

Research Survey: Leader Election Algorithms Designed For Crash-recovery Fault Models

1. Introduction

Leader election is fundamental to distributed systems, ensuring coordination, synchronization, and fault tolerance. However, in **crash-recovery environments**, where nodes can fail and later rejoin, maintaining a consistent and responsive leader remains a key challenge. This survey focuses specifically on **leader election algorithms designed for crash-recovery fault models**, tracing their evolution from classical to modern approaches and outlining directions for future research.

2. Evolution of Leader Election Algorithms

Past Approaches:

Classical algorithms such as **Bully** and **Ring** pioneered leader election, emphasizing deterministic selection but assuming stable networks. Later, **failure detector-based** methods introduced partial fault awareness, improving reliability but increasing message overhead.

Present Developments:

Modern algorithms like **Raft**, **Zab**, and **Multi-attribute Self-Stabilizing Leader Election (2025)**—incorporate self-stabilization, adaptivity, and partial asynchrony handling. Techniques such as **hierarchical re-election** and **multi-attribute scoring** optimize recovery and minimize message complexity during transient failures. Research also explores **near-linear time algorithms** for large multiagent systems and **knowledge-free** designs to operate without global state assumptions.

Future Directions:

Emerging trends point toward **self-adaptive, context-aware election** that integrates machine learning for predictive fault handling, **secure leader verification** to prevent malicious takeovers, and **hybrid consensus-election protocols** to unify fault tolerance with performance guarantees.

3. Evaluation Framework

This study compares algorithms across:

- **Stabilization time** after leader failure
- **Message complexity** during re-election
- **Resilience** under crash-recovery and partition scenarios
- **Recovery efficiency** during node reintegration

Controlled simulations will assess performance under varying fault frequencies and network latencies.

4. Expected Outcomes

We expect to:

- Provide a **comparative summary** of state-of-the-art algorithms under crash-recovery conditions.
- Identify **trade-offs** among message efficiency, re-election speed, and fault tolerance.
- Highlight **research gaps** in adaptivity, asynchrony handling, and robustness to network churn.
- Recommend strategies for **self-stabilizing and lightweight leader re-election** in dynamic distributed environments.

5. Team Roles

Ishank Sharma: Implements the main leader election algorithm and collaborates on system integration. Collects and analyzes stabilization time, message complexity, and recovery efficiency. Helps summarize and visualize experimental results for the survey.

Prerana: Co-develops fault injection and recovery logic, integrating it with the main algorithm. Leads the literature review and synthesizes insights for the report. Participates in coding, debugging, and documenting the system and findings.

Srilalitha: Sets up the evaluation framework and codes monitoring/analysis tools. Collaborates on implementation, testing, and running experiments. Assists in comparative analysis and presentation of results.

Key References

- Multi-attribute Self-Stabilizing Leader Election, **J Supercomputing (2025)**
- Hierarchical Adaptive Leader Election for Crash-Recovery Model, **ACM (2024)**
- Enhancing Election Algorithms for Distributed Systems, **IJSCE (2025)**
- Near-Optimal Knowledge-Free Resilient Leader Election, **Automatica (2022)**