Leader Election

RAMCloud Lunch Talk

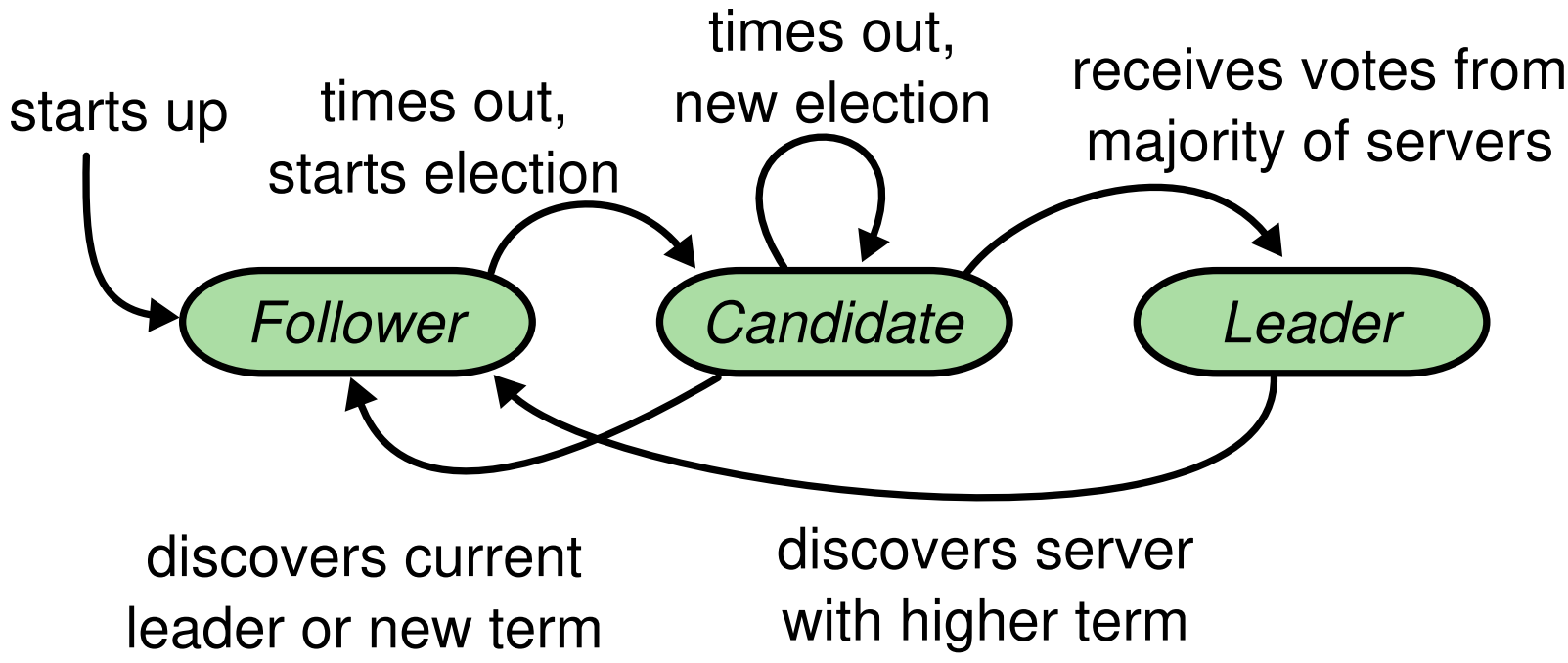
Diego Ongaro

December 12, 2013

Intro

- Leader election safety is easy
- Its performance/availability/liveness is hard to reason about
  - Flaky networks, down servers, partitions, reconfiguration
  - Very dynamic, state space explodes
- Had a known bug in reconfiguration, felt unsure of potential solutions
- Built a simulator to understand better and evaluate solutions

Current Algorithm (Basic)

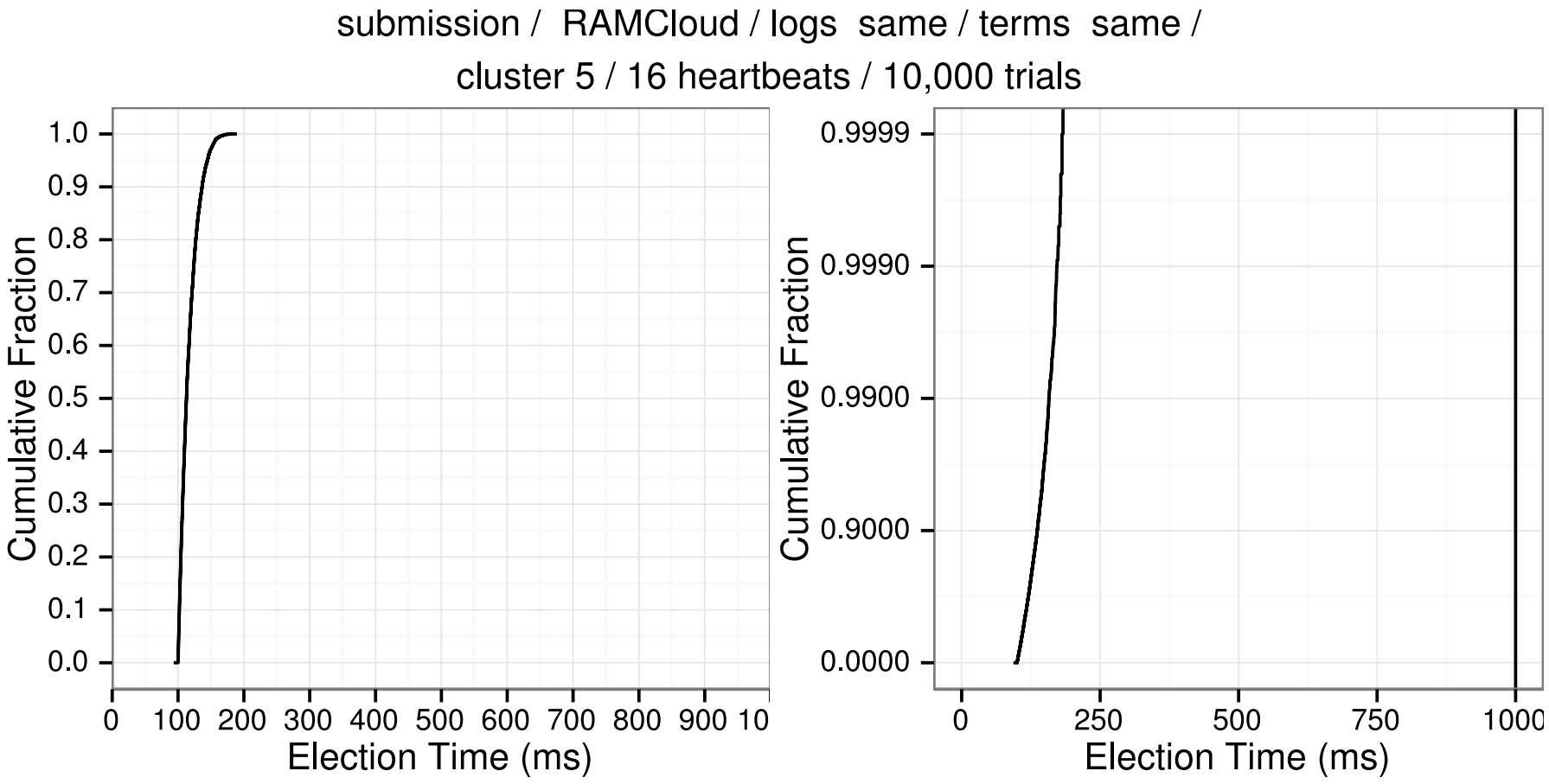


- Servers may only vote once per term
- Server increments its term number when starting new election
- Term numbers propagate across messages
- Start new election after random [100ms, 200ms] without receiving heartbeat from current leader or granting vote

Current Algorithm (Up-To-Date Comparison)

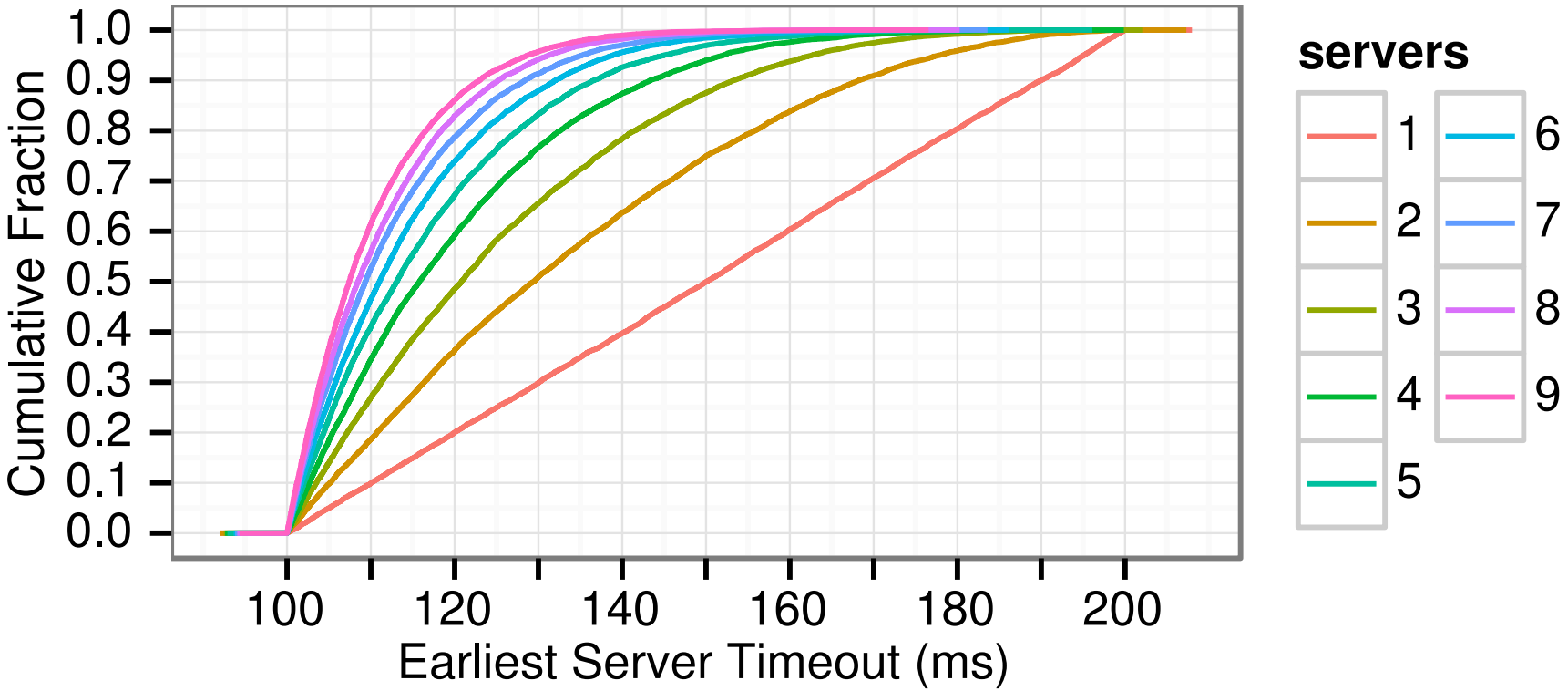
- RequestVote RPC includes "length" of candidate's log (it's slightly more complicated than that, but length will work for this talk)
- A voter will not vote for a candidate with a shorter log than its own
- $\Rightarrow$ elected leader's log is at least as up-to-date as majority of cluster
- Used to ensure Raft's safety properties

Normal Behavior (RAMCloud network)



- "RAMCloud": 5-10 microsecond one-way network latencies
- Works well, close to baseline (100ms)

Analytical Model

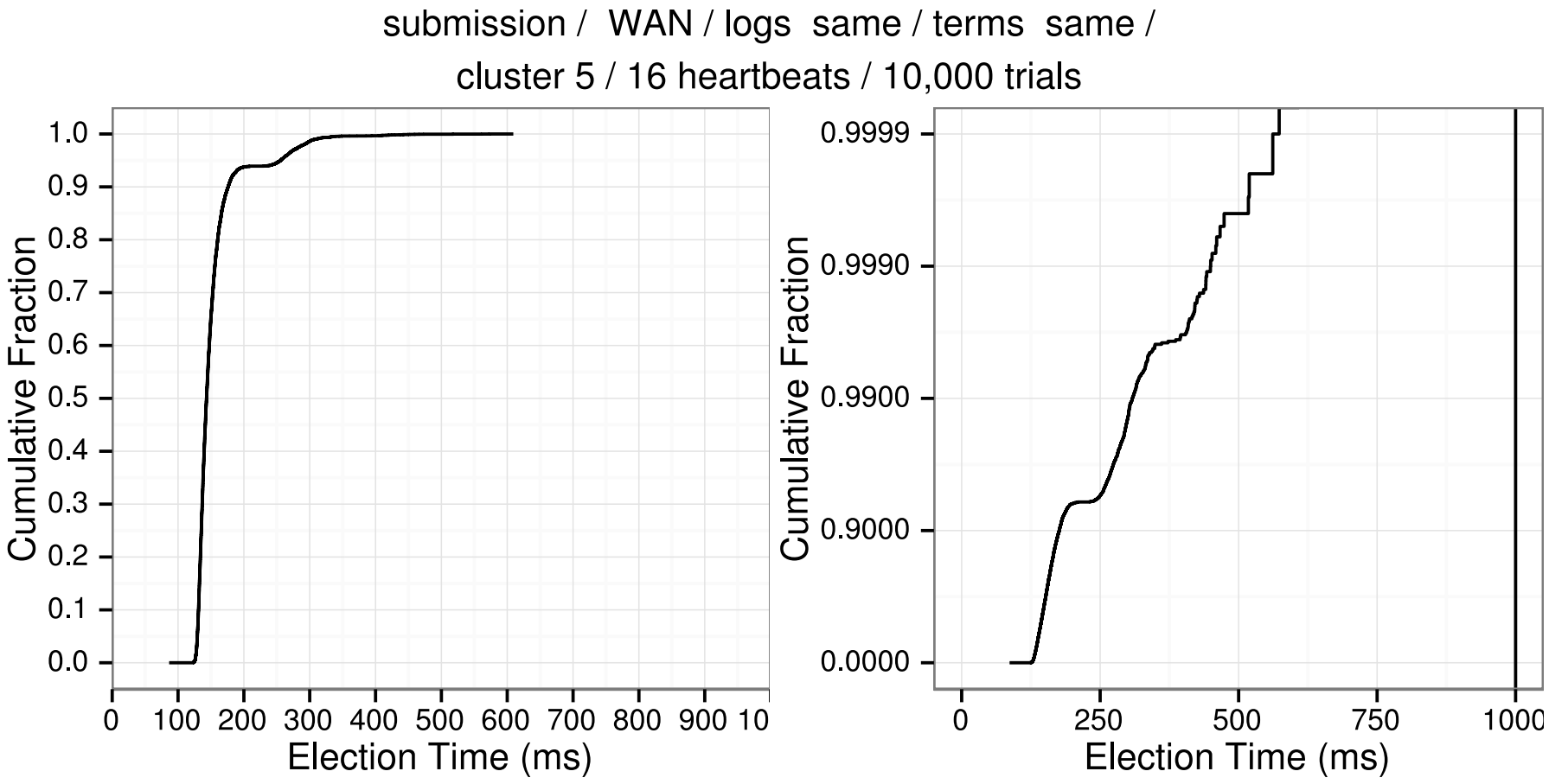


$$P(T < t) = 1 - (1 - t)^s$$

$$P(T = t) = s(1 - t)^{s-1}$$

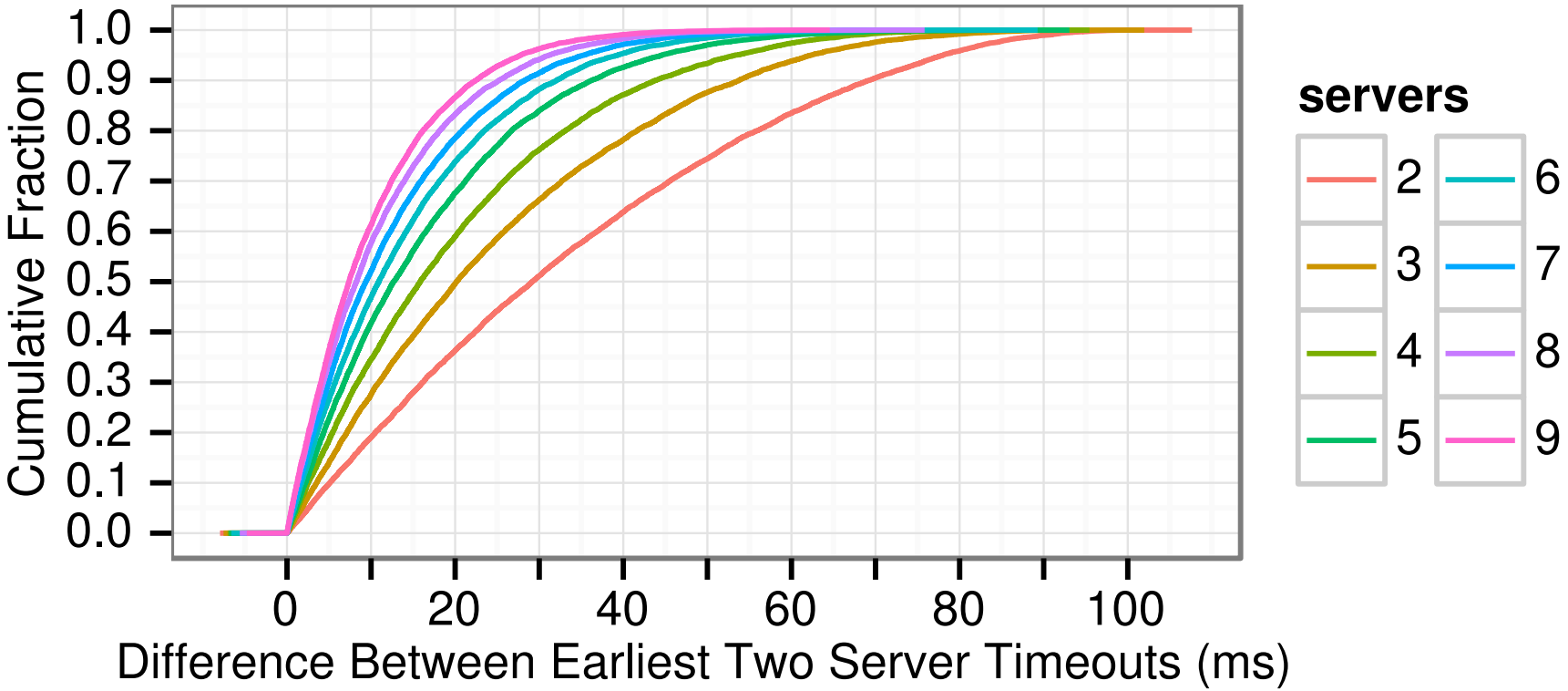
$$E[T] = \frac{1}{s + 1}$$

Normal Behavior (WAN network)



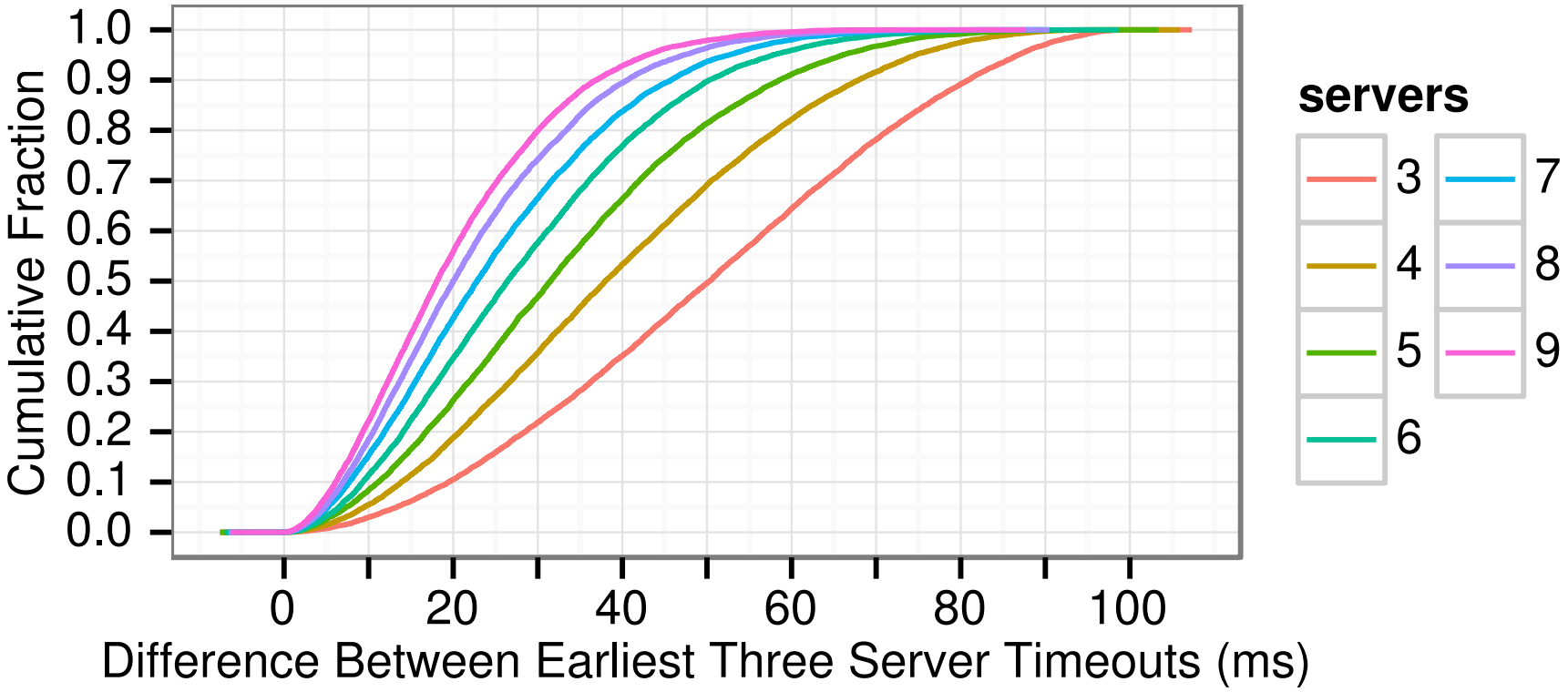
- "WAN": 10-20 millisecond one-way network latencies
- ~6% of elections a bit slower, why?

Analytical Model (2 candidates)



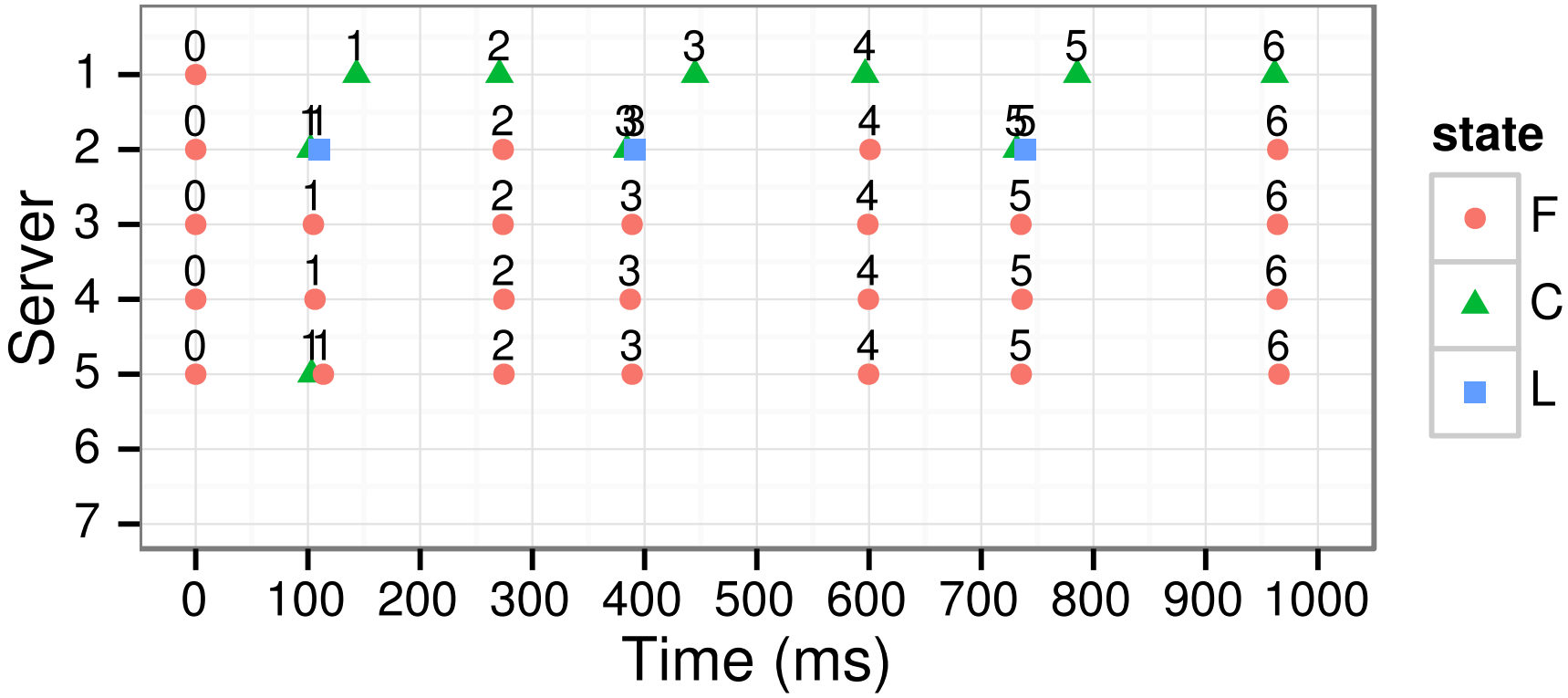
- Two concurrent candidates very frequent with WAN latency
- But two concurrent candidates ok, 1 can still get majority
- Difference has same distribution as earliest timeout?

Pseudo-Analytical Model (3 candidates)



- Three concurrent candidates 8% with 10ms latency
- What's this distribution?

Bad Receive

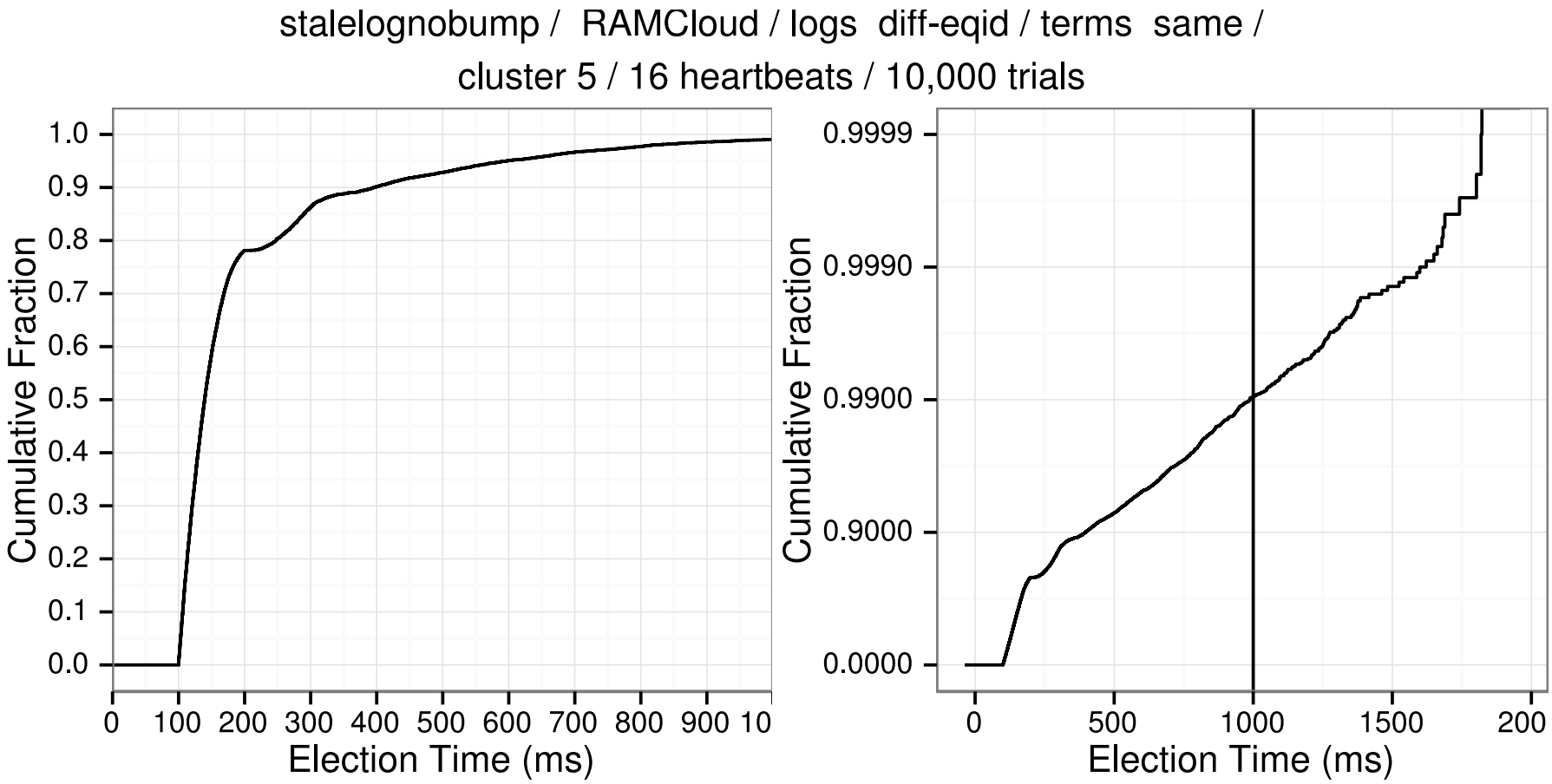


- Server 1 can send but can't receive messages
- Doesn't get heartbeats, disrupts leaders
- Somewhat Byzantine, but similar issue occurs when servers are removed from the cluster

Stale-Log-No-Bump Algorithm

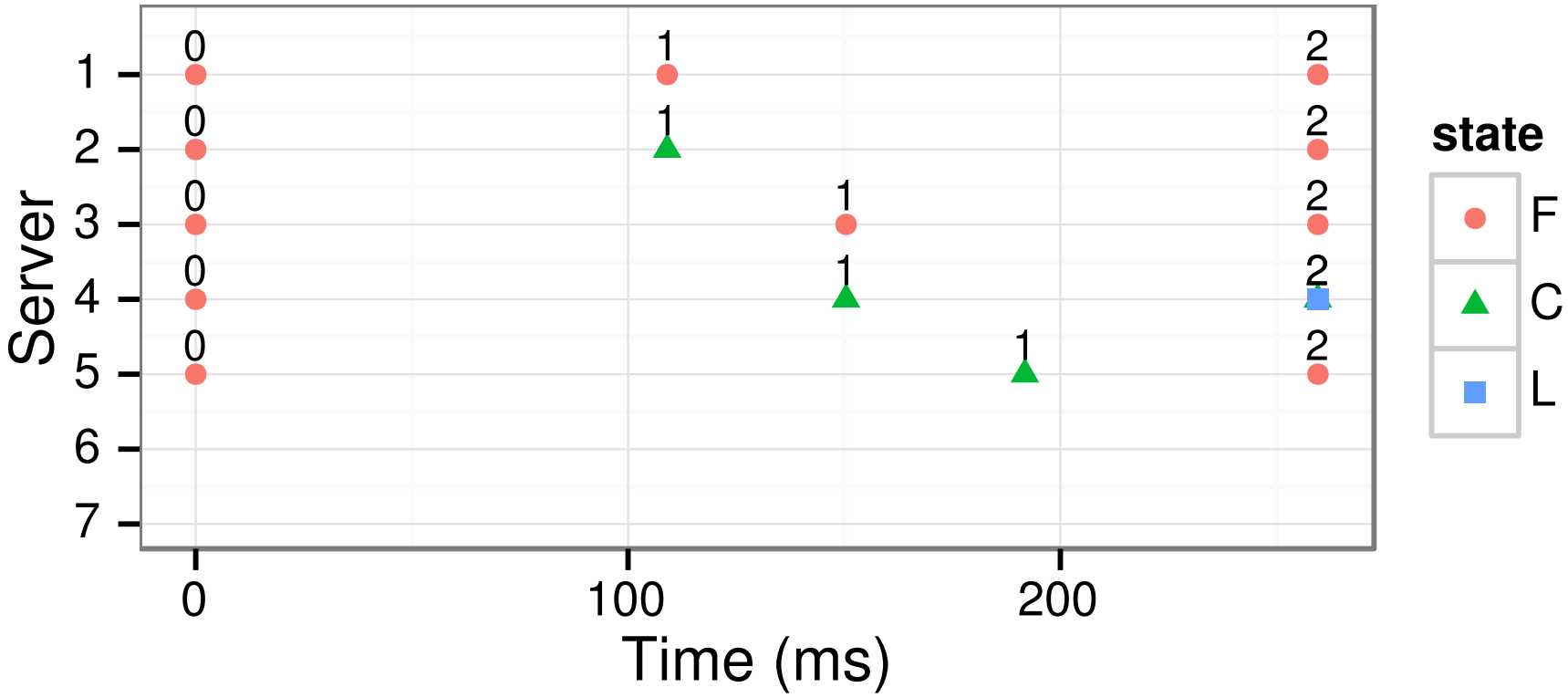
- Voter won't adopt candidate's term unless candidate's log is as up-to-date as voter's
- Idea: ignore RequestVotes from ineligible candidates
- Awkward: terms not quite logical clock anymore

Different Log Lengths (Distribution)



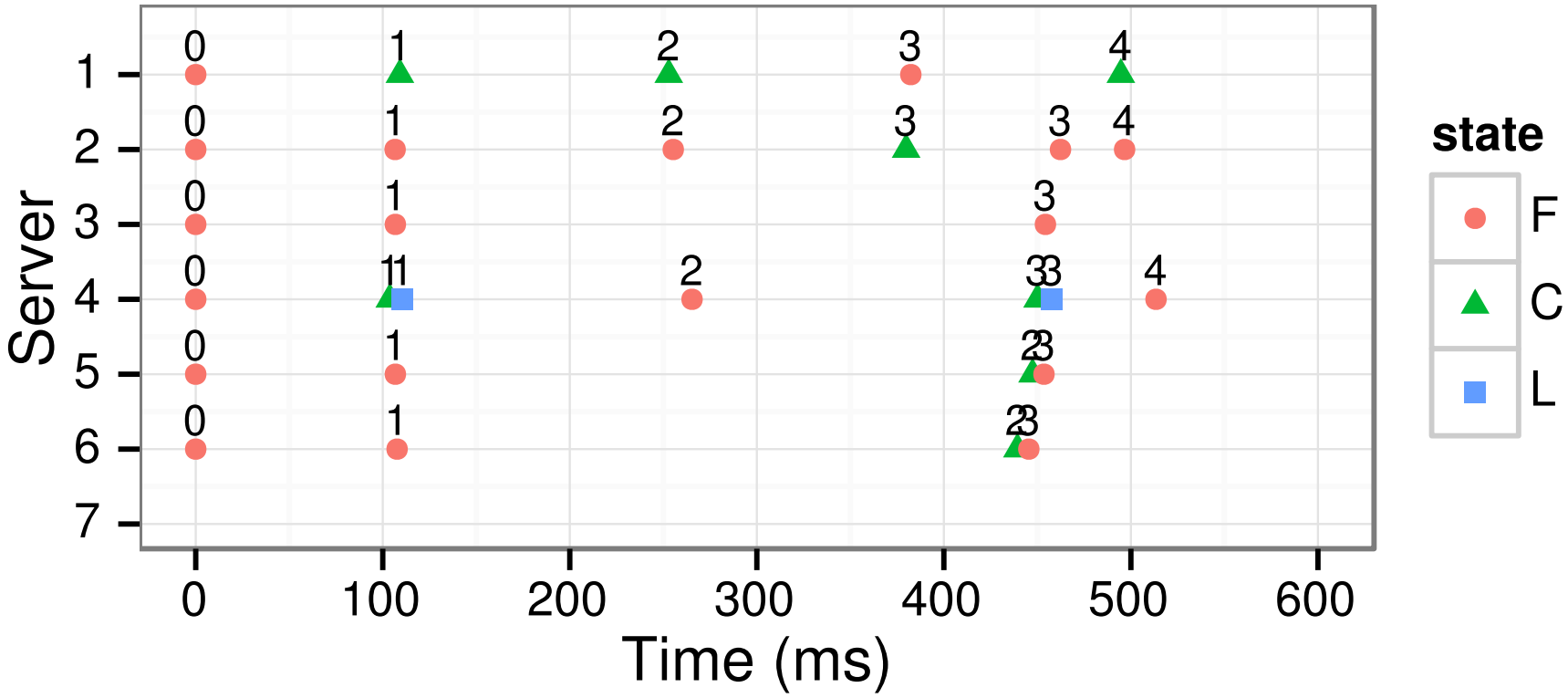
- Every server has a different log length $\Rightarrow$ only 3 servers eligible to be leader
- Still acceptable, but what's going on?

Different Log Lengths (Timeline)



- Server 1: log length 1 . . . Server 5: log length 5
- Ineligible servers tie up votes
- Eligible servers need to time out another time to increment their terms

Reconfiguration



- Reconfigure from S1-S5 to S2-S6
- Log lengths: S1:1, S2:1, S3:3, S4:3, S5:2, S6:1
- S1 disrupts S2, in turn disrupts S4
- Key problem: hard to bound time leader needs to update servers' logs
- Also fails for Bad Receive case when bad server has stale log

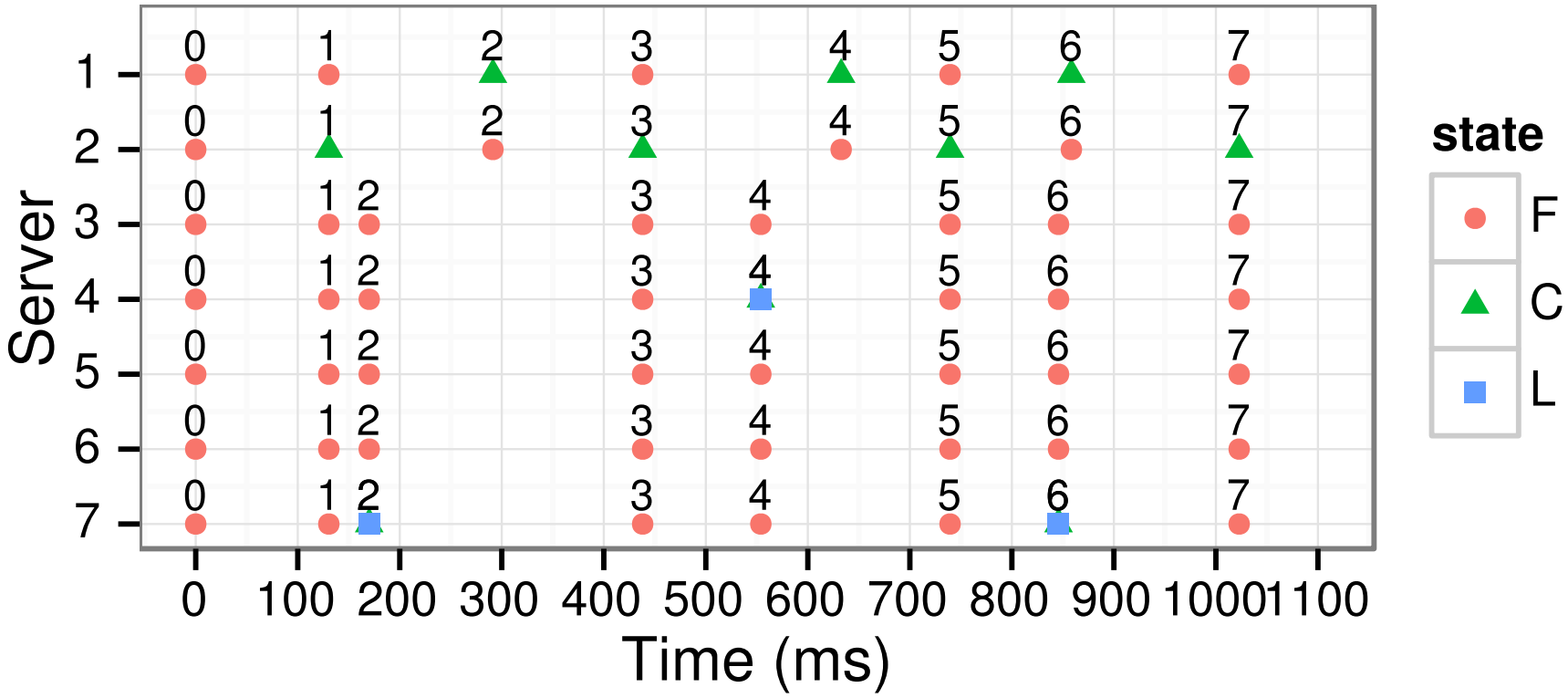
ZooKeeper Algorithm

- Pre-vote phase: before incrementing term, check to make sure your log is at least as up-to-date as a majority
- Works really well in all cases
- Decrease in performance on WAN?
- Large implementation change

Hesitant Algorithm

- Idea: Why do ZooKeeper's pre-vote phase all the time when most of the time we don't need it?
- Candidate only restarts new election in next term if a majority of voters say the candidate's log is at least as up-to-date as theirs
- Depends on property that if servers A's log is less up-to-date than server B, it remains that way until server A's log changes (not 100% true but probably true enough)
- Works well for Bad Receive case, reconfiguration with 1 server removed

Reconfiguration (2 servers)

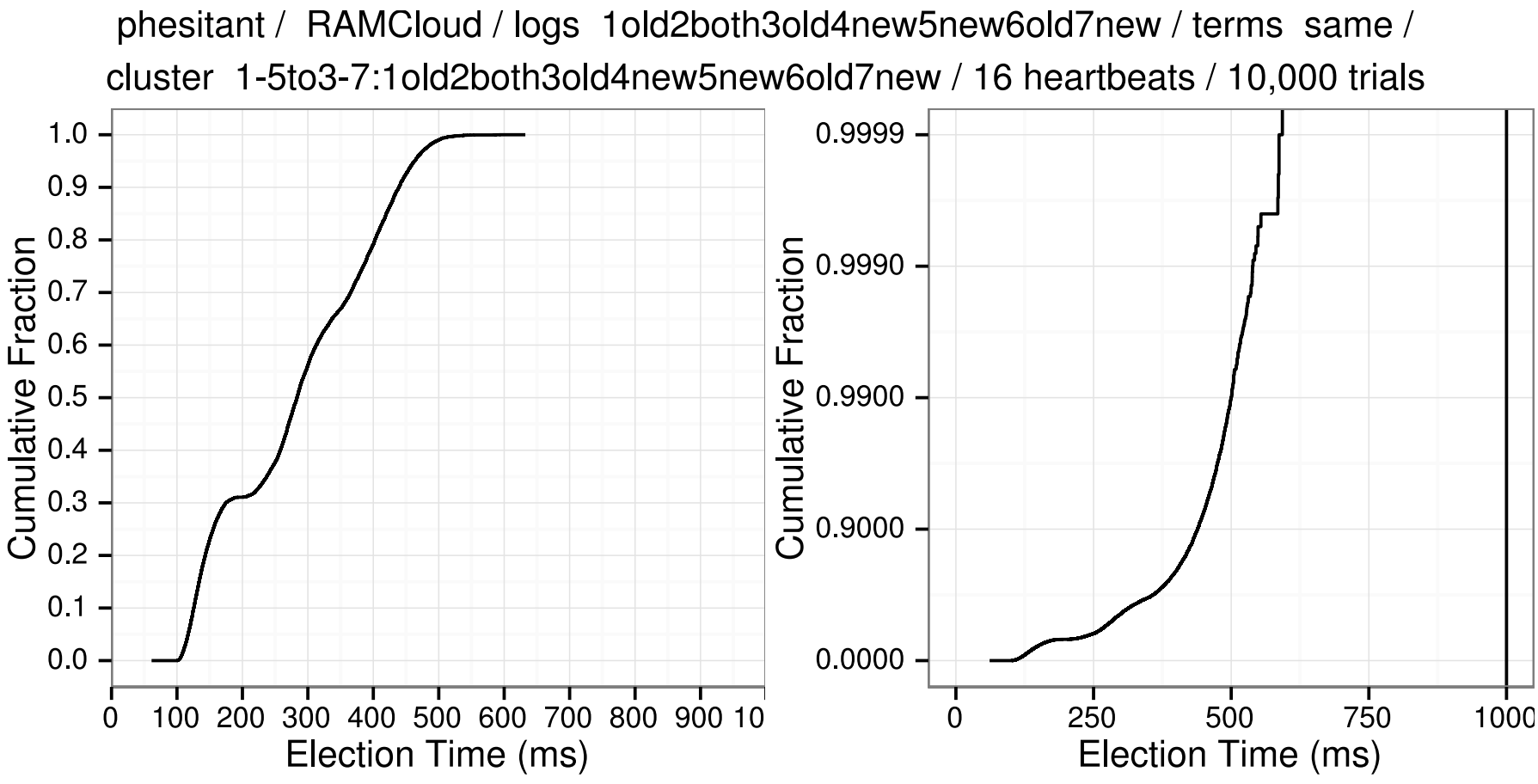


- Reconfigure from S1-S5 to S3-S7
- Log lengths: S1:1, S2:2, S3:1, S4:3, S5:3, S6:1, S7:3
- S2 disrupts cluster, then learns not to, then S1 makes S2 return to follower state
- Key problem: S2's amnesia

Persistent-Hesitant Algorithm

- What if servers remembered across terms that their logs were less up-to-date than others?
- Any way this is better than ZooKeeper (understandability)?

Reconfiguration (2 servers)



- Works ok
- Extra time needed for S1, S2 to collect rejections

Approaches Outside the Model

Multicast: send heartbeats on a well-known multicast address

- Won't fix Bad Receive alone, but handles reconfiguration cases
- Deployment concerns?

Leases: servers would ignore RequestVotes for a base election timeout period after receiving a heartbeat

- Trivial implementation
- Fragile: If any single server doesn't ignore the RequestVote, cluster will be disrupted (clock drift, overload, packet loss)
- Not easy to evaluate concerns in simulator

Conclusions

ZooKeeper pre-vote very robust and easy to understand

- Other approaches: broken, subtle, not general, or fragile
- Leverages existing properties: server won't be disruptive unless it knows it is eligible
- They had this pre-vote phase before they had reconfiguration?

Simulation nontrivial but paid off quickly

- Extremely valuable: being able to see detail at the right level
  distributions $>$ individual timelines $>$ full traces
- Real-time interactivity helpful