(D) + O(D) + O(D)$ time is required to elect a leader. That means the complexity of the proposed algorithm is $O(D)$.

Worst case: In this case, after $N-1$ incomplete elections, one complete election happens. One complete election process takes $O(D)$ time. So $N-1$ consecutive incomplete elections and one complete election can take maximum $O(N \cdot D)$ time. Hence, in the worst case the time complexity of this algorithm is $O(N \cdot D)$.

6.2.4.3 Bit complexity

In a distributed algorithm, nodes exchange information with their neighbors through message passing. Bit complexity refers to the number of bits contained in the message. In

TABLE 6.1: Time complexity and message complexity in the best case and worst-case scenarios of the proposed algorithm in different regular network topologies.

	Message complexity		Time complexity	
Topology	Best case	Worst case	Best case	Worst case
Ring	$O(N)$	$O(N^2)$	$O(N)$	$O(N)$
Bus	$O(N)$	$O(N^2)$	$O(N)$	$O(N)$
Full-mesh	$O(N^2)$	$O(N^2)$	$O(1)$	$O(1)$
2D torus	$O(N)$	$O(N\sqrt{N})$	$O(\sqrt{N})$	$O(\sqrt{N})$
Star	$O(N)$	$O(N)$	$O(1)$	$O(1)$
k-ary tree	$O(N)$	$O(N^2)$	$O(\log_k N)$	$O(\log_k N)$

the proposed leader election algorithm, five types of messages i.e., EIM, ACK, AGM, $MQFM$ and LDM are used to elect the leader. Each of EIM, ACK, and AGM contains two node Ids, LDM contains a single node Id, and $MQFM$ contains quality factor of a node and two node Ids. If the system is consists of n nodes, then $\log_2 N$ bits are required to represent a node Id uniquely. On the other hand, quality factor of a node takes constant number of bits. So the bit complexity of the proposed algorithm is $O(\log_2 N)$.

Now we calculate the message complexity and time complexity of the proposed leader election algorithm for several regular network topologies and present them in Table 6.1. We do not include the bit complexity of the proposed algorithm in this table because, for any network topology, the bit complexity is $O(\log_2 N)$.

6.2.5 Illustrative Example

To illustrate the algorithm, we consider a distributed system consisting of 7 nodes and 9 edges (cf. Figure 6.1 (a)). In Figure 6.1 (a), a circle represents a node, the integer inside the circle represents the node Id and the fractional number outside the circle represent the quality factor (qf) of a node. First we consider the scenario where no link and node failure occur during the election. Suppose node 2 initiates the election. So node 2 creates an election-initiating message ($EIM[2, 2]$) and sends it to its adjacent nodes i.e., nodes $0, 1$ and 3 (as shown in Figure 6.1 (b)). After receiving the $EIM[2, 2]$, each of nodes $0, 1$ and 3 creates an acknowledgement message i.e., $ACK[2, 0]$, $ACK[2, 1]$ and $ACK[2, 3]$ respectively and sends the acknowledgement to node 2. Nodes $0, 1$ and 3 consider node 2 as

their parent node and store Id 2 in $Pnode_0$, $Pnode_1$ and $Pnode_3$ respectively. Besides, each of these three nodes replaces the second field of the received election-initiating message with their own Id and sends the modified election-initiating message i.e., $EIM[2, 0]$, $EIM[2, 1]$ and $EIM[2, 3]$ to their adjacent nodes (node 0 sends to nodes 3 and 6, node 1 sends to node 5, and node 3 sends to nodes $0, 4$ and 5 (as shown in Figure 6.1 (c))).

When node 2 gets the ACK messages from its adjacent three nodes i.e., nodes $0, 1$ and 3, it consider them as the child nodes and store their Ids in the $Cnode_2$. On the other hand, after getting the $EIM[2, 0]$ from node 0, the node 6 creates $ACK[2, 6]$, sends it to node 0, considers node 0 as its parent node and stores Id 0 in $Pnode_6$. Then it replaces the second field of the received election-initiating message with its own Id and sends this modified election-initiating message i.e., $EIM[2, 6]$ to node 5. When node 5 gets the $EIM[2, 1]$ message from node 1, it creates $ACK[2, 5]$ and sends it to node 1. The node 5 considers node 1 as its parent node and stores Id 1 in $Pnode_5$. Then it replaces the second field of the received election-initiating message with its own Id and sends the modified $EIM[2, 5]$ to nodes 3 and 6. When node 0 and node 3 get election-initiating message (i.e., $EIM[2, 3]$ and $EIM[2, 0]$ respectively) from each other, they create $AGM[2, 0]$ and $AGM[2, 3]$ respectively and send their created AGM message to each other because they already got the same election-initiating message from node 2. Then node 0 and node 3 include the node Id 3 and node Id 0 in the $Cpnode_0$ and $Cpnode_3$, respectively because they consider each other as their co-parent node. When node 4 gets the $EIM[2, 3]$ message, it considers node 3 as its parent node. It then creates an $MQFM[0.60, 4, 4]$ message and sends it to its parent node i.e., node 3 because node 4 is a leaf node (as shown in Figure 6.1 (d)).

When node 5 and node 3 get EIM message from each other, they act in the same way as node 0 and node 3 acted when they got the EIM message from each other. Likewise, when node 5 and node 6 get EIM message from each other, they also act in the same way as node 0 and node 3 acted when they got the EIM message from each other (as shown in Figure 6.1 (e)).

When node 3 gets $AGM[2, 5]$ from node 5, it gets to know that it has two co-parent nodes i.e., nodes 0 and 5 and one child node i.e., node 4. The node 3 has already got the

expected $MQFM$ message from node 4. So node 3 chooses the node between node 4 and itself whoever has the higher quality factor. As node 3 itself has the higher quality factor, it creates $MQFM[0.94, 3, 3]$ message and sends it to nodes 0 and 5. On the hand, after getting $AGM[2, 3]$ and $AGM[2, 6]$ from node 3 and node 6 respectively, the node 5 gets to know that it has two co-parent nodes i.e., nodes 3 and 6 and no child node. So it creates $MQFM[0.69, 5, 5]$ message (using self-information) and sends it to nodes 3 and 6. After receiving $AGM[2, 5]$ message from node 5, the node6 gets to know that it has only one co-parent node and no child node. So it also creates $MQFM[0.74, 6, 6]$ message (using self-information) and sends it to node 5 (as shown in Figure 6.1 (f)).

When node 6 receives $MQFM[0.69, 5, 5]$ from node 5, it learns that it has the higher quality factor between node 5 and itself. So it sends $MQFM[0.74, 6, 6]$ message to its parent node i.e., node 0. On the other hand, when node 5 gets expected two $MQFM$ messages from nodes 3 and 6, it finds that node 3 has the maximum quality factor among the nodes $3, 6$ and itself. So node 5 creates $MQFM[0.94, 3, 5]$ and sends it to its parent node i.e., node 1 (as shown in Figure 6.1 (g)).

When node 0 gets the $MQFM$ from its only child node i.e., node 6, it creates $MQFM[0.74, 6, 0]$ because 0.74 is the maximum quality factor between the quality factor received through the $MQFM$ message from its child node and its own quality factor. Then it sends $MQFM[0.74, 6, 0]$ to its single co-parent node i.e., node 3. After getting the $MQFM[0.74, 6, 6]$ message, the node 0 got all the expected $MQFM$ messages from its child node and co-parent node and 0.94 is the maximum quality factor among all the received quality factor and its own quality factor, so it sends $MQFM[0.94, 3, 0]$ to its parents node i.e., node 2. Node 1 also sends the $MQFM[0.94, 3, 1]$ to its parent node i.e., node 2 (as shown in Figure 6.1 (h)). Likewise, when node 3 receives $MQFM$ message from node 0, it sends the $MQFM[0.94, 3, 0]$ to node 2 (as shown in Figure 6.1 (i)).

Finally, when the node 2 gets all the expected $MQFM$ messages from its adjacent nodes, it finds that node 3 has the highest quality factor among all the node in the network. So node 3 is elected as the new system leader. To declare the elected leader, the node 2 creates an $LDM[3]$ message and sends it to its all the adjacent nodes (as shown in Figure 6.1 (j)). After that nodes $0, 1$ and 3 send the $LDM[3]$ message to all of their adjacent

nodes except their parent node (as shown in Figure 6.1 (k)). Thus all the node gets to know about the newly elected system leader.

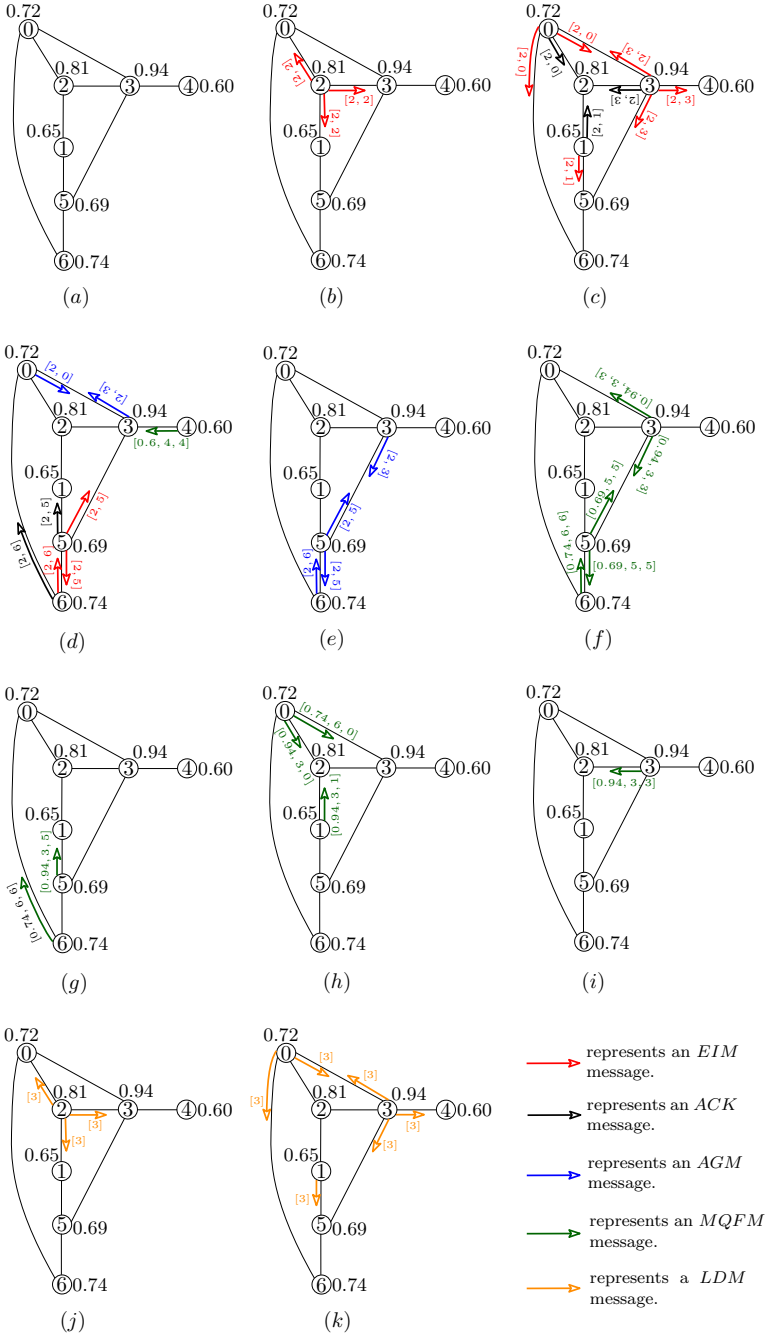

FIGURE 6.1: (*a*) A distributed system consisting of 7 nodes and 9 edges. (*b*), (*c*), (*d*), (*e*), (*f*), (*g*), (*h*), (*i*), (*j*) and (*k*) are the different steps of the proposed leader election algorithm to elect the leader for the system.

Now we consider link and node failures during the election and try to determine whether the proposed algorithm can elect the leader in the presence of link and node failures.

First, we consider single link failure, then we consider multiple link failures, and at last, we consider node failure.

Let's assume when node 3 receives the EIM message from node 2, the link between node 3 and node 2 fails. In this case, the $MQFM$ message of node 3 is transmitted to node 2 via node 0 as well as via nodes 5 and 1, and node 3 gets elected as the leader. We find that the proposed algorithm can tolerate any single link failure of a node during the election to elect the system leader. If a link failure divided a network into two parts, the proposed election algorithm elects a leader for each part. When the link between node 3 and node 4 fails, node 4 becomes an isolated node, and the network gets divided into two parts (as shown in Figure 6.2 (a)). In this case, node 4 (one part of the network) becomes an isolated node, so no need of leader election for this isolated node. On the other hand, node 3 is elected as the leader for the other part (consisting of nodes $0, 1, 2, 3, 5$ and 6) of the network.

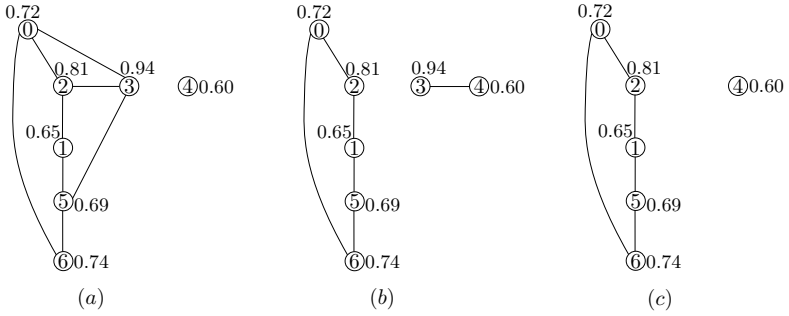

FIGURE 6.2: Node and link failures in the network.

Now we consider the case where multiple links of a node may fail during the election. Let's assume when node 3 receives the EIM message from node 2, the links between nodes 3 and 2, and nodes 3 and 0 fail. In this case, node 2 does not get the ACK message from node 3, so node 2 does not consider node 3 as its adjacent node anymore. However, the $MQFM$ message of node 3 is transmitted to node 2 via nodes 5 and 1 and node 3 gets elected as the leader. Instead of two links, if three links of node 3 (i.e., the links between nodes 3 and 2, nodes 3 and 0, and nodes 3 and 5) fail, the network gets divided into two parts. One part contains nodes $0, 1, 2, 5$ and 6 and another part contains nodes $3, 4$ (as shown in Figure 6.2 (b)). In this case, node 2 does not get the ACK message from node 3, so node 2 does not consider node 3 as its adjacent node anymore. As nodes 3 and 4 get

disconnected from the network, node 2 does not get the $MQFM$ message of node 3. In this situation node 2 is elected as the leader of one part (containing nodes $0, 1, 2, 5$ and 6) and broadcast leader declaration message ($LDM[2]$). However, nodes 3 and 4 do not get this $LDM[2]$ message. After waiting a certain amount of time ($Tf_{e\text{-}m}$) when node 3 does not get the LDM message, it re-initiates the election and node 3 is elected as the leader of the another part (consisting of nodes 3 and 4) of the network.

Now we consider node failure and discuss the node failure tolerability of the proposed algorithm. If one or more nodes among nodes $0, 1, 4, 5$ and 6 fail during the election, node 3 is elected as the system leader. If node 3 fails, node 4 gets isolated from the network and the network gets divided into two parts. One part contains nodes $0, 1, 2, 5$ and 6 and another part contains only node 4 (as shown in Figure 6.2 (c)). In this situation, node 2 becomes the system leader. On the other hand, after creating the EIM message if the election initiating node (node 2) fails, the LDM message is not created. So, after sending the $MQFM$ messages other nodes wait for LDM message. After waiting a certain amount of time ($Tf_{e\text{-}m}$), when a node does not get an LDM message, it re-initiates the election.

6.3 Experiment and result analysis

Nowadays, replicated cloud database is becoming popular as it increases data availability and reliability, speeds up the query evaluation, reduces the load, makes a system more fault-tolerant and improves its performance. Multiple replicated data servers located at different places get connected through a network and build a replicated cloud database. Cloud Spanner [32] is a globally distributed and synchronously replicated cloud database developed by Google. In a replicated distributed database, data inconsistency [22] [87] is a significant issue, and maintaining data consistency is challenging. Generally, at the time of data modification in the servers, data inconsistency happens. The concept of leader election is a way to handling the data inconsistency. Here, one of the servers is elected as the leader to coordinate all the servers' data modification activities. Now we conduct an experiment and show how a good quality leader elected by the proposed leader election method helps improve the overall system performance.

This experiment considers three arbitrary network topologies consisting of eight, ten, and twelve nodes (as shown in Figure 6.3). Here a node works as a replicated server. First, we consider eight replicated servers connected through an arbitrary network topology and elect one of the servers as the leader using the proposed election method. Next, we add two replicated servers through random connection with these eight servers and implement the proposed election method to elect the leader. In the same way, we add two more servers through random connection with the existing ten servers and elect the leader. For this experiment, we involved six experts in identifying the important attributes of the node that should consider electing the leader for a replicated distributed system. The experts identified five main attributes for our considered system. They are Closeness centrality (Cc), Degree (Deg), Failure rate (Fr), CPU capacity (CPU) and Available memory (Am) where, Closeness centrality, Degree, CPU capacity and Available memory are the benefit attributes, and Failure rate is the cost attribute. They assigned $0.28, 0.25, 0.22, 0.15$ and 0.10 as the weight of the Closeness centrality, Degree, Failure rate, CPU capacity and Available memory, respectively. During the data modification, the user cannot access the data. Hence to improve the system performance, data modification needs to perform as quickly as possible. It is possible if the elected leader's closeness centrality and degree are comparatively high. That is why the experts gave the highest priority and second-highest priority to the Closeness centrality and degree, respectively. All the system's collaborative works get halted during the election process due to the lack of coordination. Due to the frequent leader failure may hamper the system performance. If an elected leader fails frequently, the system needs to invoke the election algorithm again and again in electing the leader. In this situation, the system spends much time in leader election that hampers the system performance, so frequent leader failure is not expected. That is why the experts gave the third-highest priority to the failure rate. On the other hand, the leader with moderately high CPU capacity and moderately high available memory can easily handle the data modification task in a replicated distributed system. That is why the expert gave the least priority and second-least priority to the Available memory and CPU capacity. The weight, the minimum value and the maximum value of each attribute is given in Table 6.2 and all the nodes' information is given in Tables 6.3, 6.4 and 6.5.

We have conducted all the experiment in a single machine and it was set up with Ubuntu

TABLE 6.2: The weight given by the experts, the minimum value and the maximum value of the attributes.

	CPU	Am	Fr	Cc	Deg
Weight	0.15	0.10	0.22	0.28	0.25
Minimum value	1.8	0	0	0	1
Maximum value	6	64	1	1	12

TABLE 6.3: Information of all the nodes of the system consisting of 8 nodes and 9 links.

Node Id	CPU	Am	Fr	Cc	Deg	Qf
0	4.9	26	0.07	0.46	1	0.3492
1	2.9	28	0.06	0.50	2	0.3469
2	4.1	24	0.03	0.77	5	0.3637
3	3.9	22	0.08	0.58	3	0.3516
4	3.8	22	0.06	0.58	3	0.3523
5	3.0	32	0.07	0.36	1	0.3424
6	3.2	24	0.01	0.38	1	0.3455
7	3.6	24	0.06	0.53	2	0.3491

Linux Release 16.04 (xenial kernel 4.4) operating system, Intel Core(TM) i5-2410M (2.3 GHz, 4 MB cache, 2.9 GHz Turbo Boost) processor, NVIDIA GeForce graphics, 8 GB DDR3 RAM and 1 TB hard disk drive. We used python 3.6 as programming language and mpi4py as Message Passing Interface (MPI).

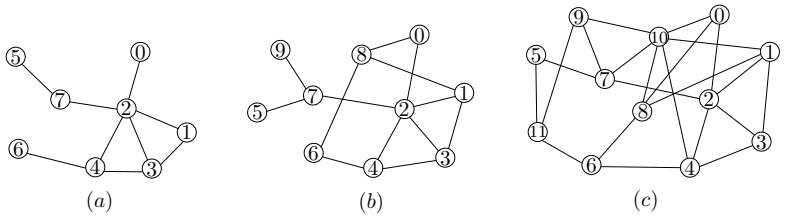

FIGURE 6.3: (a) eight replicated servers connected through an arbitrary network topology, (b) ten replicated servers connected through an arbitrary network topology and (c) twelve replicated servers connected through an arbitrary network topology

When we implement the proposed leader election method on the system consisting of 8 nodes (cf. Figure 6.3 (a)), node 2 is elected as the system leader. Node 2 has the highest closeness centrality, highest degree, second lowest failure rate, second highest CPU capacity and moderate available memory (cf. Table 6.3). Let us assess the quality of the elected leader to maintain the system performance and consistency. We know that the data modification in replicated distributed database consisting of multiple replicated servers is

TABLE 6.4: Information of all the nodes of the system consisting of 10 nodes and 13 links.

Node Id	CPU	Am	Fr	Cc	Deg	Qf
0	4.9	26	0.07	0.50	2	0.3518
1	2.9	28	0.06	0.53	3	0.3493
2	4.1	24	0.03	0.69	5	0.3614
3	3.9	22	0.08	0.52	3	0.3500
4	3.8	22	0.06	0.52	3	0.3508
5	3.0	32	0.07	0.36	1	0.3424
6	3.2	24	0.01	0.42	2	0.3478
7	3.6	24	0.06	0.52	3	0.3504
8	3.7	20	0.05	0.45	3	0.3490
9	3.5	22	0.06	0.36	1	0.3431

TABLE 6.5: Information of all the nodes of the system consisting of 12 nodes and 22 links.

Node Id	CPU	Am	Fr	Cc	Deg	Qf
0	4.9	26	0.07	0.52	3	0.3539
1	2.9	28	0.06	0.55	4	0.3515
2	4.1	24	0.03	0.61	5	0.3592
3	3.9	22	0.08	0.50	3	0.3495
4	3.8	22	0.06	0.57	4	0.3538
5	3.0	32	0.07	0.44	2	0.3457
6	3.2	24	0.01	0.55	3	0.3526
7	3.6	24	0.06	0.57	4	0.3534
8	3.7	20	0.05	0.57	4	0.3538
9	3.5	22	0.06	0.55	3	0.3507
10	4.3	26	0.02	0.68	6	0.3645
11	3.3	28	0.03	0.47	3	0.3503

handled by the leader server. When a data modification is required, the leader sends a *lock message* against that data to all the servers and waits for those servers' response. The leader sends the *lock message* so that nobody (users or clients) can access that data during the data modification. When the leader gets all the servers' responses regarding the *lock message*, it sends the *data modification message* to all the servers. The *data modification message* tells what modification needs to be performed on the data. After data modification, every server sends a *modification completion message* to the leader. When the leader gets the *modification completion messages* from all the servers, it sends an *unlock message* to all the servers to make the data accessible for the users. Suppose a

message takes c unit time (where c is constant) to traverse one hop (one node to another adjacent node). If node 2 is elected as the leader, then $10c$ unit time is required to modify a data. On the other hand, if any other node except node 2 is elected as the leader, atleast $15c$ unit time is required to modify a data. In this system, instead of any other node, if node 2 is elected as the leader, the data modification can be performed atleast 1.5 times faster way.

As the leader coordinates all the system's cooperative activities, all the nodes or servers need to communicate with the leader frequently. The proposed election method elects a node with a higher degree. The higher degree of the leader node increases the communication path diversity between the leader and the other nodes in the system. The higher path diversity decreases the communication traffic between the leader node and other nodes. In the system (as shown in Figure 6.3 (a)), node 2 gets elected as the leader with highest degree that helps in reducing communication traffic between the leader and other servers. This helps to increase the overall system performance. The election method also elects a node with a moderately low failure rate. Node 2 has the second lowest failure rate among all the nodes. So the leader failure probability is also low. That means the proposed election method elects a leader who will live for a long time in the system. All the system's collaborative works get halted during the leader election process due to the lack of coordination. As the aliveness probability of the elected leader is high, the system does not need to execute the leader election algorithm repeatedly, which helps to increase the system's performance. On the other hand, the proposed algorithm elects a leader with moderately high CPU capacity and available memory that also takes care of the system performance.

We also observe that when we prioritize a particular attribute by assigning a much higher weight than other attributes, then the node with the best value of that attribute gets elected as the leader. To prioritize an attribute, we assign 0.8 as the weight of that attribute and 0.05 as the weight of every other attribute. In this way, when we prioritize the CPU capacity, Available memory, and Failure rate, then node 0, node 5, and node 6 get elected as the leader, respectively. On the other hand, when we prioritize the Closeness centrality and Degree, then node 2 gets elected as the leader. The proposed algorithm also elects node 2 as the leader. Let's compare these four nodes (nodes $0, 2, 5$ and 6) to

find out which one is most suitable as the system leader of the above-mentioned system. Though node 0 has the highest CPU capacity and moderate available memory, its closeness centrality and degree are comparatively low and its failure rate is comparatively high. If node 0 is elected as the leader, it will take $15c$ unit time to modify the data, whereas node 2 takes $10c$ unit time to modify the data. The degree of node 0 is 1 whereas the degree of node 2 is 5. So node 2 can handle its communication traffic in a better way than node 0. The failure rate of node 2 is much lower than that of node 0. That means node 2 has a higher probability of aliveness in the system than node 0. When we compare node 2 and node 5, we know that node 5 has the highest available memory, but regarding all other attributes, node 2 is better than node 5. On the other hand, regarding the failure rate node 6 is better than node 2 but for the other attributes node 2 is better than node 6.

Likewise, when we implement the proposed leader election method in the systems consisting of ten nodes (cf. Fig 6.3 (b)) and twelve nodes (cf. Fig 6.3 (c)), node 2 and node 10 get elected as the system leader, respectively. Here, these two systems also get similar benefits from their elected leader, like the system consisting of eight nodes benefited by electing node 2 as the system leader.

Now we compare the performance of the proposed algorithm and the existing PALE algorithm. This performance comparison is based on the number of exchanged messages and the time required to elect a leader in the best-case and worst-case scenarios. We consider Five different networks consisting of 10, 15, 20, 25 and 30 nodes and implement the proposed election algorithm and PALE algorithm to elect the leader. We create each of these five networks by connecting the nodes randomly. For the PALE algorithm we consider $maxRatio = 3$ (the authors also consider $maxRatio = 3$ in their paper).The number of exchanged messages and the time these two algorithms take to elect the leader is shown in Figure 6.4. Figure 6.4 (a) and (b) show that the proposed algorithm exchanges fewer messages than the PALE algorithm to elect the system leader in the best-case and worst-case scenarios. On the other hand, Figure 6.4 (c) and (d) tell that the proposed algorithm takes less time than the PALE algorithm to elect the system leader in best-case and worst-case scenarios.

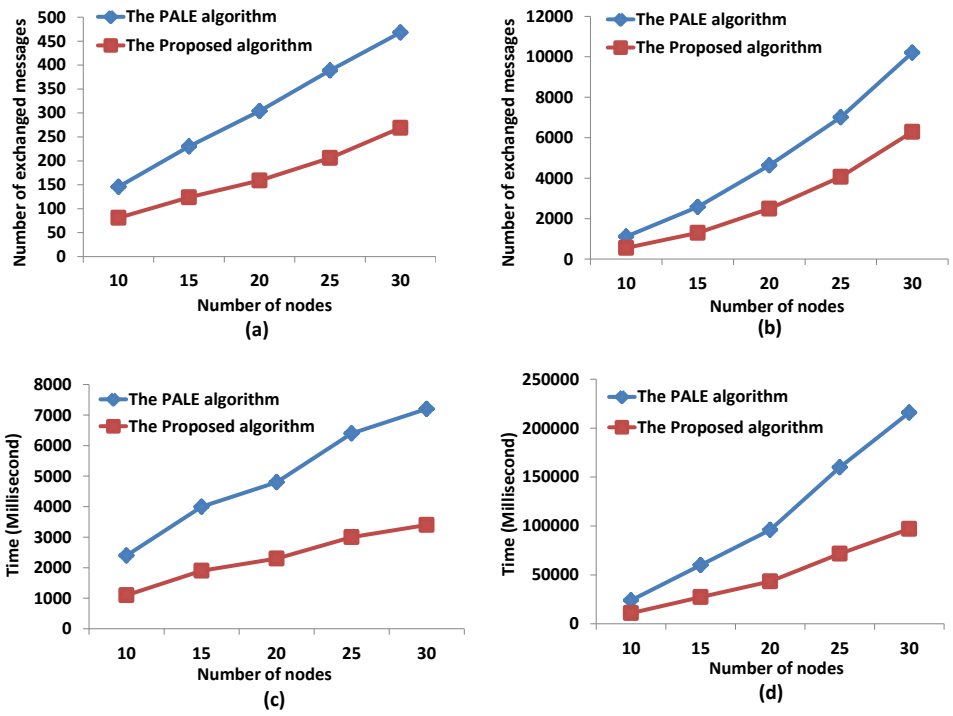

FIGURE 6.4: Performance comparison of the PALE algorithm and the proposed algorithm. (*a*) and (*b*) show the required time to elect the leader in the best case and worst case, respectively. (*c*) and (*d*) show the number of exchanged messages to elect the leader in the best case and worst case, respectively.

6.4 Summary

This chapter introduced a leader election method to elect a good quality leader (according to the system requirements) for a dynamic and partially asynchronous distributed system with weak assumptions. Here, we involved a group of experts and used the concept of Multi-attribute decision-making (MADM) to elect a suitable leader for the system. The experts play a vital role in pointing out the necessary attributes of the nodes and their weight according to their importance to elect the leader. Once the attributes and weights are determined, a node calculates its quality factor using the proposed modified TOPSIS method. The proposed leader election algorithm then elects the node with the highest quality factor as the system leader. We proved that the proposed election algorithm satisfies the *Uniqueness, Agreement, and Termination* conditions which means the proposed algorithm is a self-stabilizing one. So the proposed algorithm can maintain the system

consistency that helps the system complete its tasks successfully. According to the system requirements, our election method elects a good quality leader for the system, so the elected leader helps to improve the system resource utility, reliability, fault tolerability, and overall performance. That directly helps improve the performance of the applications of the various fields that adopt this system. The simulation results show that the algorithm reduces the time and message overhead to elect a suitable leader.