Chapter 7

Conclusion and Future Directions

This chapter summarizes the research works done in the previous chapters of this thesis and highlights the significant contributions and findings. This chapter also gives a few future directions for the researchers by explaining open research challenges of the leader election problem and its application in modern technologies.

7.1 Conclusion

In this thesis, we have worked on the leader election problem in the distributed systems and proposed four new leader election algorithms to overcome several research gaps. Chapter 1 of this thesis described the leader election problem and showed how it plays an important role in distributed systems. Further, this chapter detailed the motivation behind the work. This chapter also lists the objectives of the thesis and contributions to the thesis. Chapter 2 provides preliminaries of the leader election problem and a detailed literature review of the existing methods of leader election. Based on the network topology of the distributed systems, we divide the existing leader election algorithms into two categories i.e., the algorithms for regular network topology and the algorithms for arbitrary network topology. Then we study the algorithms designed considering various distributed system models for these two categories and try to find the research gaps. Next, we propose,

analyze and characterize four different self-stabilizing leader election methods in Chapter 3, Chapter 4, Chapter 5, and Chapter 6 to overcome those research gaps.

Our literature review found that no leader election algorithm is designed considering a crash-recovery failure model for the ring network. Most existing algorithms considered that node or link failures or recovery do not occur during the election. However, in a practical scenario, node or link failures or recovery can occur during the election. In chapter 3, we propose a self-stabilizing leader election algorithm entitled "FRLLE: A Failure Rate and Load-based Leader Election Algorithm" for a crash-recovery bidirectional ring network. The proposed algorithm elects a node with a minimum failure rate and minimum load as the system leader. That is why the system gets a reliable leader which can comfortably concentrate on leadership roles and activities. In the best case, both the message complexity and the time complexity of the existing algorithms for the ring network is $O(N)$ (where N is the number of nodes in the network). In contrast, both the proposed algorithms' message complexity and time complexity are $O(1)$. So, in the best case, we have reduced both the message complexity and the time complexity of the election process from $O(N)$ to $O(1)$. The simulation results and complexity analysis also show that in the worst case, the message and time overhead of the FRLLE algorithm is less than any other existing algorithms designed for the ring network except the HS algorithm. In the worst case, if the number of nodes in the system is more than 230, the HS algorithm exchanges fewer messages than the FRLLE algorithm, but it is near about 6 times slower than the FRLLE algorithm. The FRLLE algorithm satisfies the uniqueness, agreement, and termination conditions that make it a self-stabilizing leader election algorithm.

From the literature survey, we found that only two leader election algorithms are designed for the 2D torus network, which can tolerate very few link failures, and their message and time overhead are pretty high. The authors did not consider node failures to design these two algorithms. We also found that no lower bound message complexity is given in the 2D torus network for the leader election problem. In chapter 4, first, we proposed a lower bound $\Omega(N \log_3 N)$ of message complexity on a comparison-based leader election for a $2D$ torus network (where N is the number of nodes in the network). Next, we sketched a new leader election algorithm (Lea-TN) considering both the node and link failures for a $2D$ torus network. This Lea-TN is a deterministic and self-stabilizing algorithm that elects

a leader for a partially synchronous distributed system. The algorithm chooses a leader, even when there are some link or node failures in the system. We have proved that in some scenarios the Lea-TN algorithm can tolerate $(N - 2\sqrt{N})$ (if N is even) or $(N - 2\sqrt{N} - 3)$ (if N is odd) links failures. We consider the number of non-faulty links and the subsisting nodes' failure rate to elect a reliable leader. We introduce new message-sending patterns that help reduce the number of exchanged messages and the execution time of the election process. This algorithm enables a node to identify its link failures during the election also. The complexity analysis and the simulation results show that in both the best and worst cases, the Lea-TN algorithm's message and time overhead is lesser than the existing algorithms designed for the 2D torus network.

The literature survey says that no leader election algorithm is designed considering a distributed real-time system. In chapter 5, we design a leader election algorithm for a distributed real-time system. Here, we introduce the concept of the primary leader and the provisional leader that helps to an instant selection of a leader. The proposed algorithm identifies r comparatively higher potential leader capable nodes in the system and designates the highest potential node among them as the primary leader. The other $r - 1$ higher potential leader capable nodes are kept in reserve so that while the primary leader fails, another leader capable node from these nodes can be selected instantly. To reduce the time complexity and the message complexity of the election process, based on the eccentricity of the nodes, we divide a distributed system into two layers (i.e., inner-layer and outer-layer). Only the inner-layer nodes take part to identify the list of potential nodes. We introduce the concept of the quality-coefficient to identify the potential nodes of the system. The quality-coefficient of a node is calculated by combining the eccentricity, processing capacity, Memory capacity, Degree, and Eccentricity of the node. This algorithm always tries to elect a node with the highest quality-coefficient as the leader so that the system gets a high quality leader. The algorithm proposed herein satisfies the uniqueness, agreement, and termination conditions that make it a self-stabilizing one. We also simulate the proposed algorithm on several arbitrary network topologies and compare the results with the well-known existing algorithms to evaluate the algorithm's performance. The simulation results show that the proposed algorithm exchanges fewer messages and takes less time than the other well-known leader election

algorithm to elect a leader for the system. It can also tolerate multiple links and node failures.

A good quality leader elected according to the system requirement can help to improve a distributed system's resource utility, reliability, fault tolerability, and overall system performance that directly improve the performance of the applications of the various fields that adopt this system. No leader election algorithm has been designed that considers the system requirements to elect a good quality leader. In chapter 6, to elect a good quality leader (according to the system requirements) for the system, we proposed a multi-attribute based self-stabilizing leader election method for a dynamic distributed system. The involvement of the experts and the concept of the quality factor of the nodes help in electing a good quality leader. First, based on the system requirements, a group of experts identifies the quality attributes of the nodes for electing a suitable leader and assign weight according to their importance. Next, we modify the TOPSIS MCDM method to calculate the quality factor of every node, and based on the quality factor, the leader is elected for the system. Here, we give an illustrative example of the proposed election method and show that the algorithm is self-stabilized and can tolerate multiple nodes and link failures. Further, we analyze the time complexity, message complexity, and bit complexity of the proposed algorithm. We simulate the proposed election method and compare it with the existing methods to evaluate and validate the proposed method's performance and the elected leader's quality.

7.2 Future Research Directions

For the several benefits of distributed computing, new evolving technologies such as cloud computing, blockchain, IoT, edge computing, machine learning are interested in adopting the concept of distributed computing. When these new technologies use the distributed system, we can explore the applicability of proposed leader election algorithms for these new technologies to manage, synchronize, and utilize the distributed resources efficiently to improve their performance as part of future research. The applicability of the leader election idea may also be explored for ensuring systems fairness and cost-effectiveness under blockchain arrangement.

Recently, the concepts of Distributed Machine Learning (DML) and Distributed Deep Learning (DDL) are getting more popular, and they are the combinations of machine learning and distributed computing. Here multiple nodes work simultaneously on a vast dataset to train a machine learning model in a distributed manner. Distributed computing provides a fault-tolerant, reliable, and large-scale robust computational platform with Big Data handling support for machine learning or deep learning, decreasing the training time. However, coordination among the nodes, consistency maintenance, communication overhead, resource management, etc., are significant issues of a truly distributed model, making it challenging to implement the concept of DML and DDL. In the DML and DDL, two types of parallelism are common, i.e., data parallelism and model parallelism. In data parallelism, the data is partitioned as per the number of nodes in the system. All nodes apply the same algorithm to different partitions of data. On the other hand, in the case of model parallelism, exact copies of the entire data are processed by the nodes, which operate on different parts of the model. In both data parallelism and model parallelism cases, when the nodes complete the assigned tasks, the results are aggregated to get the ultimate result. To improve the performance, learning accuracy, fault-tolerability, reliability, and reduce the learning time, we can explore the new challenges of distributed machine learning and deep learning and design new leader election algorithms to distribute learning tasks and aggregate the results.