Group No: 5, Team Status Report

Project: Leader Election Simulator under Crash-Recovery Fault Models

Team: Ishank Sharma, Srilalitha Subbaswamy and Prerana Puttaswamy

GitHub: github.com/ishank-dev/leader-election-fault-tolerance

Scope. This work has two deliverables: a *survey* and a *benchmark*. The *survey* synthesizes leader-election algorithms for crash–recovery systems—classic *Bully* [5], *Ring* [6], and *Raft*'s election mechanism [7]—and distills design assumptions (synchrony, failure detectors), safety/liveness guarantees, and time/message complexity, guided by recent advances in self-stabilizing and crash-recovery election [1–4]. The *benchmark* implements these algorithms in a single-process, event-driven simulator (10 nodes) and evaluates them under a standardized fault profile (50 ms links; one crash at t=2 s; optional restart at t=3 s). We report stabilization time, re-election latency, message counts, and success rate, with planned variants adding latency jitter and probabilistic loss.

Progress and Completion

- **Overall completion: 90% of proposed work achieved.**
- Core algorithms (Bully, Ring, Raft election) and the event-driven simulator implemented successfully.
- Fault injection and recovery integrated; metrics computed for election time, re-election latency, and total messages.
- Comparative analysis completed across algorithms, aligned with proposed evaluation framework.
- Remaining: variable latency, failure detectors, and self-stabilization support [8–10].

Current Results

Setup: 10 nodes, 50 ms latency, crash@2.0s, restart@3.0s.

Algorithm	Election (s)	Re-election (s)	Messages	Success
Bully	0.050	1.060	1532	✓
Ring	0.900	1.060	31	✓
Raft	0.100	1.110	54	✓

Analysis: Bully achieved the fastest initial election (0.050 s) but required more messages ($O(n^2)$) [5]. Ring showed minimal message overhead (31 msgs) and stable re-election (1.060 s) [6]. Raft balanced speed and efficiency (54 msgs) [7]. All three algorithms achieved 100% success in leader recovery after crash events.

Team Contributions

- **Ishank Sharma:** Designed simulator core, implemented Raft election, and metric computation. Analyzed stabilization time, message complexity, and recovery efficiency—matching proposal's algorithm and analysis goals.
- **Srilalitha:** Developed evaluation and monitoring framework, automated timing/message tracking, and comparative testing—fulfilling her proposed role in system evaluation.
- **Prerana:** Implemented crash-recovery logic, integrated restart features, and authored documentation/literature synthesis connecting theory to results—aligned with proposal deliverables.

Difficulties and Next Steps

- **Timing precision:** Early leader detection lag fixed using real-time event listeners.
- **Re-election trigger:** Needed explicit invalidation of `leader_id` to ensure consistent recovery.
- **Simplified model:** Fixed latency and deterministic ticks limit realism.
- **Next:** Add latency jitter, probabilistic message loss, and per-phase metrics; integrate heartbeat-based failure detection [8,9] (including Bully/Ring adaptations); run multi-seed trials for $p50/p95$ election stability; explore self-stabilizing variants [1, 4, 10].

Note: The simulator implements core algorithms and the evaluation framework from the proposal. Phase 2 will complete adaptive timing and fault-tolerant extensions, with results synthesized against prior art [1–4].

References

[1] *Multi-attribute Self-Stabilizing Leader Election*, Journal of Supercomputing, 2025.

[2] *Hierarchical Adaptive Leader Election for Crash-Recovery Model*, ACM, 2024.

[3] *Enhancing Election Algorithms for Distributed Systems*, IJSCE, 2025.

[4] *Near-Optimal Knowledge-Free Resilient Leader Election*, Automatica, 2022.

[5] H. Garcia-Molina, "Elections in a Distributed Computing System," *IEEE Transactions on Computers*, 1982.

[6] E. Chang and R. Roberts, "An Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processes," *Communications of the ACM*, 1979.

[7] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm (Raft)," *USENIX ATC*, 2014.

[8] T. D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *Journal of the ACM*, 1996.

[9] N. Hayashibara, X. Défago, R. Yared, and T. Katayama, "The φ Accrual Failure Detector," *IEEE SRDS*, 2004.

[10] E. W. Dijkstra, "Self-stabilizing Systems in Spite of Distributed Control," *Communications of the ACM*, 1974.