Preface

With the ever-growing technological expansion in the world, distributed computing is becoming popular and widely used in several fields. From scientific research to everyday life applications, everywhere we are using distributed computing and enjoying the benefits of this computing paradigm. Distributed computing helps improve the performance of large-scale projects by combining the power of multiple computing devices. Distributed systems are the backbone of distributed computing. A distributed system is a collection of multiple independent computing components or nodes that work together to attain a common goal and appear as a single coherent system to the user. These independent nodes are connected through a network, and they perform the tasks of the system with collaboration. In a distributed system, to complete an enormous task fast, the task is partitioned into a set of possible sub-tasks and allocated to multiple nodes. Then the nodes execute their assigned sub-tasks in a concurrent and consistent manner to complete the task in a faster way. The banking system, manufacturing industry, artificial intelligence, e-commerce, transport, health care, agriculture, defense systems have been using distributed computing for a long time. Various modern technologies like smart city, blockchain, cloud computing, edge computing, and so on are constructed based on the concept of decentralized systems. Recently, distributed machine learning and deep learning technologies are inclined to adopt the concept of distributed systems to fulfill their need for a reliable, high-performance, and large-scale computational platform. Though a distributed system is supposed to be scalable, cost-effective, fault-tolerant, highly available and reliable, and provides a high-performance computing environment, several issues and challenges are also associated with such a system. System management, synchronization, consistency maintenance, and efficient resource utilization are some significant issues. In a centralized system, a central node controls and manages the whole system, synchronizes the events of

the system, utilizes the resources efficiently, and maintains consistency among the several entities of the system. However, no such central node exists in a truly distributed system. Hence, a node needs to be elected as the system leader to manage, coordinate, synchronize, and efficiently utilize a distributed system. The leader helps distribute the computing load, aggregate the computing results, and present a single system image for processes and users. It also simplifies the system management, coordination, and operational complexity and reduces the message and time overhead of the system. Thus, the leader election concept helps reduce the complexity of a decentralized system by effectuating control through a leader node elected by an appropriate leader election algorithm and improves overall system performance. A suitable system leader can help to improve a distributed system's resource utility, reliability, and fault tolerability that directly improve the performance of the applications of the various fields that adopt this system. So, the leader election plays an important role in the distributed systems.

In this thesis, through a detailed literature survey on existing Leader Election Algorithms (LEAs), we first find out some major issues and challenges with them. Then we propose possible solutions to those issues. While doing that, to ease out the survey process, we first divide the existing LEAs into two categories based on network topology. One, algorithms for regular network topology, and two, algorithms for arbitrary network topology. Then we study the LEAs designed considering various distributed system models for these two categories and find the research gaps. Finally, we propose, analyze and characterize four different self-stabilizing leader election methods to overcome those research gaps. The first two algorithms are designed for two regular network topologies (ring and 2D torus), and the subsequent two algorithms are designed for the arbitrary network topology.

We design our first algorithm for a ring network topology considering a failure-recovery and partially synchronous distributed system model. The main aim of designing this algorithm is to reduce the message complexity and time complexity of the election process and elect a low-loaded and higher reliable node as the system leader. The second algorithm is designed for a 2D torus network. Here, we propose a lower bound message complexity of a comparison-based leader election algorithm in a 2D torus network. The objectives of this work are to design a more fault-tolerant leader election algorithm and reduce the message and time overhead of the election process. We introduce several message-sending patterns that make the algorithm

more link and node failures tolerant, reduce the number of exchanged messages and time steps of the election process, and elect a good quality leader for the system.

Our last two algorithms are designed for arbitrary network topologies. The first one of these two algorithms is designed considering a distributed real-time system. Here, considering the characteristics of a distributed real-time system, the proposed algorithm not only elects a leader but also identifies some higher potential nodes for leadership such that if a leader crashes, the system can select another potential node as the system leader. Using the eccentricity of the nodes, we divide the network into two layers, i.e., the inner layer and the outer layer, and only the inner layer nodes take part in the election directly, which helps reduce the message and time overhead of the election process.

A good quality leader can manage a distributed system in a better way and utilize the resources efficiently that improve the overall system performance. 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 and that is why the fourth algorithm is designed to elect a good quality leader according to the system requirements by introducing the concept of the quality factor of the nodes. We modify the concept of the TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution) MCDM method to calculate the quality factor of the nodes. Here, we involve 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 for the system. This algorithm can tolerate multiple links and node failures during the election and reduce the time and message overhead of the election process.