RABIA: SIMPLIFYING STATE-MACHINE REPLICATION THROUGH RANDOMIZATION

Srujana Golconda

50442205

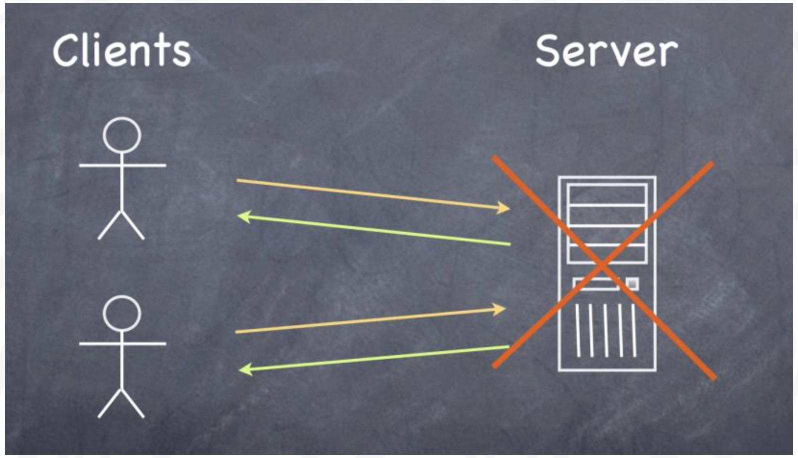
03/29/2023

University at Buffalo The State University of New York

State Machine Replication

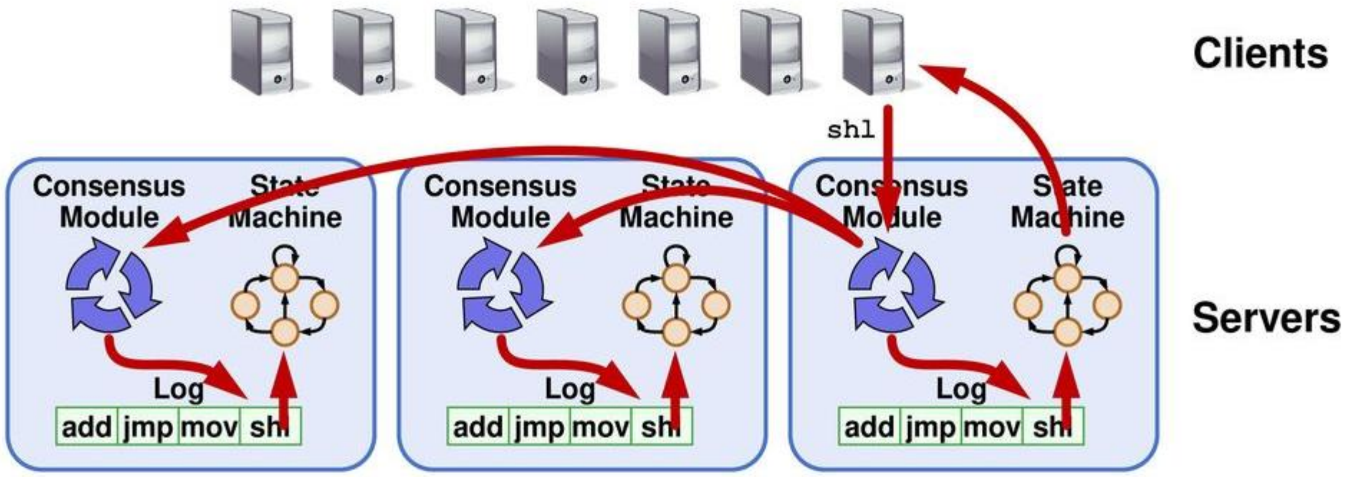
1. Fault Tolerance: The application must be able to, fully or partially, operate when some part of the system fails.

2. Transparency: Regardless of the actual number of servers in the network, the output the client sees should be the same as if the input is being processed by a single highly available server.



State Machine Replication

- Data Replication in Distributed Systems enhances fault tolerance, reliability, and accessibility by maintaining consistency amongst redundant resources such as software and hardware components.
- State-Machine Replication (SMR) uses replication to ensure that a service is available and consistent in the presence of failures.



SMR:
- Behave as if service is provided by single machine.
- Use Consensus to agree on order of client requests.

Consensus Algorithms

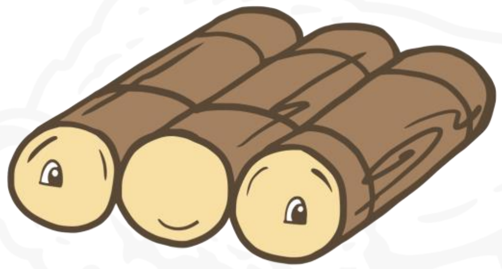
PAXOS:

- Paxos and variants had mostly been the de facto choice for implementing SMR.
- The intuition behind Paxos and variants is difficult to grasp.

"There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system . . . the final system will be based on an unproven protocol."

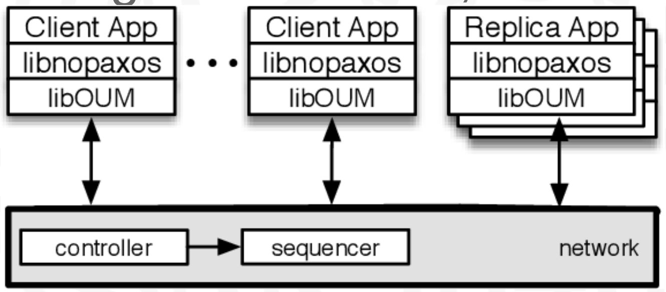
RAFT:

- Addresses understandability by using stronger notion of leader
- Adopted by Cockroach DB, Redis, Rethink DB and many more
- Still very difficult to implement due to engineering complexity.



Network Order PAXOS:

- NOPaxos uses the network fabric to sequence requests (i.e., a sequencer in switches) to simplify design and improve performance.
- Most public clouds do not allow users to directly implement their own logic in switches, which makes NOPaxos less adoptable.

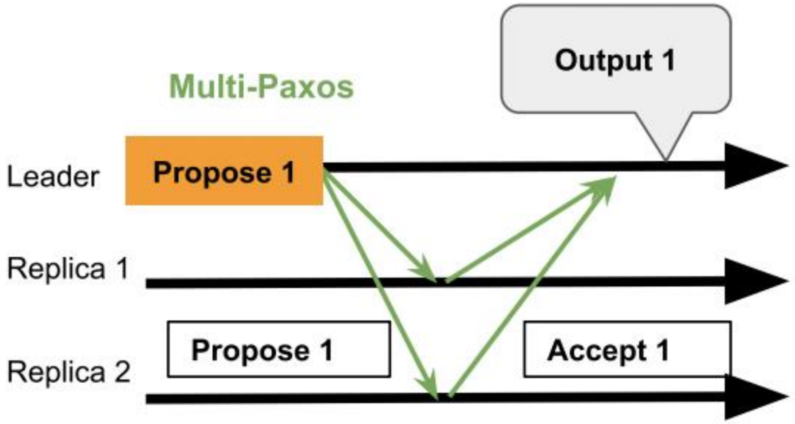


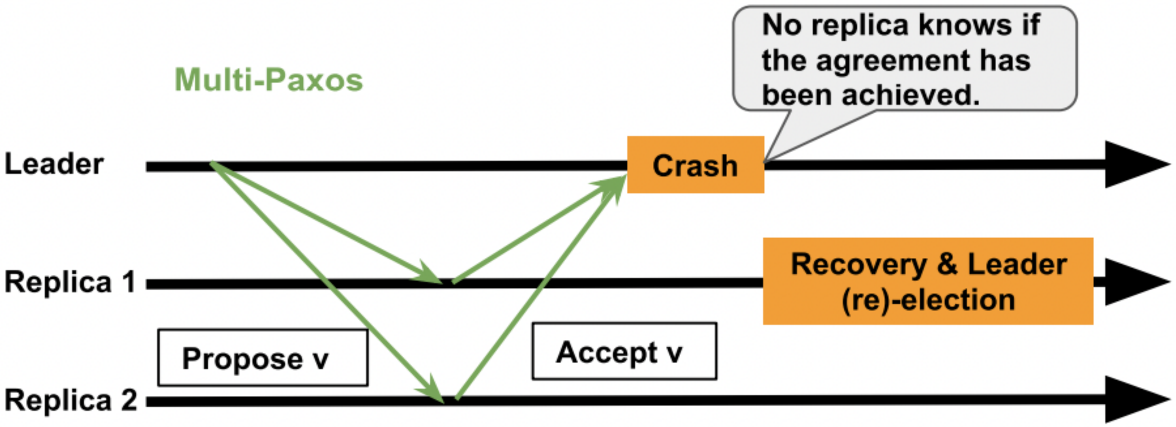
Why Another Simple Consensus Algorithm?

Consensus

Auxiliary Protocols
- Fail-over
- Reconfiguration
- Log Compaction
- Snapshotting

Leader-based Consensus: Challenge



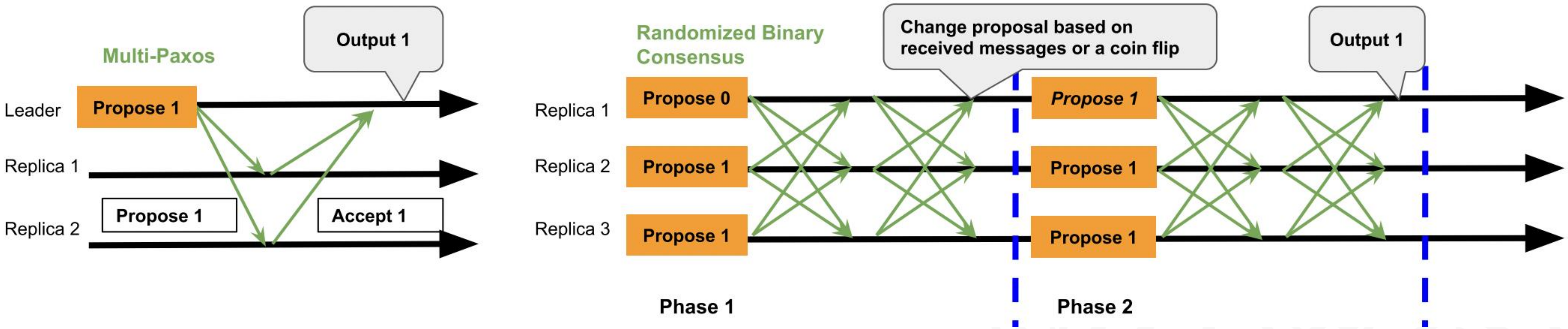


RABIA: Randomized Binary Agreement

Majority of replicas see similar set of messages

- State Machine replication designed for stable network
- Leaderless asynchronous consensus protocol.
- No fail-over protocol and supports simple log compaction.
- Better than Multi Paxos and E Paxos in 3 server cluster (same availability zone)
- Based on Ben-Or simple randomized binary consensus algorithm.

Ben-Or's Algorithm



All the replicas proceed in phases, in which they need to propose and make a "joint decision." Replicas may propose different values. In this case, replicas use a randomized rule (i.e., the outcome of a coin flip) to break a tie.

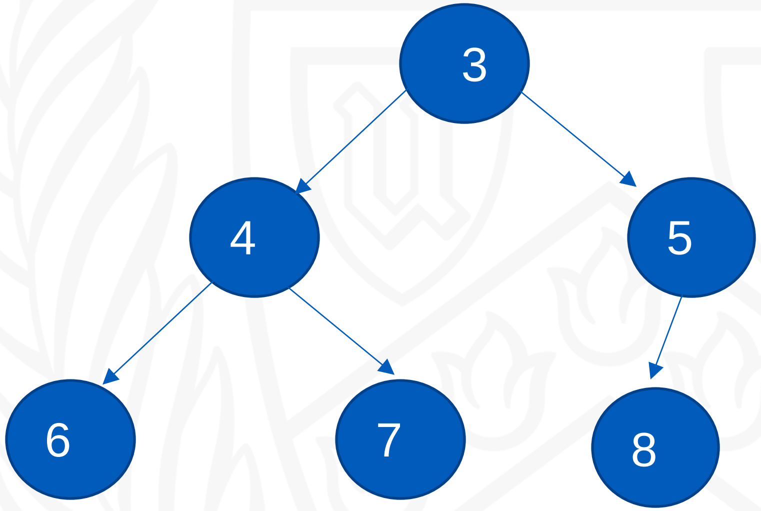
1. All-to-all communication
2. Exponential #rounds
3. Unstable performance

Challenges for building SMR systems based on randomized consensus algorithm

- Most randomized algorithms have less stable performance due to randomized rules that depend on the outcome of a “coin flip”.
- Latency in randomized consensus algorithms is sub optimal.
- Best case latency is worse than Multi paxos.
- Challenge Rabia addresses is to efficiently convert binary consensus into a (weaker) form of multi-valued consensus that can be used for SMR.
- How to stay on fast path as often as possible.
- How to avoid long tail latency in a stable network.

Key Techniques

- In Rabia, a client sends requests to an assigned replica, which then relays them to all the other replicas.
- Each replica stores pending requests that are not added to the log yet in its local min priority queue (PQ).
- Replicas then use consensus algorithm, Weak-MVC, to agree on the request for each slot of the log. Each replica's input to Weak MVC is the oldest pending request in its local PQ.



Addressing performance challenges of randomized consensus

- Stable network makes randomized consensus fast - all replicas have the same proposal.
    i. Each replica forwards a request to other replicas when it receives an incoming request and
    ii. If message delay is small compared to the interval between two consecutive requests, then it is highly likely that most replicas have the same oldest pending request r.
- Weak Mvc a novel implementation of a relaxed version of multi-valued consensus.
- **Weak Validity:** the value stored in each slot of the log must either be a request from some client or a NULL value ⊥.

Obtaining a No OP is faster than obtaining a request

- If replicas propose different requests - design Weak-MVC in a way that if it seems difficult to terminate fast, then replicas choose to forfeit the proposal, and Weak-MVC outputs a NULL value ⊥ in this case.
- The oldest proposal that has not been agreed upon is highly likely to be propagated to all the replicas in a stable network. These replicas would then store this proposal in local PQs, and Rabia can hit the fast path again on the next slot after forfeiting.

Slot#	0	1	2
Request	NO-OP	v0	

The Rabia SMR Framework

Algorithm 1 Rabia: Code for Replica N_i

Local Variables:

PQ_i ▷priority queue, initially empty

seq ▷current slot index, initially 0

Code for Replica N_i:

while *true* **do**

  $proposal_i \leftarrow$ first element in PQ_i that is not already in log

    ▷$proposal_i = i$'s input to Weak-MVC

  $output \leftarrow \textsc{Weak-MVC}(proposal_i, seq)$

  $log[seq] \leftarrow output$ ▷Add $output$ to current slot

  if $output = \perp$ **or** $output \neq proposal_i$ **then**

    $PQ_i.push(proposal_i)$

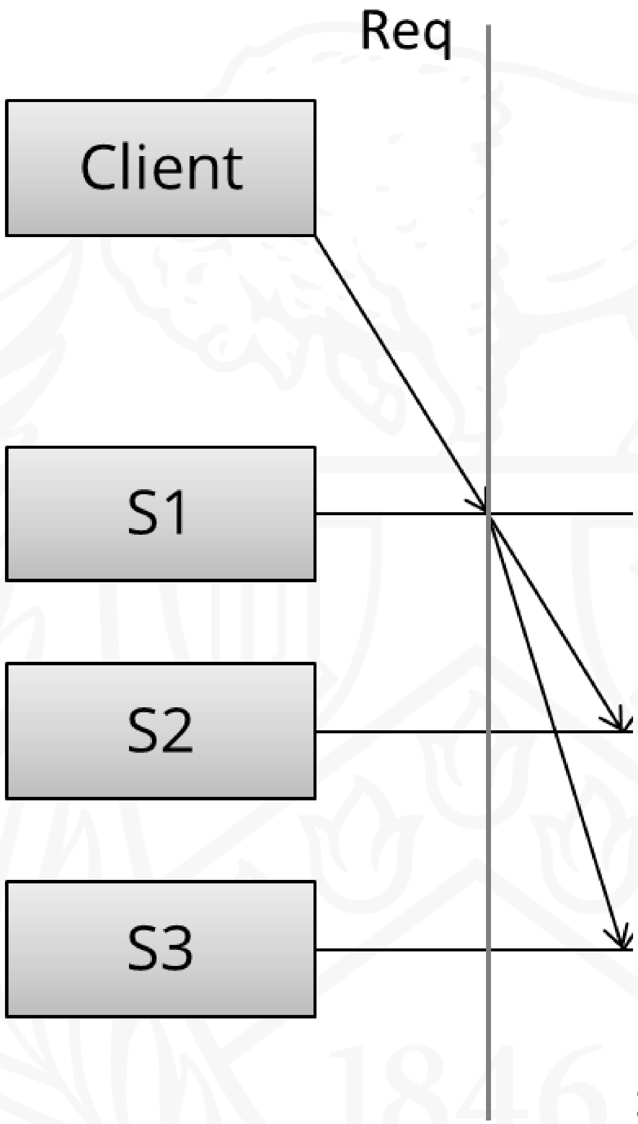
  $seq \leftarrow seq + 1$

/* Event handler: executing in background */

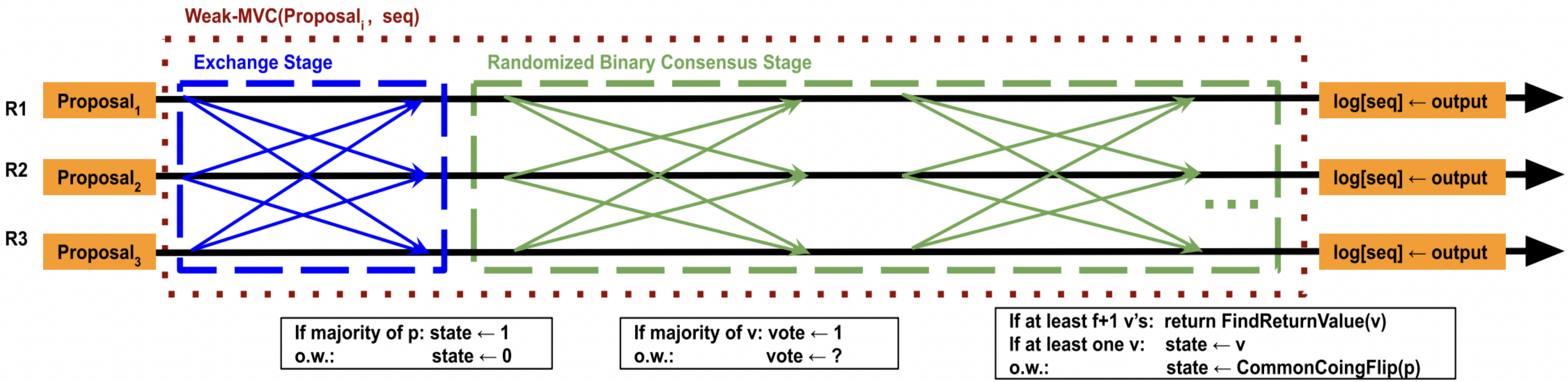
Upon receiving $\langle \textsc{Request}, c\rangle$ **from client** c:

$PQ_i.push(\langle \textsc{Request}, c\rangle)$

forward $\langle \textsc{Request}, c\rangle$ to all other replicas



Weak MVC

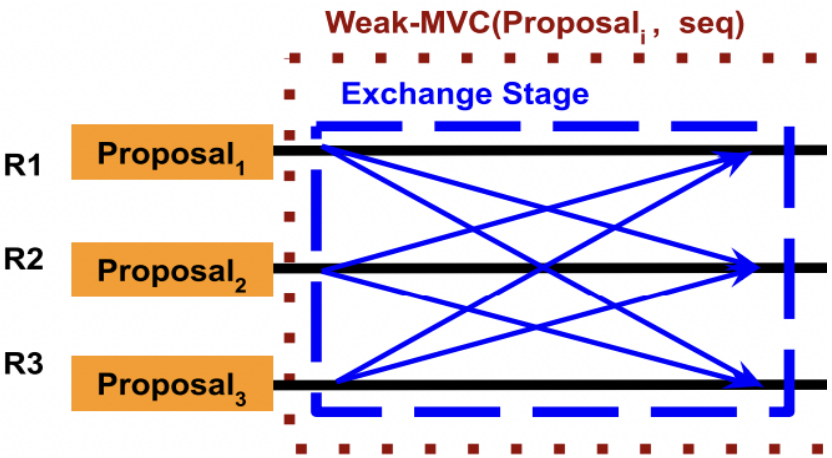


Weak-MVC(Proposal$_i$, seq)
Exchange Stage
Randomized Binary Consensus Stage
R1
R2
R3
Proposal$_1$
Proposal$_2$
Proposal$_3$
log[seq] ← output
log[seq] ← output
log[seq] ← output
If majority of p: state ← 1
o.w.: state ← 0
If majority of v: vote ← 1
o.w.: vote ← ?
If at least f+1 v's: return FindReturnValue(v)
If at least one v: state ← v
o.w.: state ← CommonCoingFlip(p)

Weak MVC

Stage 1 – Exchange Stage

- The first stage takes one phase (one message delay.
- Replicas exchange their proposals (a client request) - Propose Message
- Update the *state* variable based on the received proposals.



Algorithm 2 Weak-MVC: Code for Replica i

When WEAK-MVC **is invoked with input** q **and** seq**:**

```
// Exchange Stage: exchange proposals
Send (PROPOSAL, q) to all                 ▹q is client request
wait until receiving ≥ n − f PROPOSAL messages
if request q appears ≥ ⌊n/2⌋ + 1 times in PROPOSALs then
    state ← 1
else
    state ← 0
// Randomized Binary Consensus Stage (Phase p ≥ 1)
p ← 1                                     ▹Start with Phase 1
```

Weak MVC

- Stage 2 – Randomized Binary Consensus Stage
- Round 1 – Replicas exchange states and decide to vote as v or '?'. - State Message
- Round 2 – Replicas exchange vote messages and decide to either continue to terminate or proceed to next phase
    1. if v appears majority of times – terminate
    2. if v appears atleast once – set state as v
    3. if '?' Appears in majority of replicas – COMMONCOINFLIP(p)

```
// Randomized Binary Consensus Stage (Phase p ≥ 1)
p ← 1                                              ▹Start with Phase 1
while true do
    /* Round 1 */
    Send (STATE, p, state) to all                  ▹state can be 0 or 1
    wait until receiving ≥ n − f phase-p STATE messages
    if value v appears ≥ ⌊n/2⌋ + 1 times in STATES then
        vote ← v
    else
        vote ← ?
    /* Round 2 */
    Send (VOTE, p, vote) to all                    ▹vote can be 0, 1 or ?
    wait until receiving ≥ n − f phase-p VOTE messages
    if a non-? value v appears ≥ f + 1 times in VOTES then
        Return FINDRETURNVALUE(v)                  ▹Termination
    else if a non-? value v appears at least once in VOTES then
        state ← v
    else
        state ← COMMONCOINFLIP(p)                  ▹p-th coin flip
    p ← p + 1                                      ▹Proceed to next phase
```

Helper Function

Algorithm 3 Weak-MVC: Helper Function

Procedure FINDRETURNVALUE(v)

1: **if** $v = 1$ **then**

2: Find value m that appears $\geq \lfloor \frac{n}{2} \rfloor + 1$ times in PROPOSAL messages received in Phase 0

3: **Return** m

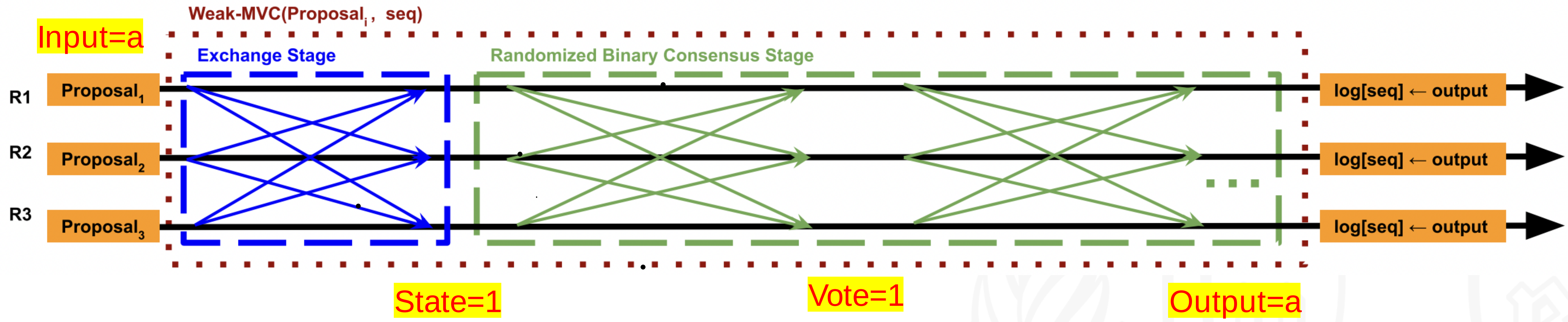
4: **else**

5: **Return** $\perp$ ▹`return null value`

Common Coin Flip

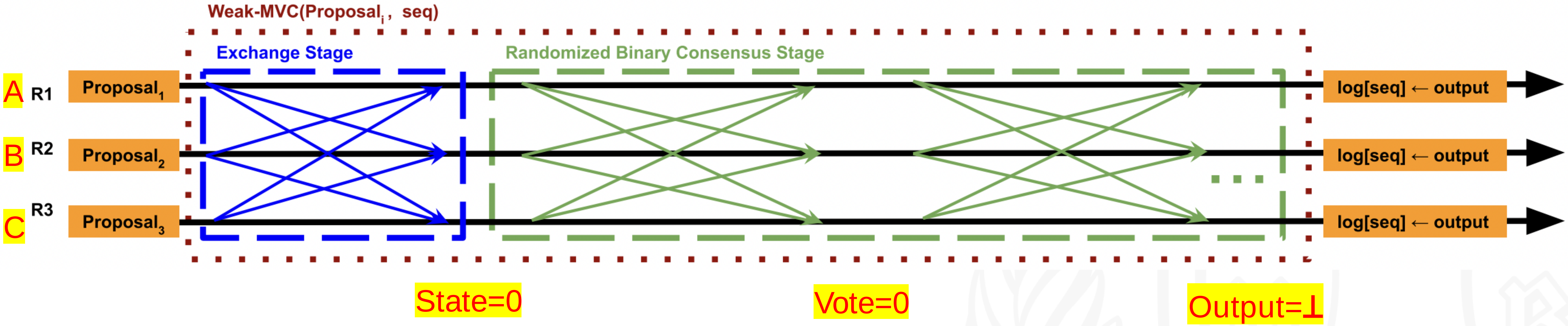
- Implement it by using a random binary number generator with the same seed across all replicas.
- Inspired by Ben Or's design.
- Gives the same value (0 or 1) to all replicas in the same phase, i.e., all replicas have the same p-th coin flip.
- Ben-Or protocol can repeat voting many times with some random coin flips in between the iterations to "jolt" the cluster into a decision.

Happy Day Scenario



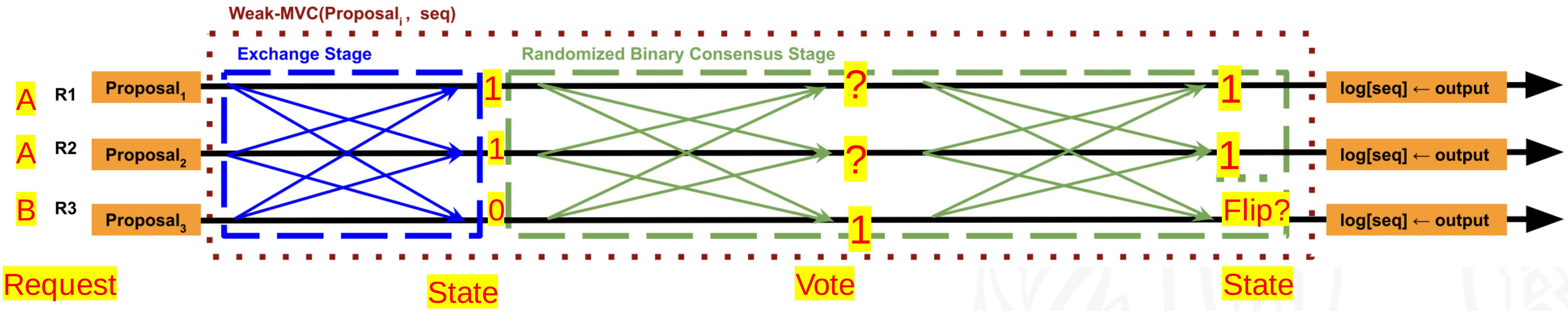
- The Weak-MVC terminates in three message delays.
- all replicas have $state = 1$ after the exchange stage.
- all replicas have $vote = 1$ after round 1 of phase 1.
- all replicas will execute Line 22 because all the vote messages contain 1.

Parallel Write Scenario



- The Weak-MVC terminates in three message delays.
- All replicas have $state = 0$ after the exchange stage.
- All replicas have $vote = 0$ after round 1 of phase 1.
- All replicas will execute Line 22 because all the vote messages contain 0.
- Meanwhile the request will be forwarded to all the replicas and next iteration will pick up single value.

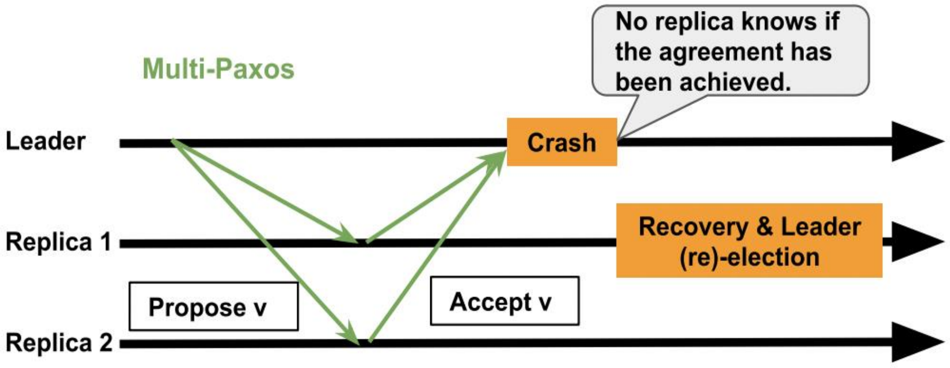
Scenario with Several Phases



- There will be multiple phases
- When replicas flip a common coin, it has probability 1/2 to flip to v and will terminate in the next phase.
- This sequence of events might repeat for a few phases if the coin flip is different from v.

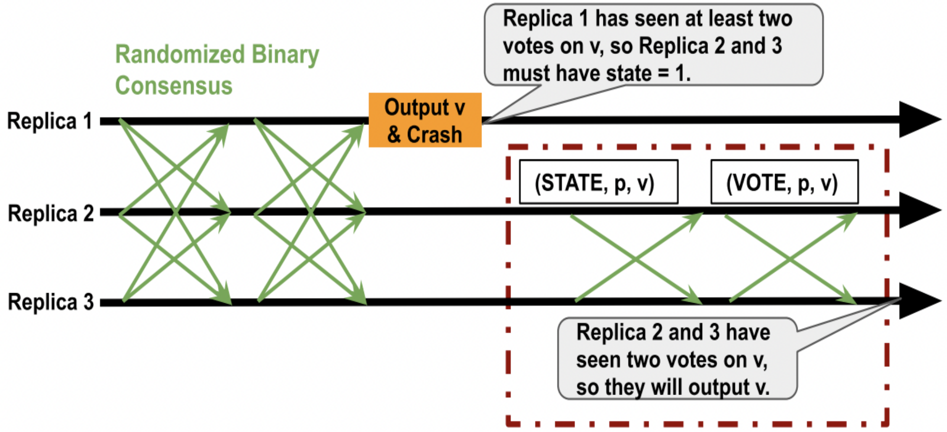
Fail-over protocol in other systems

- When a leader crashes, replicas need to execute an instance of leader election to ensure liveness.
- Replicas need to recover the decision(s) that the previous leader has already made.
- This task is non-trivial since replicas may have a stale log and may fail again during leader election.
- Fail-over becomes even more complicated, because some of the decisions may not have been persisted to a quorum of replicas yet.
- An example scenario of a crashed leader in Multi-Paxos is depicted in Figure



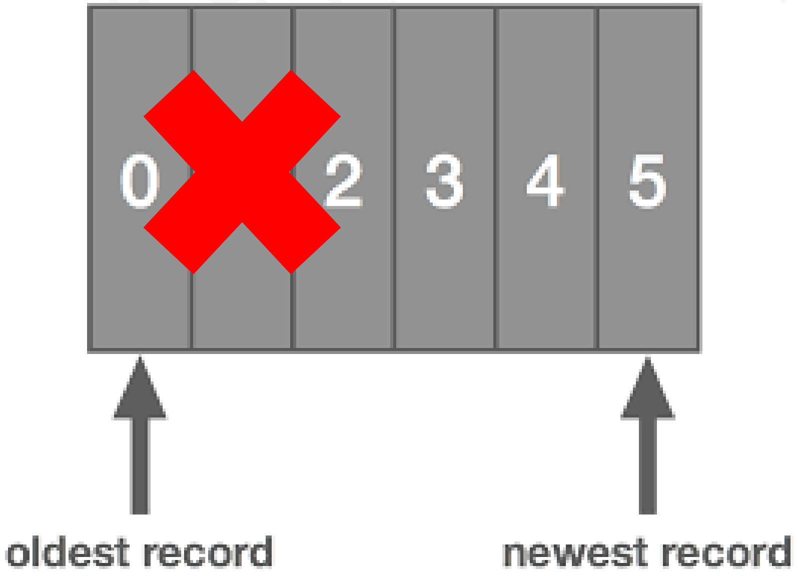
No Fail-Over Protocol in Rabia

- In Rabia, a group of replicas makes a “joint decision.”
- Replica 1 observes two vote messages of v and executes. Then, it fails shortly after it learns the output.
- By construction, after the exchange of phase-2 State messages, both Replica 2 and 3 update $vote$ to v; hence, they will output v in Phase 2.
- Note that these steps are already specified in Algorithm 2, and Weak-MVC does not need an auxiliary protocol to handle failures.



Simple Log Compaction in Rabia

- Rabia does not need a fail-over and does not rely on the notion of leader; hence, it supports a simple mechanism for log compaction
- Each replica has the same responsibility in Rabia, so a slow replica can learn from any other replica to catch up with missing slots.



Tail Latency Reduction

- Using a freeze time before participating in Weak-MVC, which allows the oldest pending request to be delivered at replicas.
- Using an eventually correct failure detector to allow replicas to receive a consistent set of messages.
- Bit complexity is dominated by client request size.

Verification

- Weak-MVC - safety properties are agreement and weak validity.
- The algorithm is verified in Ivy and Coq.
- Ivy is automated tool to check inductive invariants.
- Coq is general purpose theorem prover - used to prove correctness by induction.

Evaluation

- Rabia vs Multi-Paxos vs E Paxos(0 conflict)
- SMR Application – distributed key-value store
- Google Cloud platform
- Server Replicas - 32 GB Ram
- Client Machine – 128 GB Ram
- Deployed in single availability zone ---> Stable network
- RTT speed ≈ 0.25ms
- Size of client request is 16B

Replicas can decide on any order

Performance without Batching

	Rabia	EPaxos(NP)	EPaxos	Paxos(NP)	Paxos
Thpt	2458.56	2561.3	11480.1	1209.26	12993.07
M-Lat.	1.35	3.99	0.46	2.74	0.67

Table 1. Performance without Batching. (NP) indicates that a system has no pipelining. Throughput is represented as req/s, and median latency is measured in ms.

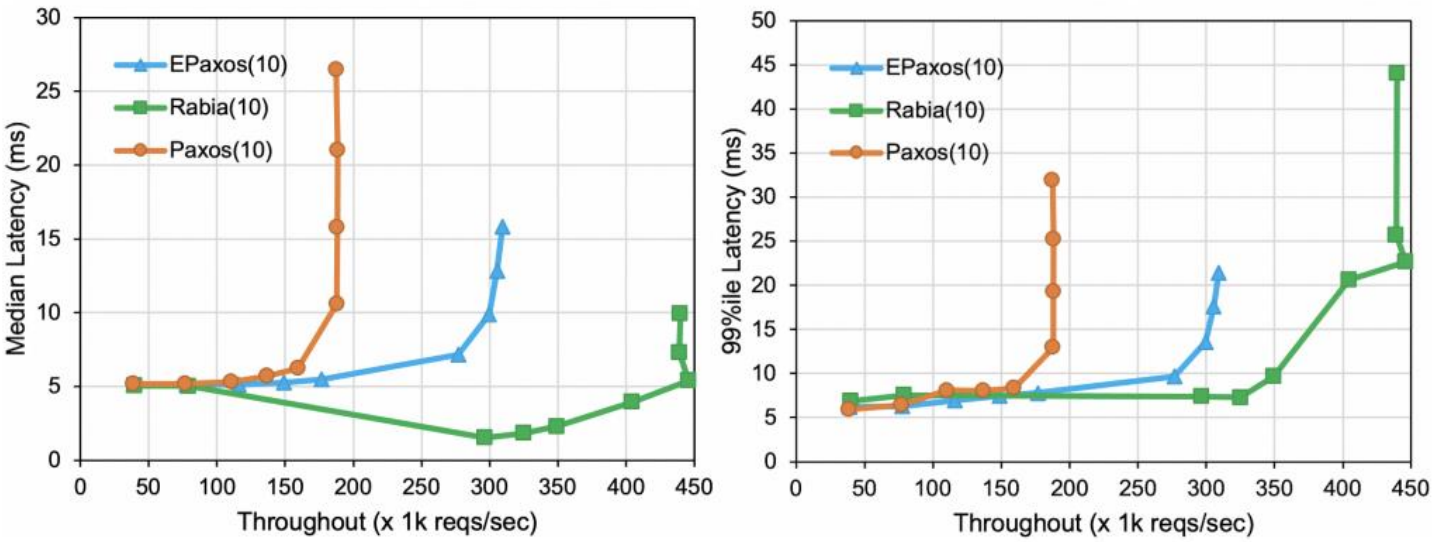
Evaluation

- Setup – 3 nodes
- Client Batch size = 10
- RTT = 0.25ms
- Request size = 16B
- Without Pipelining
- Maximum batch size

  Multi Paxos = 1000

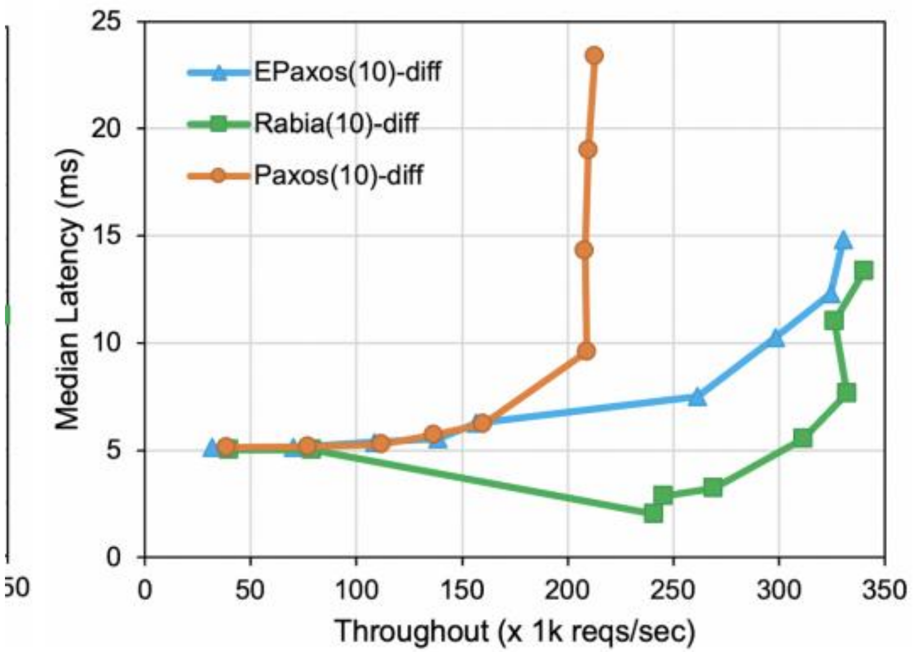
  E Paxos = 5000

  Rabia = 300

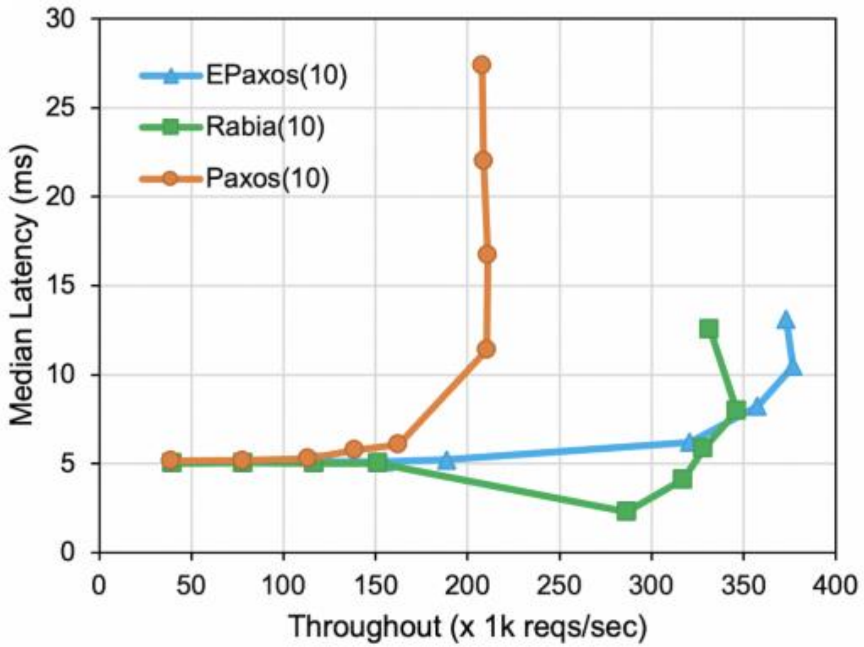


(a) Median Latency. Same Zone with 3 Replicas.

(b) 99th Percentile Latency. Same Zone with 3 Replicas.



(c) Median latency.
Multi-Zones with 3 Replicas.



(d) Median latency.
Same Zone with 5 Replicas.

Integration with Redis

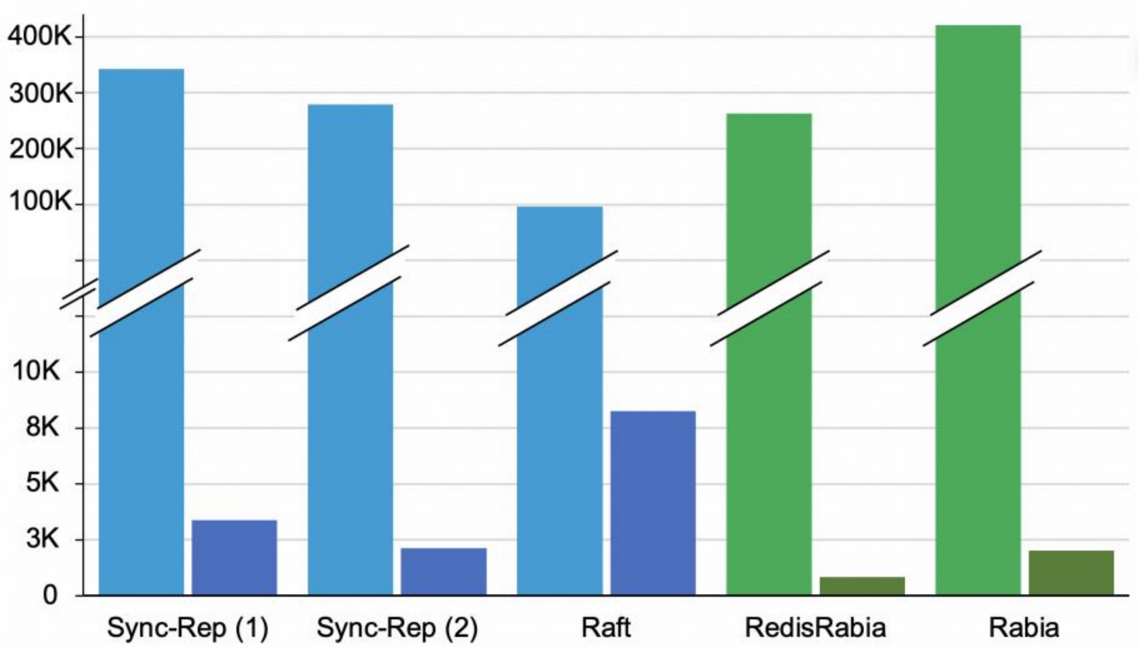


Figure 5. Throughput across different Redis integration. RedisRabia has around 1K req/s without batching, and Rabia has around 2K req/s without batching.

Conclusion

- Rabia does not need fail over protocol.
- It has simple log compaction.
- Key insight = stable network + Randomization.
- Achieves high performance in favorable settings (e.g., same availability zone with $n = 3$) and comparable performance in more general cases within a single datacenter

My Thoughts

- All systems have their own bottlenecks and Rabia as $O(N^2)$ message complexity.
- Node recovery is not discussed.
- Lack of testing under less favorable conditions.

References

- http://charap.co/reading-group-rabia-simplifying-state-machine-replication-through-randomization/
- https://arxiv.org/abs/2109.12616
- http://www.cs.yale.edu/homes/aspnes/pinewiki/RandomizedConsensus.html
- http://muratbuffalo.blogspot.com/2019/12/the-ben-or-decentralized-consensus.html