Chapter 1

Introduction

"What is not started will never get finished". - Johann Wolfgang von Goethe

Now we are in the era of distributed computing [67] [108], and this thesis is concerned with an important issue known as leader election in distributed computing. Every day, knowingly or unknowingly, we use the distributed computing and enjoy its benefits. For example, when someone accesses the internet and performs a Google search, s(he) uses the distributed computing. From scientific research to everyday applications, everywhere we are using distributed computing. It has become an integral part of our professional life as well as personal life. The distributed computing is an area of computer science that makes use of distributed systems for orchestration of distribution of the computing load, aggregation of computing results, and presenting a single system image for processes and users. A distributed system is a collection of independent computers or nodes located in different places, interconnected by a network, and work together to achieve a common goal. It appears to its users as a single coherent system. Figure 1.1 shows a classical architecture of a distributed system. Distributed computing helps improve the performance of large-scale projects by combining the power of multiple machines. In this computing paradigm, to complete an enormous task fast, the task is partitioned into a set of possible sub-tasks and allocated to multiple system nodes. Then the nodes execute their assigned sub-tasks in a concurrent and consistent manner to complete the task in a faster way. On the other hand, it is much more scalable and allows users to add nodes according to growing workload demands. Since the nodes of a distributed system are placed in multiple geographical locations, the system can serve its user request through the closest node, resulting in low latency and better performance. In other words, the distributed systems are easily scalable, cost-effective, fault-tolerant, highly available, highly reliable, and provide high-performance computing environment. Nowadays, for harnessing such benefits of the distributed systems, people are very much inclined to use distributed computing. Several fields such as the banking system, e-learning platforms, manufacturing industry, artificial intelligence, e-commerce, transport, health care, agriculture, defense system are adopting the distributed computing. Distributed system architectures are also shaping many business areas and providing countless services with ample computing and processing power. Recently, some modern technologies like Blockchain, IoT, Cloud Computing, Machine Learning (ML), and Deep Learning (DL) are also adopting the benefits of distributed computing. Handling massive growth of data, led by rapid development in technologies, to train a model in a short time, machine learning and deep learning require a large-scale computational platform. ML and DL fulfill the need for a large-scale computational platform using distributed computing and emerges the concept of Distributed Machine Learning (DML) and Distributed Deep Learning (DDL) to handle massive data efficiently. In DML or DDL, multiple nodes work simultaneously on a vast dataset to train a machine learning model in a distributed manner. The distributed computing can provide a fault-tolerant, reliable, and large-scale computational platform with Big Data handling support for machine learning or deep learning, decreasing the training time. On the other hand, cloud computing is a distributed computing paradigm that provides computing services to the users, cloud based database platforms, virtual machines, cloud storage, etc [9].

Though the distributed systems provide various facilities, however, several issues are associated with these systems. Some of them are:

- System management

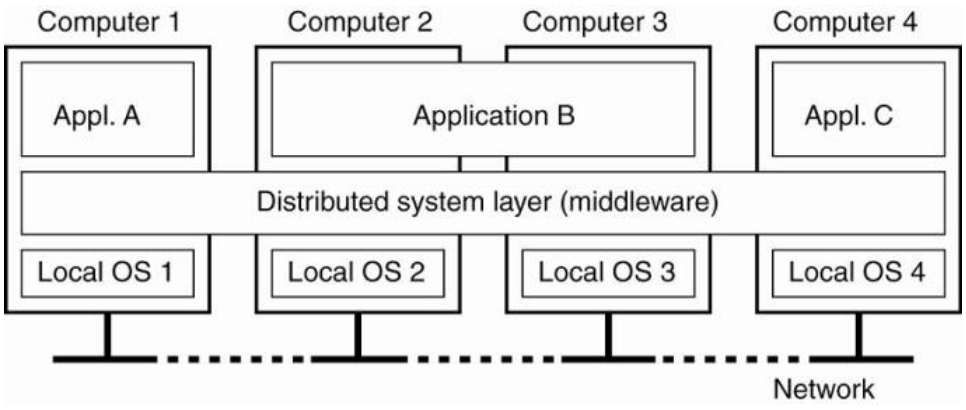

Figure 1.1: The classical architecture of a distributed system

- Synchronization
- Coordination and consistency
- Resource utilization
- Heterogeneity and interoperability
- Security and privacy

System management: System management is one of the critical issues of a distributed system. In a centralized system, a central node controls and manages the whole system, but no such central node exists in a truly distributed system. A distributed system is constructed by multiple independent nodes located at different places and connected through a network. A node may not have global knowledge of the system, making distributed system management more complex and challenging.

Synchronization: Clock and event synchronization is another important issue in a distributed system. It is often important to know when events occurred and in what order they occurred. Clock synchronization deals with understanding the ordering of events produced by concurrent processes. In a centralized system, time is unambiguous because the system has a single clock, and the events occurred following this clock. On the other hand, in a distributed system, the nodes are independent, and they have their own clocks. A clock always runs at a constant rate because its quartz crystal oscillates at a well-defined frequency. However, due to differences in the crystals, the rates at which two clocks run are normally different from each other. The difference in the oscillation period between two clocks might be tiny, but the difference accumulated over many oscillations leads to an observable difference in the times of the two clocks, no matter how accurately they were initialized to the same value. On the other hand, events can be created by the different nodes and the times of different clocks can be different. So the clock and event synchronization in distributed computing is also an vital issue. In centralized systems, critical section, mutual exclusion, and other synchronization problems are solved using semaphore. However, the concept of semaphore does not work in distributed systems because it implicitly relies on the shared memory concept.

Coordination and consistency: One of the main goals of the distributed computing is to complete a task in a faster manner by combining the power of multiple machines. Here, multiple independent nodes work together, so coordination and consistency maintenance is inevitable to complete a task successfully. In a distributed system, nodes are independent and connected through links that may be unreliable. The nodes and links may fail and recover independently, and there is no shared memory concept. So the nodes communicate and coordinate through messages passing, and the concept of semaphore does not work to maintain consistency. So coordination and consistency maintenance is not straightforward in a distributed system. On the other hand, unpredictable communication delay also makes coordination and consistency maintenance more challenging.

Resource utilization: Resource sharing is another important objective of a distributed system. Resource sharing means that several nodes in the system can access the existing resources of the system. The concept of resource sharing helps to build a cost-effective and fault-tolerant computing environment. Efficient resource utilization improves the performance of the system. A distributed system can have multiple resources situated at different places, and they can be heterogeneous. A node may not have global knowledge of the resources and their status. On the other hand, multiple nodes may try to access the same resource(s) simultaneously. So efficient resource management and utilization are difficult in a distributed system.

Heterogeneity and interoperability: It is a crucial design issue of a distributed system. The distributed system contains many different kinds of hardware and software working together cooperatively to solve problems. There may be many different representations of data and different instructions sets in the system. Different nodes may follow the different architecture and have different operating systems, programming languages, and communication media and protocols. Attempts to provide a universal canonical form of information are challenging. So interoperability among nodes is also difficult and challenging.

Security and Privacy: How to apply the security policies to the interdependent nodes is a great issue in distributed systems. Since distributed systems deal with sensitive data and information, the system must have robust security and privacy measurement. In this environment, users can access local and remote resources in order to run processes. Here nodes are connected through a network, so network security is also associated with a distributed system. Protection of distributed system assets, including base resources, storage, communications and user-interface, and higher-level composites of these resources, like processes, files, messages, and more complex objects, are important issues in a distributed system.

A coordinator or leader can solve several above explained issues such as system management, synchronization, coordination and consistency maintenance, and resource utilization of a distributed system. The leader election methods help to elect a leader or coordinator among the nodes of the system. In this thesis, we address several distributed systems' issues by proposing different leader election algorithms for the distributed systems.

OUTLINE: The rest of this chapter is organized as follows. Section 1.1 presents a brief background of distributed system and Section 1.2 defines the problem statement and the main research goals of this thesis. In Section 1.3, we present the motivation of the thesis. The significant contributions of this work are presented in Section 1.4. Finally, Section 1.5 finishes the chapter by detailing the structure of this thesis.

1.1 Background

Before presenting an overview of the development of solutions to the leader election problem, we want to present a general overview of the distributed systems.

The first distributed system named ARPANET [96], one of the predecessors of the internet was invented in 1967, and E-mail became the most successful distributed application of the ARPANET in the early 1970s. Since the invention of ARPANET, several distributed architectures have been evolved. Over time two distributed architectures became popular and have been started to use widely. They are -

- Client-server architecture
- Peer-to-Peer architecture

Client-server architecture: The Client-server architecture is a distributed computing architecture in which the server(s) hosts, delivers, and manages most of the resources and services to be consumed by the clients. In this architecture, client computers are connected to the server(s) over a network or internet connection. The clients' request for services and servers provide services regarding the clients' requests. Figure 1.2 depicts a client-server architecture of a distributed system.

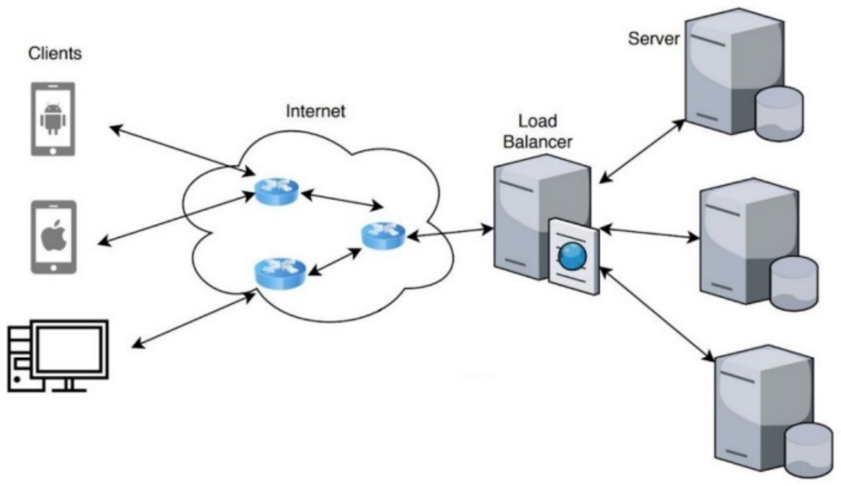

FIGURE 1.2: The client-server architecture of a distributed system

Peer-to-Peer architecture: Peer-to-peer architecture is a type of distributed architecture in which there is no division or distinction of abilities amidst the various nodes of a network. Every node has the same responsibilities and can perform the same set of actions. In this architecture, every node can act as both the server and the client as needed. All the decision-making and responsibilities are split up amongst the nodes involved. Figure 1.3 depicts a peer-to-peer architecture of a distributed system.

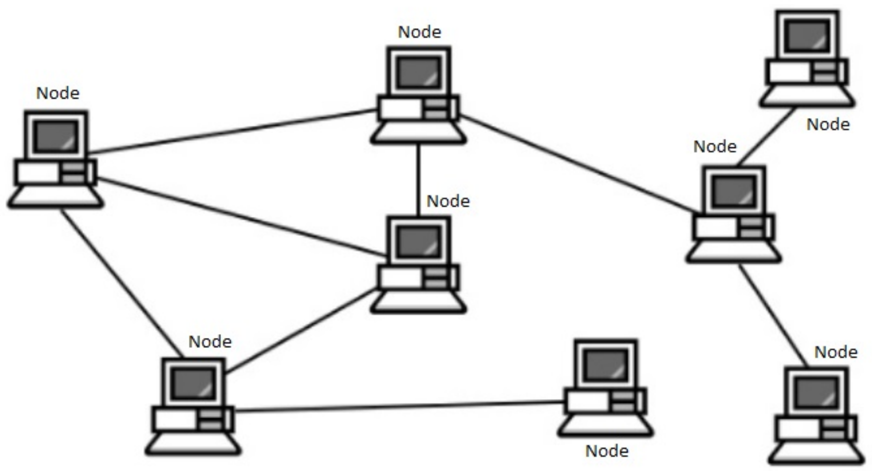

FIGURE 1.3: The peer-to-peer architecture of a distributed system

In both of these distributed computing architectures, the leader election concept helps reduce the complexity of a distributed system by effectuating control through a leader node elected by an appropriate leader election algorithm and improves overall system performance.

Different building blocks including node, communication link, protocol, middleware, and distributed algorithm are essential components to construct a distributed system. A brief description of some building blocks of a distributed system is as follows.

Node: A node can be defined as an abstract computational entity able to execute computations in a distributed system. Sometimes a process is also considered as a node. This thesis considers distributed systems composed of N nodes where N is a finite integer and $N > 1$. If a node behaves according to its specification, then it is called a correct node. On the other hand, if a node does not behave according to its specification or suffers a failure, it is called an incorrect node.

Communication link: Multiple nodes need to be connected through a network to build a distributed system. Communication links or channels represent a high-level abstraction of connections that connect the nodes in the construction of the distributed systems. The nodes of a system communicate through these links by the exchange of messages. The communication links can be two types i.e., unidirectional and bidirectional. In our works, we consider bidirectional communication links.

Protocol: A protocol is an established set of rules that govern the data transmitted, processing, and communication between different nodes in the network. Essentially, it allows connected nodes to communicate with each other, regardless of any differences in their internal processes, structure or design.

Middleware: Middleware is software that provides common services and capabilities to applications outside of what is offered by the operating system. Data management, application services, messaging, authentication, and API management are all commonly handled by middleware. It acts as the connective tissue between applications, data, and users and helps developers build applications more efficiently.

Clock: In a distributed system, each node has a logical clock that enables measuring time passage. However, it is not always possible that every clock remains synchronized with the rest of the clocks.

Distributed algorithm: A distributed algorithm can be defined as a collection of deterministic automata [91]. It is an algorithm designed to run on computer hardware constructed from interconnected loosely coupled nodes. The distributed algorithm is responsible for running different or same parts of the algorithm on different nodes simultaneously. Usually, two classes of properties are used to prove the correctness of distributed algorithms: *Safety* and *Liveness* [69] [63]. Both properties are often adopted in the design and specification of fault-tolerant distributed systems. Formally, safety and liveness guarantee the following properties:

Safety: This property states that some particular bad thing never happens. This property ensures that the algorithm should not do anything wrong.

Liveness: It states that some particularly good things will eventually happen, i.e., the algorithm will eventually produce the expected result.

1.2 Research Goals and Problem Statement

The leader election plays a vital role in addressing several issues of the distributed systems such as system management, synchronization, coordination and consistency, and resource utilization by electing a node among the nodes of the system as the system leader. Though many leader election methods have been introduced so far, there are still some research questions regarding the leader election problem.

RQ-1: Is it possible to design self-stabilizing leader election algorithms that can reduce the time and message overhead of the election process? If it is possible, then how?

RQ-2: Is it possible to design leader election algorithms that can tolerate more number of links and nodes failures than the algorithms known previously? If it is possible, then how?

RQ-3: Is it possible to design leader election algorithms that can elect a good quality leader according to the system requirement? If it is possible, then how?

In this thesis, we are interested in answering all these exciting research questions regarding the leader election problem in distributed systems by proposing some new leader election methods. The main objectives of this thesis are as follows.

- Study the several existing leader election methods and identify the various issues of these methods that need to be addressed to improve the overall system performance and management.

- Design some leader election methods to resolve the identified issues so that the system performance and management get improved.
- Analysis, simulation, and characterization of the newly designed election methods and comparison with the existing algorithms to show how this compares with the known representative algorithms.

The problem statement of this thesis can hence be defined as follows.

Design, analysis, and characterization of some new leader election methods for distributed systems to address various issues and challenges of existing leader election methods.

1.3 Motivations

With the ever-growing technological expansion of the world, distributed systems are becoming popular and widespread. Several significant issues associated with the distributed systems, such as system management, synchronization, coordination and consistency maintenance, resource utilization, and task allocation, are addressed by electing a node as the system leader. A central node controls and manages the whole system in a centralized system, but no such central node exists in a truly distributed system. A distributed system consists of multiple independent nodes. So the question is which node will take responsibility for managing the system, synchronizing the clocks and events, allocating the task, and maintaining the coordination among the nodes. Here concept of leader election comes into the scenario and answer all these questions. A node is elected among all the nodes in the system as the system leader that takes responsibility for managing the system, synchronizing the clocks and events, allocating the task, and maintaining the coordination among the nodes. Thus leader election plays a significant role in the distributed systems, and we are interested in working on this problem.

A good number of leader election algorithms have already been introduced by researchers in the past decades. However, there are still some issues and research gaps that are challenging and need to be addressed. We have performed a literature survey in the second chapter of this thesis. This literature survey helps us identify some issues and research gaps associated with the leader election in distributed systems. These issues and research gaps form the motivation of this thesis. The identified issues research gaps are as follows.

- Many leader election algorithms have been introduced based on the ring network topology. These algorithms are designed considering a system model where either no link and node failures occur during the election or the failures are permanent. However, in a real scenario, the link and node failures can happen during the election, and the failures can be transient, which means the failed node can recover during the election. So it is required to design algorithms considering the failure-recovery model for a ring network.

- The leader manages the system, coordinates the nodes, and utilizes the system resources. So the overall performance of the system depends a lot on the quality of the elected leader. Most existing algorithms elect a node with maximum or minimum node Id, which is not very practical for electing a good quality leader. They did not consider any quality attribute of the nodes. So the node with minimum or maximum Id may not be an excellent qualitative node. They only concentrated on reducing the time complexity and message complexity of the election process. So it is required to design algorithms that can elect a good quality leader for the system with minimum time and message overhead.

- Torus network topology offers many advantages such as higher speed, lower latency, better fairness, and lower energy consumption. For these kinds of benefits, nowadays, it is used to construct many parallel and distributed systems like IBM Blue Gene, IBM Sequoia, Mira, and Sugon TC8600. However, very few algorithms are designed for the torus network. These algorithms' message and time overhead are high, and they can tolerate very few links and nodes failures. The existing algorithms for the $2D$ torus network cannot even tolerate a node's two links failures. So it is required to design algorithms for the torus network that can have lower message and time overhead and tolerate more links and node failures than the existing algorithms.

- Many leader election algorithms have been designed for different types of distributed systems. However, best of our knowledge, no leader election algorithm is designed considering a distributed real-time system. Nodes in this kind of system cooperate to achieve a common goal within specified deadlines. The correctness of such (DRTS) system behavior depends not only on the logical results of the computations but also on the time when the results are produced. Missing the deadline may have disastrous consequences. So it is required to design algorithms that can instantly elect another node as a leader after crashes the leader.

1.4 Contributions

This thesis is committed to developing several leader election methods to resolve various research gaps of the existing leader election methods to address several issues in distributed systems. This section provides the thesis's important contributions, including the algorithm design, implementation, and comparative analysis of the proposed methods for addressing the leader election problem in distributed systems. The significant contributions are:

- In the past four decades, many algorithms have been proposed to solve the leader election problem. In this thesis, first, we study the related works regarding the leader election problem and find some major issues, research gaps, and challenges of the existing leader election methods.
- We propose a self-stabilizing leader election algorithm named "FRLLE: A Failure Rate and Load-based Leader Election Algorithm" for a failure-recovery

bidirectional ring network. We prove that the FRLLE algorithm is self-stabilized, and it elects a reliable and low-loaded node as the system leader. Besides, we calculate the time complexity and message complexity of the FRLLE algorithm, simulate the FRLLE algorithm, compare the simulated results with the relevant existing algorithms and show that the message and time overhead of the FRLLE algorithm is lesser than the existing algorithms.

- We design a self-stabilizing leader election algorithm entitled "Lea-TN: A leader election algorithm considering node and link failures in a torus network" for a failure-recovery $2D$ torus network. We propose 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). We introduce new patterns for sending messages that help reduce the number of exchanged messages and the execution time of the election process. The proposed algorithm (Lea-TN) can tolerate more links and nodes failures than the existing algorithms designed for $2D$ torus network and it enables a node to identify its link failures during the election also. Further, we simulate the Lea-TN algorithm and compare its performance with that of the well-known existing algorithms that corroborate that the message and time overheads of the Lea-TN algorithm are lesser than the relevant existing algorithms.

- We propose a self-stabilizing leader election algorithm for the distributed real-time systems. Here, we introduce the concept of the primary and provisional leaders, which helps reduce system performance degradation during the election. Our algorithm not only elects a leader but also makes a list of r best leader capable nodes. Whenever a leader fails, instantly, the system can choose a provisional leader from the list so that due to the lack of coordination, the system's work does not get halted for a long time. We reduce the message and time complexity of the election process by dividing a distributed system into two layers (i.e., inner-layer and outer-layer) based on the eccentricity of the nodes. Only the inner-layer nodes identify the list of potential nodes and elect the new system leader. This work also introduces the leader factor concept to elect a good quality leader for the system.

- We proposed a leader election algorithm that introduce the concept of quality factor to elect a suitable leader according to the system requirements in a dynamic distributed environment. This algorithm is designed for an arbitrary network topology so that it can also work on a regular topology also. We involve multiple experts to identify the appropriate attributes of a good quality leader (according to the system requirements) and assign weight to identified attributes according to their importance to elect the suitable leader. To calculate the quality factor, we modify the concept of the TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution) multi-criteria decision-making method so that it can be be implemented in a distributed environment. Besides electing a good quality leader, the proposed algorithm is message-effective and time-effective also.

1.5 Thesis Outline

Throughout this research work, various research issues regarding leader election in distributed systems are identified. To address the identified issues, several leader election methods are proposed, and research papers are written. This thesis contains seven chapters. Chapter 1 is an introductory chapter. Chapter 2 presents the literature survey. For better understanding and clear view, the core work of this thesis (derived from our own research papers) is presented in Chapter 3, Chapter 4, Chapter 5, and Chapter 6. Finally Chapter 7 summarizes the thesis. Figure 1.4 depicts the organization of this thesis. A brief description of every chapter in this thesis is as follows.

Chapter 1: This chapter briefly describes the distributed system and its importance. It also explains the role of leader election in the distributed systems. This chapter provides the motivations behind the thesis by explaining some research issues

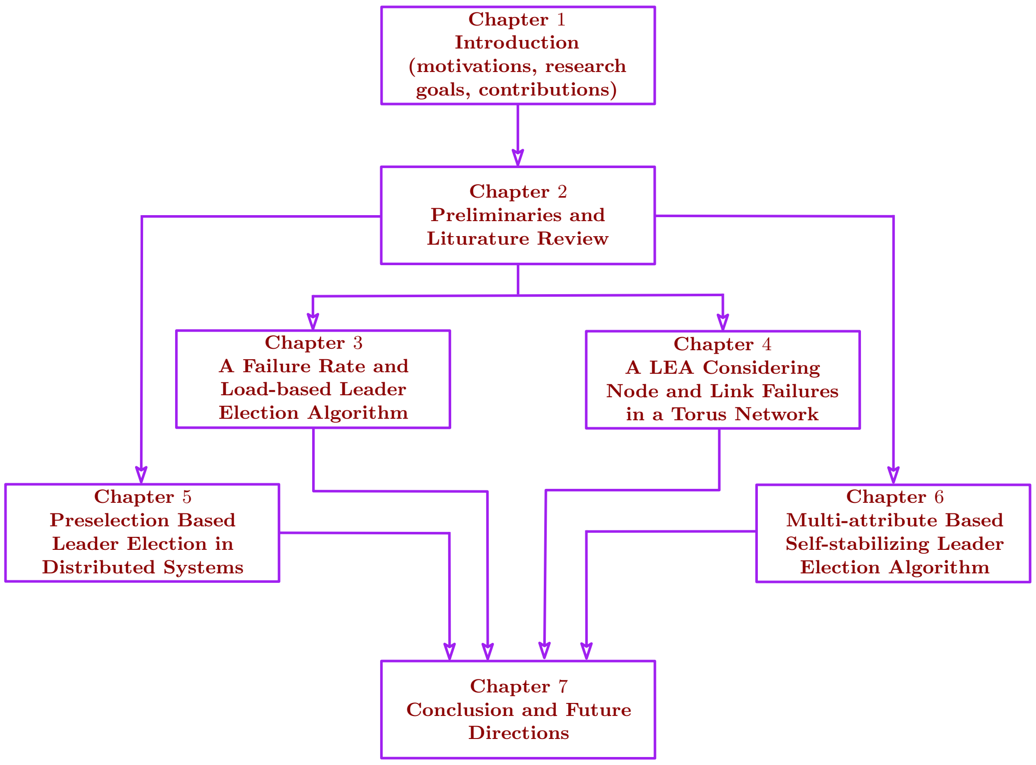
Chapter 1
Introduction
(motivations, research goals, contributions)
Chapter 2
Preliminaries and Liturature Review
Chapter 3
A Failure Rate and Load-based Leader Election Algorithm
Chapter 4
A LEA Considering Node and Link Failures in a Torus Network
Chapter 5
Preselection Based Leader Election in Distributed Systems
Chapter 6
Multi-attribute Based Self-stabilizing Leader Election Algorithm
Chapter 7
Conclusion and Future Directions

FIGURE 1.4: Thesis Structure.

in the existing leader election methods. Further, it provides our research objectives and contributions. At the end of this chapter, we describe the thesis organization.

Chapter 2: This chapter first presents the preliminaries regarding leader election algorithms. Then it performs a literature survey related to the leader election problem and identifies the issues and research gaps in the existing leader election methods.

Chapter 3: In this chapter, we propose a self-stabilizing leader election algorithm entitled “FRLLE: A Failure Rate and Load-based Leader Election Algorithm” for a failure-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 who can comfortably concentrate on leadership roles and activities. This chapter also includes a complexity analysis, simulation of the proposed algorithm, and comparison of the FRLLE algorithm with the existing algorithms.

Chapter 4: In this chapter, first, we propose a lower bound message complexity on a comparison-based leader election for a $2D$ torus network. Next, we sketch a new leader election algorithm (Lea-TN) considering both the node and link failures for a $2D$ torus network. Then we show that the 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 consider the number of non-faulty links and the subsisting nodes' failure rate to elect a reliable leader. We introduce new patterns for sending messages that help reduce the number of exchanged messages and the execution time of the election process. The proposed algorithm (Lea-TN) enables a node to identify its link failures during the election also. Finally, we simulate the Lea-TN algorithm and compare its performance with that of the well-known existing algorithms.

Chapter 5: In this chapter, 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. Our algorithm always tries to elect a node with the highest quality-coefficient as the leader so that the system gets a high quality leader. We show that 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.

Chapter 6: This chapter proposes a novel multi-attribute based self-stabilizing leader election method for a dynamic distributed system. 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, a modified TOPSIS MCDM method is used 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 prove 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.

Chapter 7: This chapter concludes the thesis by summarizing the main findings of the works done herein with some possible future research directions.