Increasing Stability of Geo-replicated Raft with Timeout Free Elections

Nick Riedman
Stanford University
nriedman@stanford.edu

David Smith
Stanford University
davidzws@stanford.edu

Abstract

In a globally replicated (or geo-replicated) state machine, long-tail network latencies and short-term disruptions may cause packet round trip times (RTT) to vary widely. This makes selecting a predetermined, static election timeout that both prevents spurious elections and detects leader failures quickly infeasible. To maximize system availability through a leader failure in a geo-replicated state machine, we implement a modified Raft that uses a timeout-free election scheme (phi accrual fault detection) that adjusts to changing network conditions on-line. We evaluate the effect of timeout-free elections on system availability by deploying globally distributed Raft clusters on Google Cloud Virtual Machines. System availability is measured by counting spurious elections and tracking leader crash detection and replacement speed. Comparing against a traditional predetermined timeout range, we examine how timeout-free election schemes can reduce unavailability of geo-replicated Raft.

1. Introduction

The Raft consensus algorithm is a widely used method for building fault tolerant distributed systems. Each server in a cluster maintains an identical log which, when applied in order to a state machine, brings the state machine to a consistent state that all nodes agree on. Raft achieves consensus by electing a node as a leader, which then processes client requests, instructs the rest of the nodes when to apply the entry to their logs, and sends regular heartbeats to its followers [2]. When a leader fails, the remaining nodes decide to elect a new leader after a random amount of time from a pre-defined range (e.g. 150-300ms) has passed without hearing from the leader. During an election, the system is unable to process requests, and is therefore considered unavailable. To ensure a leader failure is detected quickly while preventing spurious elections (minimizing unavailability), the election timeout range should be chosen based on how long it takes a leader to broadcast heartbeats.

2. Related Work

Ongaro and Ousterhout [2] introduced Raft, a consensus algorithm for managing a replicated log across a distributed cluster while tolerating crash failures. It was designed as an alternative to Paxos with an emphasis on understandability and practical implementation, while maintaining comparable performance and safety guarantees. In the replicated state machine model, each server maintains a log of commands that are applied in the same order across replicas, ensuring that all nodes reach the same state as long as a majority of servers remain operational.

Raft decomposes the consensus problem into three components: leader election, log replication, and safety. At any time, one node serves as the leader and coordinates client requests by appending commands to its log and replicating them to followers. If followers stop receiving heartbeats from the leader within an election timeout, they initiate a randomized leader election. Once a majority of servers acknowledge a log entry, it is considered committed and can be applied to the state machine. While Raft's leader-based structure simplifies log management, it relies on carefully tuned election timeouts to detect leader failures efficiently, which can be challenging in environments with highly variable network latency.

Hayashibara et al. [1] introduced the concept of the φ accrual failure detector to address the inherent trade-offs between detection speed and accuracy in dynamic network environments. Unlike traditional failure detectors that operate on a binary interaction model (using a fixed timeout to identify failures), the accrual model outputs a suspicion level on a continuous scale. The primary architectural advantage of this approach is the decoupling of network monitoring from application-specific interpretation. By providing a continuous value rather than a boolean status, the accrual detector allows distributed protocols to trigger graduated responses—such as throttling requests or initiating precautionary leadership transitions—based on varying levels of confidence in a leader's failure.

Xiong and Défago [3] expanded on the work of Hayashibara et al. [1] by proposing the Exponential Distribution Failure Detector (ED FD). While the original φ

failure detector implementation by Hayashibara et al. [1] relied on a Gaussian distribution to estimate heartbeat inter-arrival times, Xiong and Défago [3] identified that a Gaussian model often fails to accurately represent network jitter in wide-area networks (WANs) or unstable environments. The Exponential Distribution Failure Detector (ED FD) uses an exponential distribution to more effectively capture the "long-tail" latencies and message losses characteristic of intercontinental links. Their experimental results demonstrated that the exponential model significantly improves quality of service (QoS) by providing shorter detection times and a lower mistake rate compared to the original φ implementation.

3. System Design and Implementation

To evaluate the impact of adaptive failure detection on geo-replicated Raft clusters, we extended a standard Raft implementation to support two leader failure detection mechanisms: the traditional randomized election timeout and an accrual failure detector based on the Exponential Distribution Failure Detector (ED FD). This design enables direct comparison between fixed-timeout and statistical leader failure detection schemes under identical network conditions and workloads.

Our system preserves the original Raft protocol semantics while modifying the mechanism used by followers to determine when the leader has likely failed. In both configurations, leaders periodically send heartbeat messages to followers. Followers track the arrival times of these heartbeats and use them to determine whether to initiate a new election. The primary difference between the two modes lies in how followers interpret delays between heartbeat arrivals.

The implementation is structured around a modular failure detection component that operates independently of the Raft consensus logic. Each follower maintains telemetry describing the timing of recent heartbeats and periodically evaluates whether the leader should be suspected. This design allows the failure detection strategy to be swapped without modifying the rest of the Raft state machine.

3.1. Raft Consensus Implementation

Our prototype builds on a standard Raft replicated state machine implementation that maintains the three canonical roles defined by the protocol: leader, follower, and candidate. As in the original algorithm, the leader processes client requests, appends entries to its log, replicates them to followers, and commits entries once a majority of replicas acknowledge them. Followers remain passive and respond to `AppendEntries` and `RequestVote` RPCs, while candidates initiate elections when a leader failure is suspected.

Leaders send periodic heartbeat messages to followers at a fixed interval (500 ms in our implementation). These heartbeats serve two purposes: they confirm that the leader remains active and they maintain log synchronization across the cluster. Followers record the arrival time of each heartbeat and use this information as input to the failure detector.

Aside from the failure detection mechanism described below, the Raft protocol implementation remains unchanged. Election rules, log replication, and majority-based commitment operate identically in both detection modes.

3.2. Fault Detection

Leader failure detection is the primary mechanism that determines when a new election should be initiated. Our implementation supports two alternative strategies: (1) randomized election timeouts, as defined in the original Raft algorithm, and (2) accrual failure detection based on a statistical model of heartbeat arrival times. Both methods operate within the same event-driven node loop. Each node periodically evaluates whether the leader should be suspected and triggers an election if the detection mechanism indicates a likely failure.

3.2.1 Randomized Election Timeouts

The baseline implementation follows the standard Raft approach. Each follower starts a timer after receiving a heartbeat from the leader. If no additional heartbeat arrives before the timer expires, the follower transitions to the candidate state and begins a new leader election.

The timeout duration is chosen randomly from a predefined interval (e.g., 150–300 ms). Randomization helps avoid split votes by reducing the probability that multiple followers initiate elections simultaneously. While this method works well in clusters with stable latency, selecting appropriate timeout bounds becomes challenging in geo-replicated environments where round-trip times vary significantly.

In wide-area deployments, a timeout that is too short can trigger frequent spurious elections due to temporary network delays, while a timeout that is too long increases the time required to detect an actual leader failure. These competing constraints motivate the use of adaptive failure detection.

3.2.2 Analyzing Heartbeat Intervals

To enable adaptive detection, each follower maintains a sliding window of recent heartbeat intervals. Whenever a heartbeat is received, the follower records the elapsed time since the previous heartbeat and stores the interval in a circular buffer. The buffer maintains the most recent N intervals, where N is configurable.

From this window of observations, the system estimates the statistical properties of heartbeat arrival times. In particular, the follower computes the mean inter-arrival time:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} I_i$$

where I_i represents the recorded heartbeat intervals.

The sliding window allows the system to adapt to changing network conditions while limiting the influence of outdated measurements. When the window has not yet accumulated enough samples (for example during startup), the detector temporarily falls back to the randomized timeout mechanism to ensure stable behavior.

3.2.3 Exponential Accrual Failure Detection (ED FD)

Once a sufficient number of heartbeat samples have been collected, the follower switches to an accrual failure detector based on the Exponential Distribution Failure Detector (ED FD). In this model, heartbeat inter-arrival times are assumed to follow an exponential distribution. This assumption reflects the long-tailed latency distributions commonly observed in wide-area networks.

Let t denote the elapsed time since the most recent heartbeat and μ denote the mean heartbeat interval estimated from the sliding window. The exponential distribution parameter is given by

$$\lambda = \frac{1}{\mu}$$

The probability that the next heartbeat arrives later than time t is

$$P(T \geq t) = e^{-\lambda t}$$

Following the accrual failure detection model, we compute the suspicion level φ as

$$\varphi = -\log_{10}(P(T \geq t))$$

Substituting the exponential survival function yields

$$\varphi = (\lambda t) \log_{10}(e)$$

This value represents the statistical confidence that the leader has failed given the observed delay since the last heartbeat. Larger values of φ indicate stronger suspicion.

Each follower compares the computed φ value against a configurable threshold Φ. If

$$\varphi > \Phi$$

the leader is considered failed and the follower initiates a new election.

The advantage of this approach is that the detection threshold corresponds to a probability-based confidence level rather than a fixed timeout. For example, increasing Φ makes the detector more conservative, reducing false positives at the cost of slower failure detection.

3.2.4 Integration with Raft

Both detection strategies are integrated into the Raft event loop through a unified timer mechanism. In timeout mode, the timer duration is chosen randomly from the configured timeout range. In accrual mode, the timer fires at a short fixed interval, allowing the follower to periodically recompute the φ value based on the current time and the most recent heartbeat arrival.

Heartbeat arrivals update the sliding window statistics and reset the suspicion level. Election events and heartbeat observations are recorded in telemetry logs for offline analysis.

Each log entry contains:

- timestamp,
- node identifier,
- event type (heartbeat, election start, election win),
- Raft term number,
- current leader identifier, and
- computed φ value (in accrual mode).

These logs enable detailed analysis of leader detection latency, false election rates, and overall system availability.

4. Evaluation

To evaluate the impact that timeout-free fault detection has on the system, we performed two sets of experiments:

1. A long-running trace to characterize steady-state performance.
2. A series of forced leader knockouts to exercise leader fault discovery and recovery.

In the first experiment, the system is kept in steady state by an extremely high election timeout. By preventing timeouts during the 8 hour run, we record a steady and uninterrupted stream of heartbeats from the leader. The leader sends a heartbeat every 500ms, and each follower logs the interval between heartbeat arrivals. Using these traces, we can perform offline analysis of different election schemes to see if any spurious elections would have been set off if that policy were in place.

In the second set of experiments, the systems election recovery mechanisms are exercised by repeatedly forcing the leader to crash. The system starts normally, and a test harness logs how many heartbeats the leader has sent. When the system reaches a firm steady state (50-100 heartbeats sent), the leader aborts. A new leader is elected, and the cycle repeats. As discussed in Section 3.2.4, each node logs timestamps for relevant events, including heartbeat receipt

and election events. We use this internal telemetry to measure the time it takes the failure detection policy to detect and replace the missing leader.

Finally, to accurately measure the heterogenous latency of geo-replicated systems, we deployed our Raft implementation on 5 GCP instances across globally distributed regions (asia-east, us-west, europe-north, australia-southeast, asia-south). An additional, a centrally located GCP instance served as a client in both experiments. To evaluate client-side availability, the client sends a steady stream of non-overlapping requests to the cluster and records the outcome.

5. Results

5.1. Heartbeat Interval Characterization

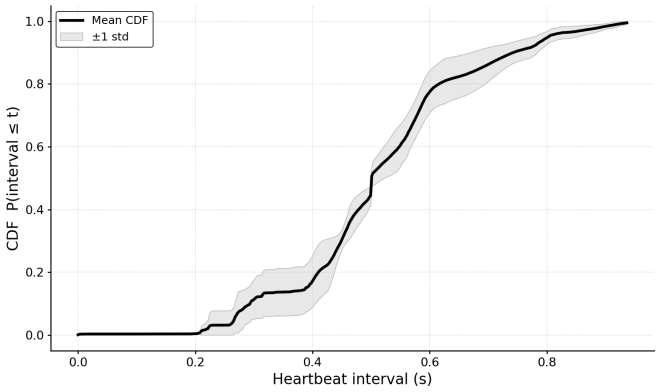


Figure 1. CDF of steady-state heartbeat intervals across 5 nodes in a globally distributed cluster ($N = 35,000$ per node). The leader sends heartbeats every 500ms. While WAN conditions cause some variability between nodes and occasional long-tailed jitter, heartbeat intervals are weighted towards the mean.

First, we used the steady-state trace to evaluate the distribution of heartbeat intervals from the perspective of a follower. As summarized in Figure 1, heartbeat intervals are stable and consistent across nodes in steady state, with a long tail due to occasional network delay. The vast majority of heartbeats fall within 300ms of the actual heartbeat interval, with 99th percentile intervals ranging from 0.803 seconds to 0.935 seconds across nodes.

5.2. Crash Detection & Leader Replacement Time

Figure 2 summarizes the results of the leader crash trials. Overall, it is clear that the ϕ accrual detector took significantly longer to detect and replace the leader in the common case. However, it is also interesting to note that the randomized timeout is much longer tailed. In the worst case, randomized election timeouts took much longer than the worst case accrual runs. This indicates that the accrual detector is sacrificing fast detection for a conservative estimate of when the leader has crashed. Given the frequency

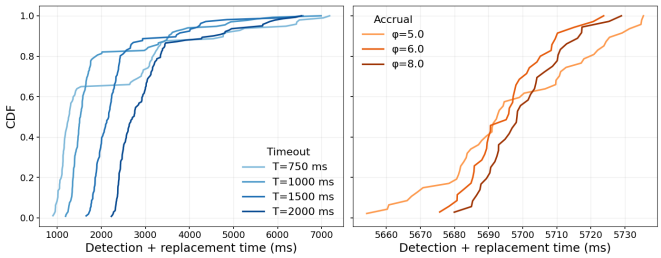


Figure 2. Time to detect and replace crashed leader (N¿=35 for each curve) using ϕ accrual and timeout.

of network disruptions in globally distributed systems that may cause a more aggressive system to flail, the conservative approach may be beneficial.

5.3. Election Success Rate

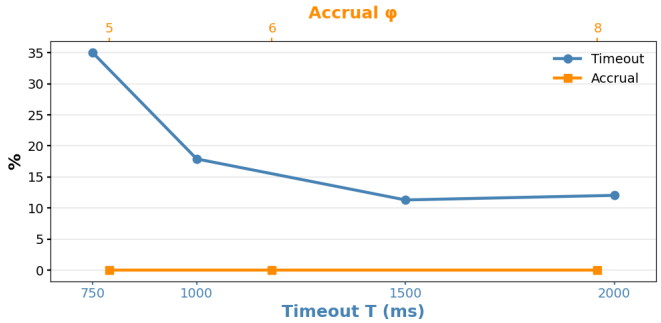


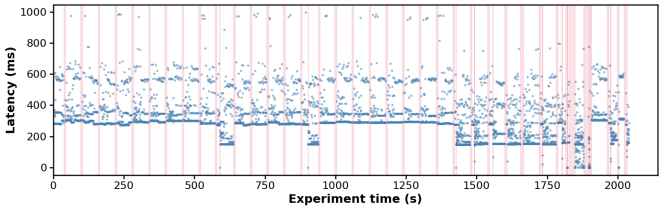
Figure 3. Percentage of leader crashes whose recovery involved at least one failed recover, by fault detector scheme.

One possible reason for the long-tailed crash recovery for timeout-based Raft is election stalling. When no node is able to reach a majority in a given election, the election fails, and a new election must begin. This can happen when multiple . The original Raft authors addressed this issue by randomizing the timeout from a large range [2]. We applied the same technique, and nonetheless observed up to 35% of crash-recoveries involving more than one election cycle for time-out based elections. By contrast, as seen in Figure 3, no ϕ accrual detector crash recovery required more than one election cycles.

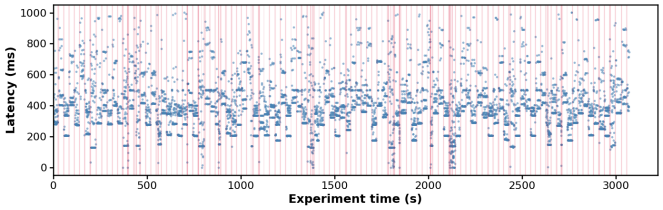
5.4. Client Availability

So far, all evaluation of the system has been based on internal telemetry collected at the node. However, the ultimate purpose of designing a fast and reliable system is to provide maximal availability to the client.

Figure 4 includes a trace of the client's logs during a leader crash experiment, both for accrual- and timeout-based policies. Aligning with the results of Section 5.3, we



(a) A trace from a crash experiment with accrual, $\varphi = 6.0$. Unavailable for 268.561s over 2039.6s runtime (86.83% available).



(b) Random timeout between 1.5-3s. 347.191s unavailable over 3069.8s runtime (88.69% available).

Figure 4. Client-side availability during leader crashes. Red bars indicate unavailable windows, and blue dots indicate successful request acknowledgment. Latency is the time between when the messages was sent and the ack was received.

can see that the ϕ accrual cluster spent more time unavailable than the timeout-based cluster.

6. Conclusion

Geo-replicated distributed systems experience highly variable network latency due to long-distance communication and transient network disruptions. In such environments, selecting a static election timeout that simultaneously minimizes spurious elections and detects leader failures quickly is inherently difficult. This work explored whether an adaptive failure detection mechanism could improve the availability of Raft clusters operating under these conditions.

We implemented a modified Raft system capable of operating with either traditional randomized election timeouts or a timeout-free leader detection scheme based on an accrual failure detector. Our implementation uses the Exponential Distribution Failure Detector (ED FD), which models heartbeat inter-arrival times using an exponential distribution and computes a continuously increasing suspicion value that reflects the likelihood of leader failure. This approach allows the system to dynamically adjust its failure detection behavior according to observed network conditions.

Through experiments on a globally distributed cluster of Google Cloud virtual machines, we evaluated both detection strategies using steady-state traces and repeated forced leader crashes. Our results show that while the accrual detector reduces election instability and eliminates multi-round election failures, it tends to detect leader failures more conservatively than randomized timeouts, resulting in longer recovery times in the common case. As a result, the accrual-based approach did not consistently improve client-visible availability compared to a carefully tuned timeout scheme.

These findings highlight an important trade-off in leader failure detection for geo-replicated consensus systems. Adaptive detectors can improve election stability and reduce pathological election behaviors, but their conservative detection characteristics may increase failover latency. Future work could explore hybrid approaches that combine statistical detection with bounded timeouts, or dynamically adjust suspicion thresholds based on recent network conditions. Such techniques may provide a more balanced approach to failure detection in highly variable wide-area environments.

References

[1] Naohiro Hayashibara, Xavier Défago, Rami Yared, and Takuya Katayama. The ϕ accrual failure detector. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)*, pages 66–78, Fortaleza, Brazil, 2004. IEEE. 1, 2

[2] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA, 2014. USENIX Association. 1, 4

[3] Naixue Xiong and Xavier Défago. ED FD: Improving the phi accrual failure detector. Technical Report IS-RR-2007-007, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan, April 2007. 1, 2