Escape to Precaution against Leader Failures

Edward (Gengrui) Zhang and Hans-Arno Jacobsen
University of Toronto

UNIVERSITY OF TORONTO

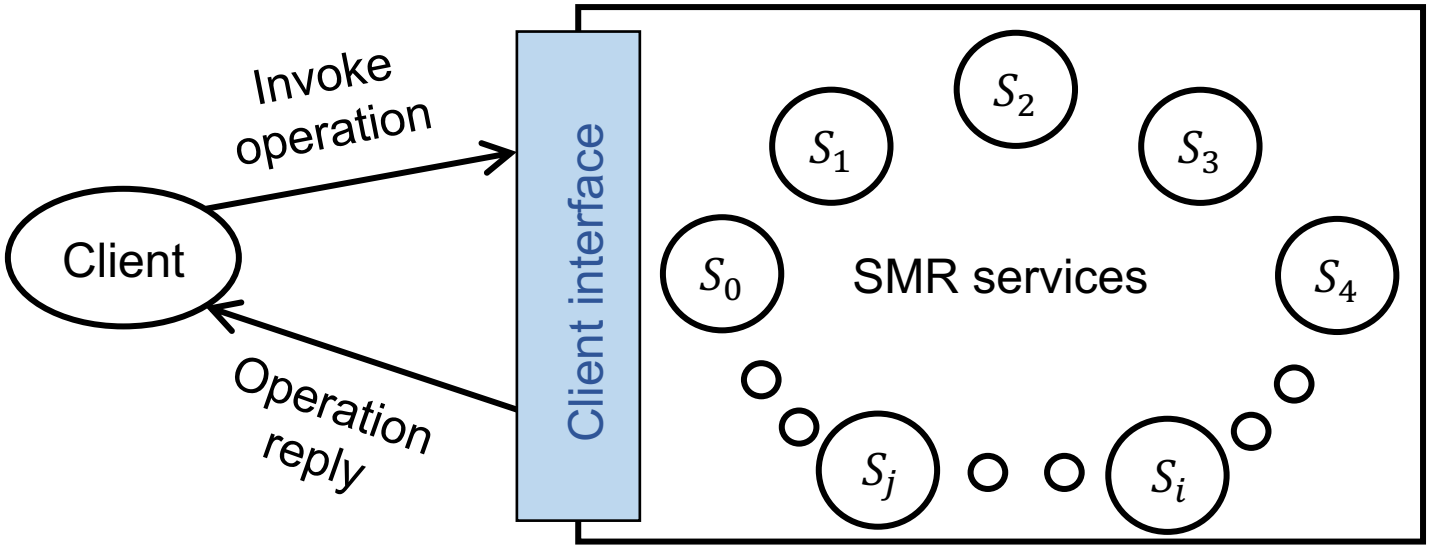
ICDCS'22

Content

- Consensus and state machine replication
- Leader-based consensus algorithms at a glance
- Problem statements: split votes in leader election
- **The Escape protocol** – avoiding split votes with fast leader election

Consensus and state machine replication

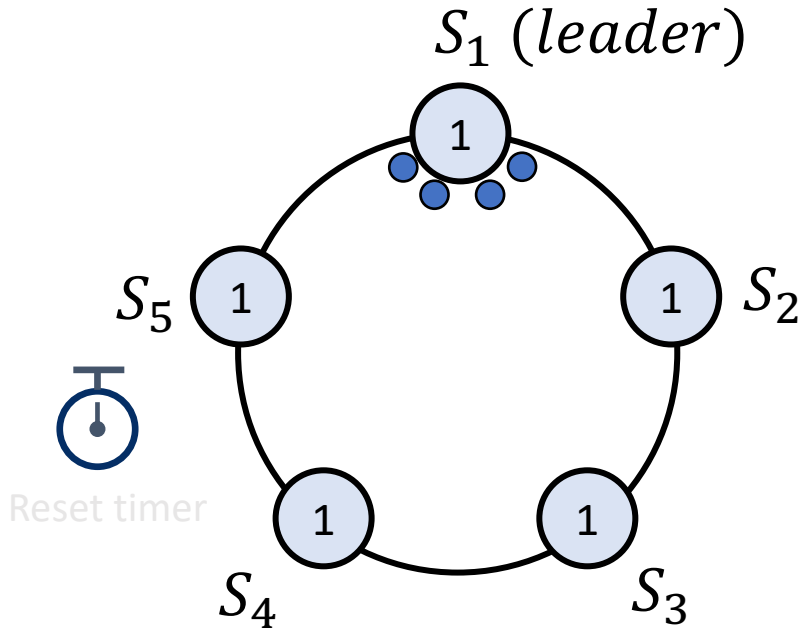
- Consensus algorithms stand at the core of distributed systems
  - Provide state machine replication (SMR) services
  - Coordinate server actions to reach agreement
  - Fault tolerance: Crash and Byzantine fault tolerance (CFT/ BFT)



Safety
All non-faulty replicas agree on a total order for the execution of requests despite failures

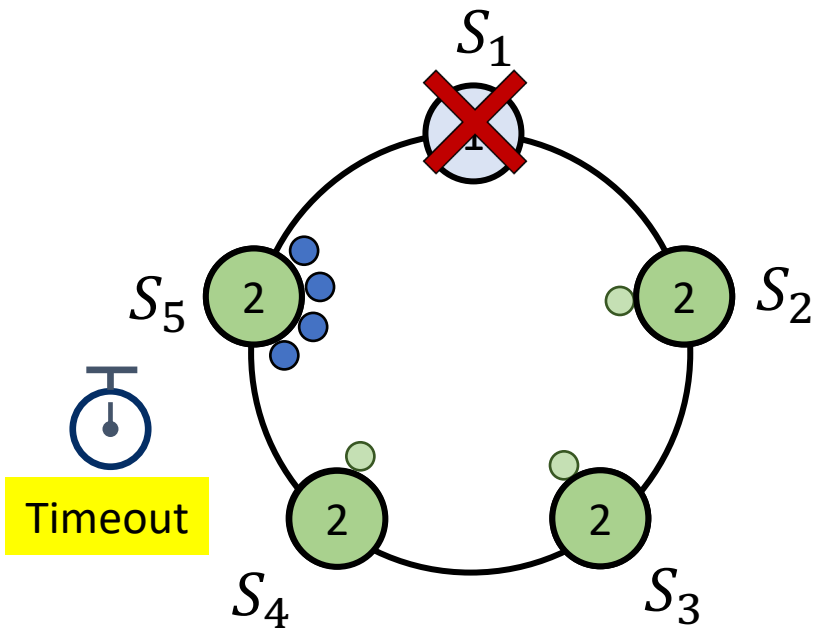
Liveness
Clients eventually receive replies to their requests

Leader, leadership, and Raft



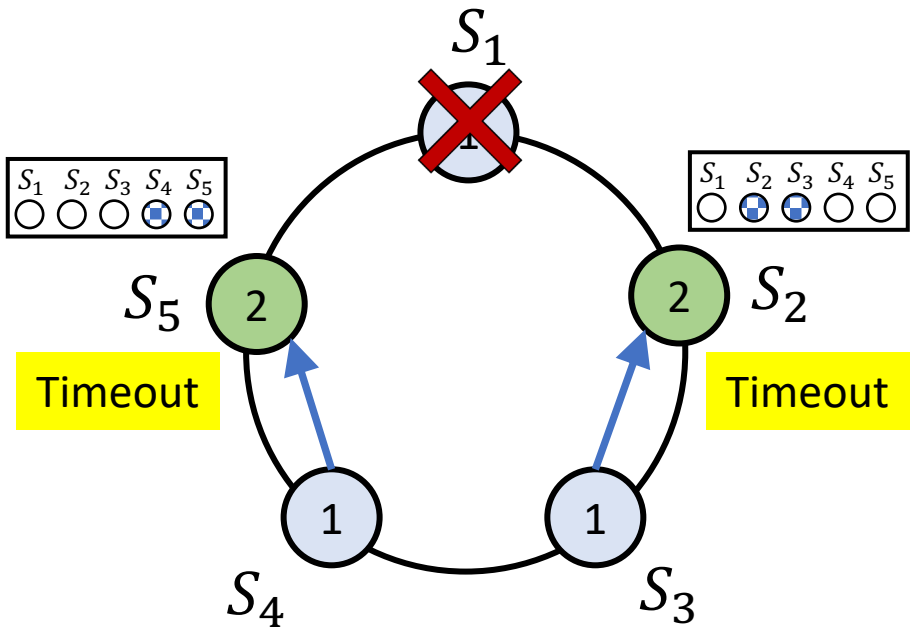
- Leader-based consensus algorithms have been widely developed and deployed
  - Paxos, Viewstamped replication, and **Raft**
- Raft's strong leadership: log entries only flow from the leader to other servers
  - *Heartbeats* – *periodic messaging shuttle*
  - *Terms* – integers representing logical time
  - Three server roles: *leader*, *follower*, and *candidate*
- Raft operates in two phases
  - Log replication when leader is correct
  - Leader election when leader fails

Leader election – when a leader falls



- Candidate starts a new leader election campaign
  - Increments its term
  - Broadcasts a leader election request
  - Votes for itself
  - Needs to collect a majority vote in current term
- A server grants a vote if
  - Candidate's log is at least as up-to-date as its log
  - Candidate's term is not less than its term
  - It has not voted in the current term

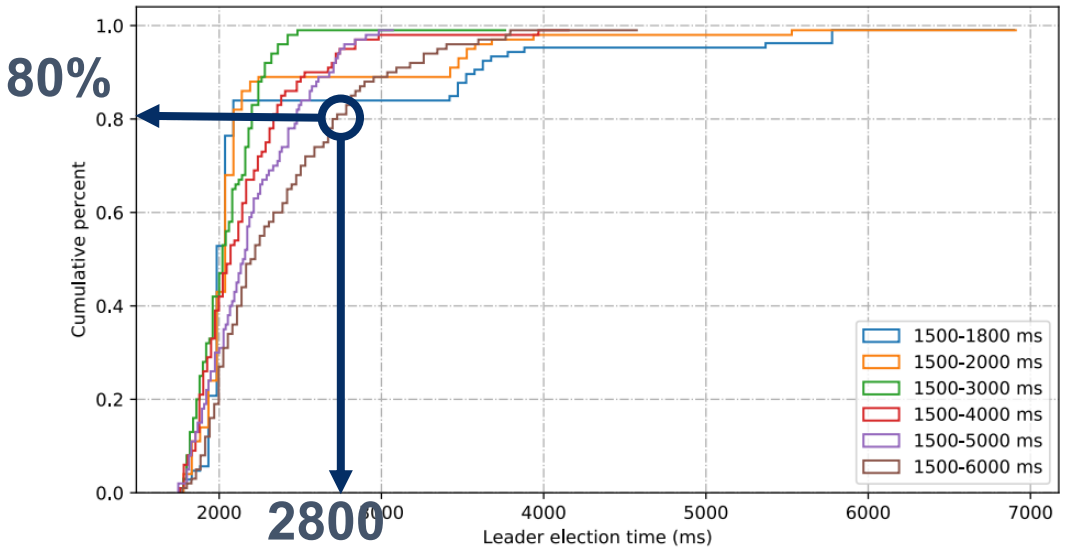
Competition in Raft's leader election



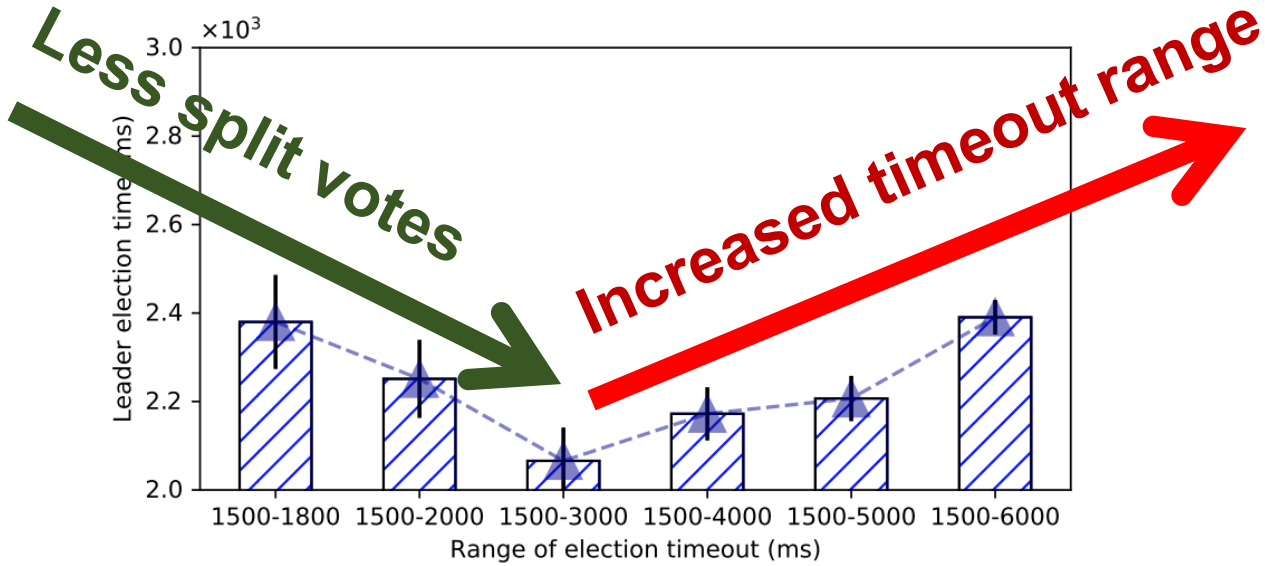
- Split votes: what if no one collects a majority vote?
  - E.g., in a 5-server cluster, the leader is down with 4 servers remaining. Each of the two candidates collects only two votes (one from itself)
- Candidates need to wait for a new timeout
- Split votes significantly increase leader election time (no leader, no service)

Impact of split votes

- Prolonged election time due to split votes



- 1000 runs in a 5-server cluster, 100-200ms network latency
- 80% of evaluation results are less than 2800ms. I.e., 20% of the 1000 runs took more than 2800ms



- Expanding timeout range can mitigate split votes but does not always yield to best configuration
- Best configuration changes adaptively according to network conditions

Escape to precaution against leader failures

- Escape is a leader election protocol that addresses split votes using two key concepts to dynamically prioritize servers
  - Stochastic configurations assignment (SCA)
    - Assigns each server a configuration contains a unique priority and a timeout value
  - Probing patrol function (PPF)
    - Rearranges server priorities based on their log responsiveness
- SCA and PPF work in concert to prepare a pool of candidates as “future leaders” with differently prioritized configurations
- Escape is a general-purpose leader election protocol (not restricted to Raft); it can be adopted by various election algorithms

Stochastic configurations assignment (SCA)

A configuration, $\pi^{\mathcal{P}_i}$, contains:

- Unique priority $(\mathcal{P}_i)$ such that $\mathcal{P}_i = i\ (server\ ID\ of\ S_i)$
- Timeout value $(period_i)$ such that $period_i = baseTime + k(n - \mathcal{P}_i)$

E.g., $baseTime = 100ms, k = 10, n = 10$

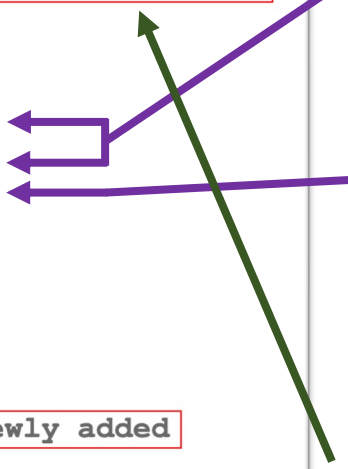
S_1 (1) $\pi^{\mathcal{P}_1} = \begin{cases} \mathcal{P}_1 : 1 \\ period_1 : 190\ ms \end{cases}$ ⇨ (2)

S_{10} (1) $\pi^{\mathcal{P}_{10}} = \begin{cases} \mathcal{P}_{10} : 10 \\ period_{10} : 100\ ms \end{cases}$ ⇨ (11)

- Priority determines term growth
- Timeout impacts speed for leader failure detection

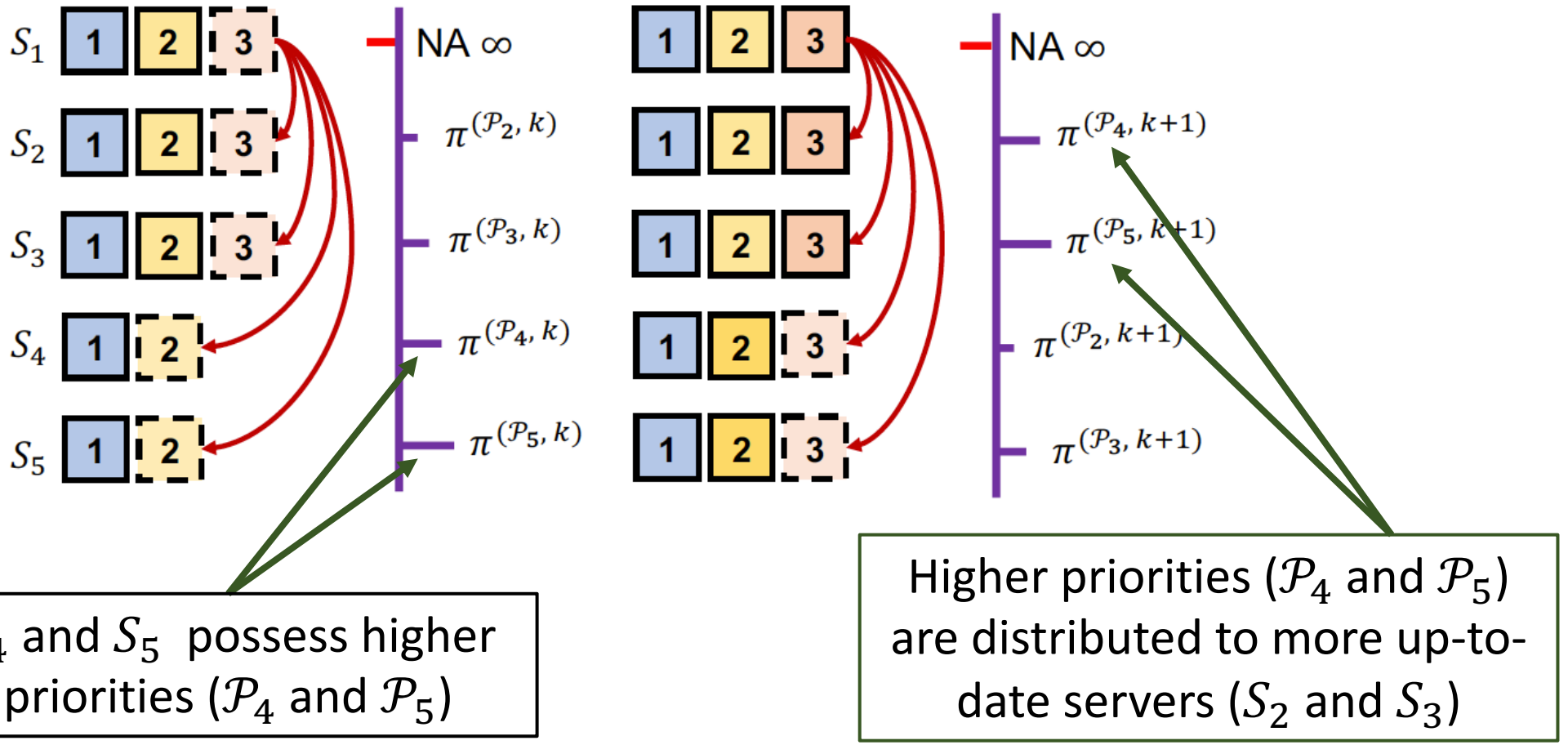
Probing patrol function (PPF)

```
//the parameters of AppendEntries RPCs
type AppendEntriesArgs struct{
    term          int64
    leaderId      string
    prevLogIndex  int64
    prevLogTerm   int64
    entries[]     Entries
    leaderCommit  int64
    newConfig     Configurations //newly added
}

type Configurations struct{
    timerPeriod time.Duration
    priority    int64
    confClock   int64
}

//the reply messages
type AEReplyArgs struct{
    term       int64
    success    bool
    status     configStatus //newly added
}

type configStatus struct{
    LogIndex     int64
    timerPeriod time.Duration
}
```

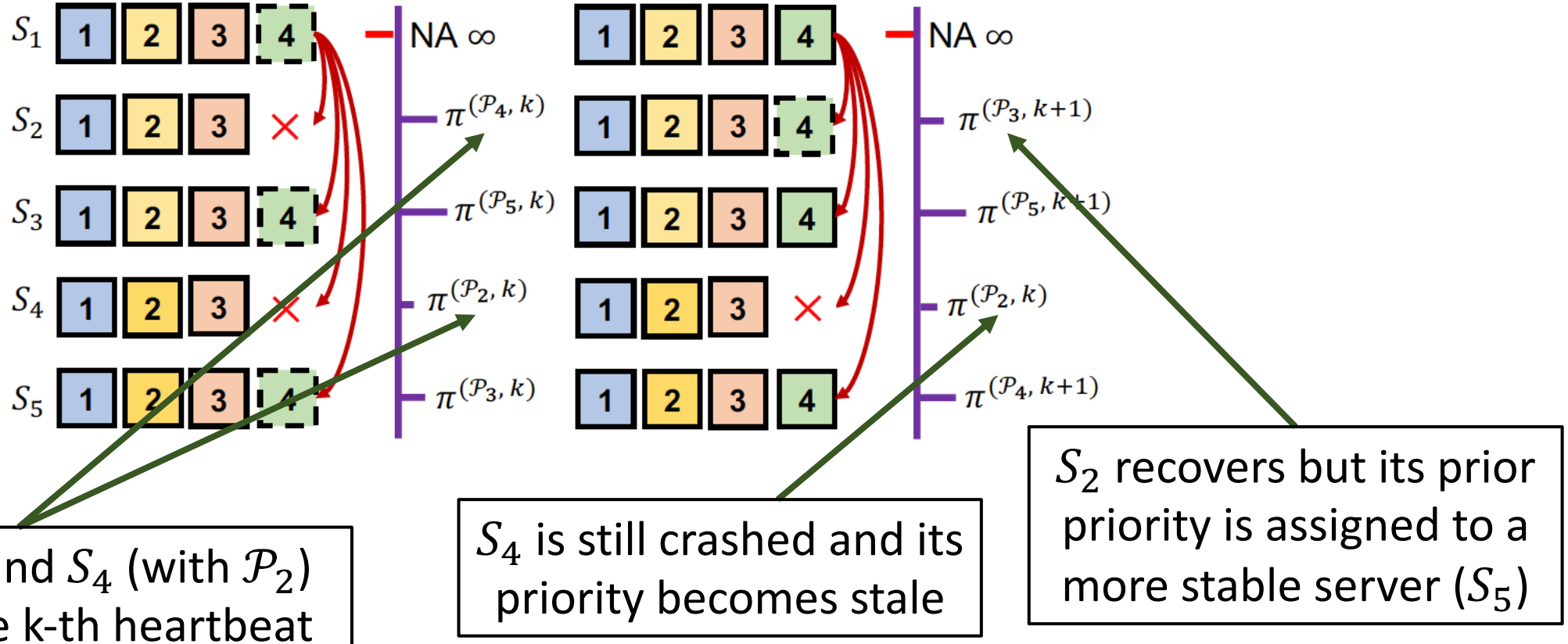


- In each heartbeat, a leader arranges configurations
  - Collects old configurations and rearranges new configurations based on server log responsiveness
  - Assigns logical clocks to new configurations, indicating the freshness of configurations
  - Distributes a new configuration to other servers
- PPF can be decoupled from regular heartbeats
  - E.g., less frequent rearrangement if network is more synchronous

Examples – higher log responsiveness, higher priorities



Examples – stale configurations of crashed servers



S_2 (with $\mathcal{P}_4$) and S_4 (with $\mathcal{P}_2$) crashed in the k-th heartbeat

S_4 is still crashed and its priority becomes stale

S_2 recovers but its prior priority is assigned to a more stable server (S_5)

Design philosophy of leader election protocols

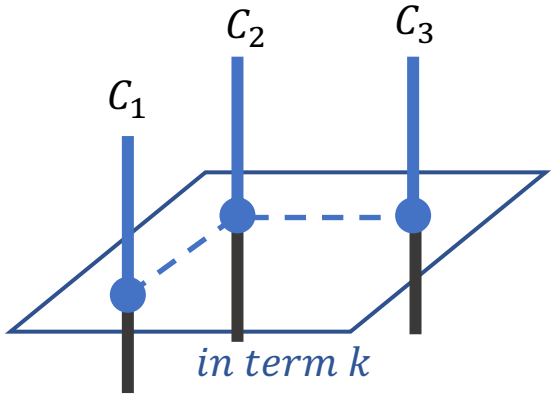
Raft

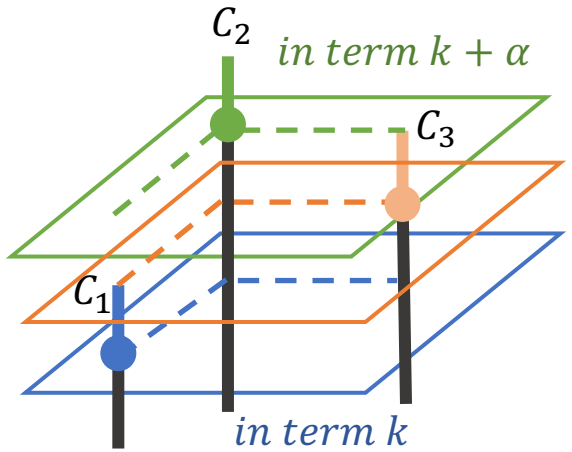
- All servers are created equal
  - Every server has an equal chance to be the next leader
  - The candidate who collects votes faster (a majority vote) is more likely to be elected
- Leadership competition may take place when a leader fails

Escape

- Servers are born with priorities
  - A higher-priority candidate is more likely to defeat its counterparts and win an election campaign
- A queue of future leaders is maintained; leadership competition is resolved before a leader fails

Escape to leadership competition

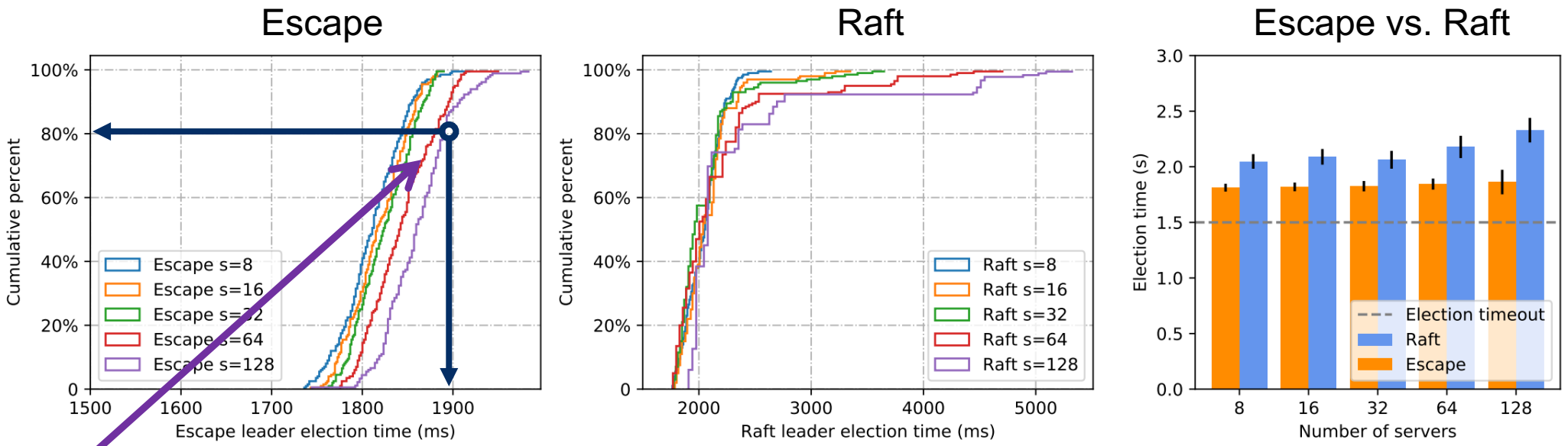




Raft intends to rank competing candidates whose campaigns are in the same term

ESCAPE uses priority-based configuration assignments to distribute concurrent campaigns into different terms

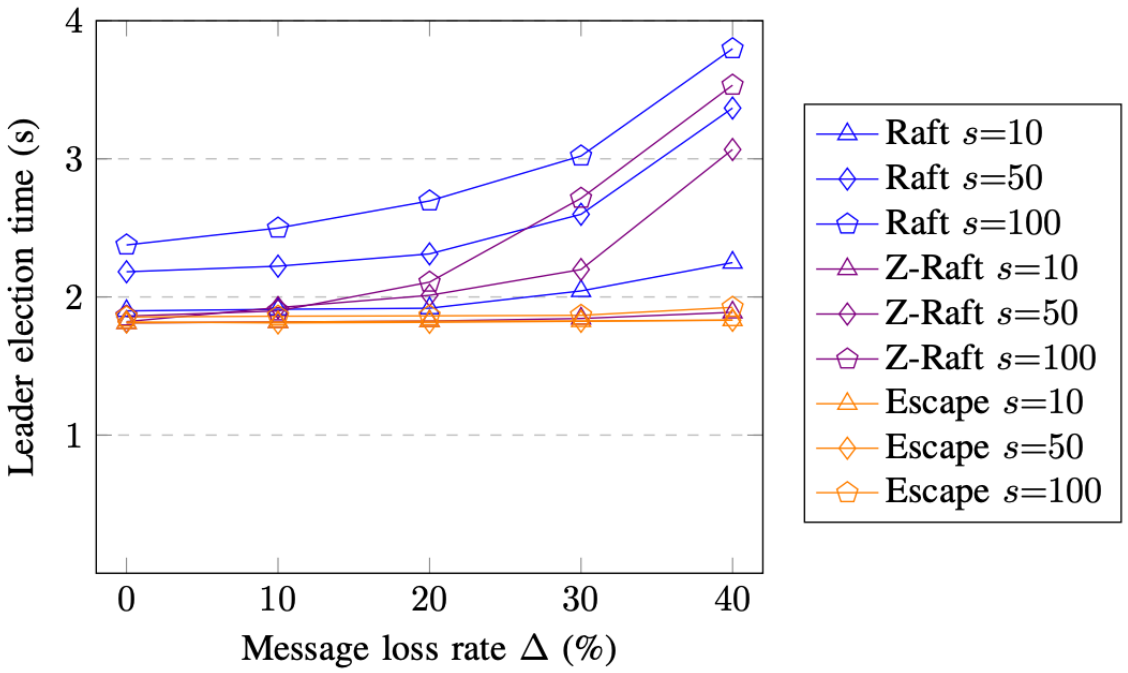
Evaluation – election time under leader failures



80% of the 1000 runs are less than 1900ms. I.e., 20% of runs took more than 1900ms

- Prototypes deployed on 4, 8, 16, 32, 64 and 128 VMs
  - Timeouts set to 1500-2000ms; network latency varies from 100-200ms
  - Each curve shows the cumulative percentage of 1000 runs;
- Escape converges leader election faster at all scales
  - Compared with Raft, Escape shortens leader election time by 11.6% and 21.3% at sizes of 8 and 128 servers, respectively

Evaluation – election time under message loss



- Z-Raft (Zookeeper variant)
  - No rearrangement of configurations

- Under higher message loss rates, configuration rearrangements become more effective as no stale candidate can possess high priorities
- In 10-server cluster, compared with Raft, Escape reduces election time by 9.6% and 19% under Δ=10% and Δ=40%, respectively
- In cluster of 100 servers, Escape reduces the leader election time by 21.4% and 49.3% when Δ=10% and Δ=40%, respectively

Conclusions

- Escape fundamentally resolves split votes by dynamically prioritizing servers according to their log responsiveness
  - A more up-to-date server receives a better configuration that leads it to run an undefeated leader election campaign
  - A pool of differently prioritized candidates is prepared before a future leader election takes place
- Escape progressively reduces leader election time when the cluster scales up, and the improvement becomes more significant under message loss

Thank you for listening!

Gengrui (Edward) Zhang, PhD candidate

University of Toronto

More on https://gengruizhang.github.io/