Replication

Feiran Chen

Outline

- Overview
- State Machine Tutorial
- Chain Replication

Keywords: Replication, fault tolerance

Outline

- Overview
- State Machine Tutorial
- Chain Replication

Keywords: Replication, fault tolerance

Outline

- Overview
- State Machine Tutorial
- Chain Replication

Keywords: Replication, fault tolerance

Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial

- Fred Schneider

Why State Machine?

- A general method for implementing a fault-tolerant service by replicating servers and coordinating client interactions with server replicas.

- Using a single, centralized, server is the simplest way to implement a service, the resulting service can only be as fault tolerant as the processor executing that server. If this level of fault tolerance is unacceptable, then multiple servers that fail independently must be used.

Why a Tutorial?

The “State Machine Approach” was introduced by Leslie Lamport in “Time, Clocks and Ordering of Events in Distributed Systems.”

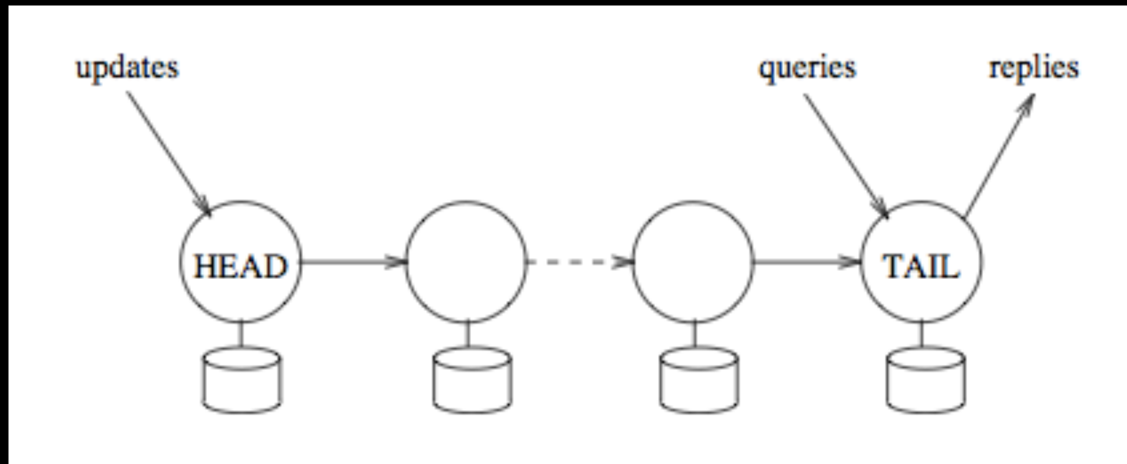
Lamport says on his page, about that paper:

> “This is my most often cited paper. Many computer scientists claim to have read it. But I have rarely encountered anyone who was aware that the paper said anything about state machines. People seem to think that it is about either the causality relation on events in a distributed system, or the distributed mutual exclusion problem. People have insisted that there is nothing about state machines in the paper. I've even had to go back and reread it to convince myself that I really did remember what I had written.”

Chain Replication For Supporting High Throughput and Availability

- Robert Van Renesse
- Fred Schneider

Chain Replication For Supporting High Throughput and Availability



Outline

- Overview
- State Machine Tutorial
- Chain Replication

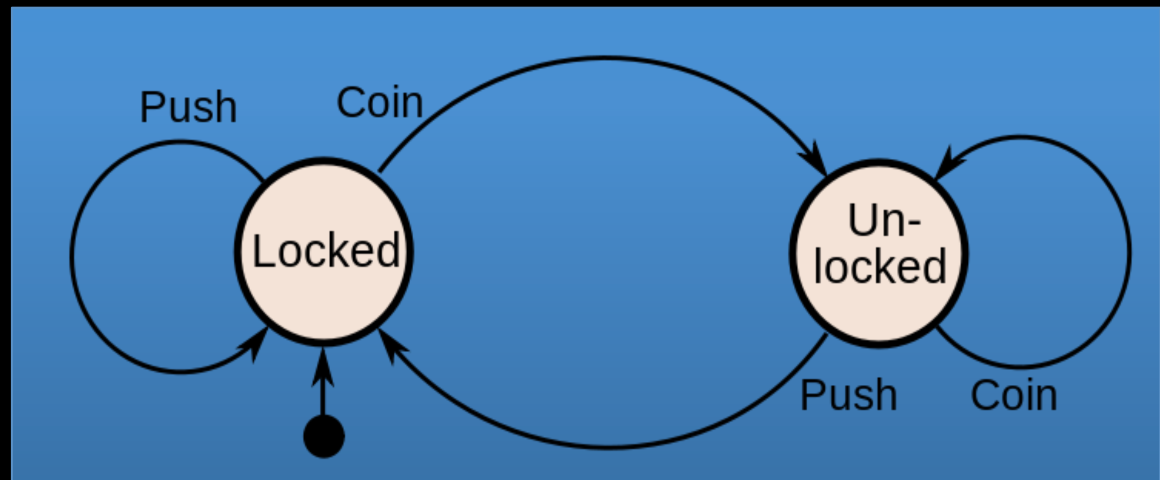
Keywords: Replication, fault tolerance

State Machine Tutorial Outline

- State machines
- Faults
- State Machine Replication
  - Agreement
  - Ordering
- Failures Outside the state machines
- Reconfiguring

State Machines

- State Variables
- Deterministic Commands



Requests and Causality, (Before Tutorial)

- Process order consistent with potentially causality.
- Guarantees:
  - Client A sends r, then r'.
  - r is processed before r'.
  - r causes Client B to send r'.
  - r is processed before r'.

State Machine Coding

- State Machines are procedures
- Client calls procedure
- Avoid loops.
- More flexible structure.

Consensus

- Termination
- Validity
- Integrity
- Agreement

- Ensures procedures are called in same order across all machines

State Machine Tutorial Outline

- State machines
- Faults
- State Machine Replication
  - Agreement
  - Ordering
- Failures Outside the state machines
- Reconfiguring

Faults: The Malicious and the Benign

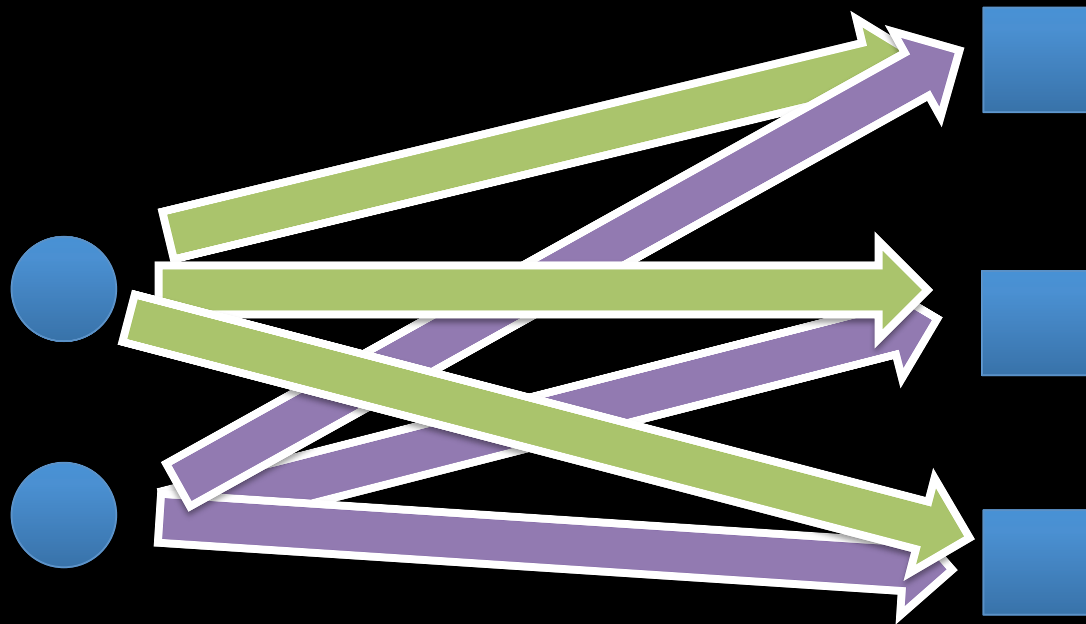
- Byzantine Faults:
  - Malicious/arbitrary behavior by faulty components.
  - Weakest possible failure assumption.
- Fail-Stop Faults:
  - Changes to fail state and stops.

Tolerating Faults

- "t fault tolerant"
  - ≤ t components become faulty
  - Simply where the guarantees end.
- Statistical Measures
  - Mean time between failures
  - Probability of failure over interval
  - Other
  - Not measuring degree of tolerance

State Machine Tutorial Outline

- State machines
- Faults
- State Machine Replication
  - Agreement
  - Ordering
- Failures Outside the state machines
- Reconfiguring



- **Agreement** governs the behavior of a client in interacting with state machine replicas
- **Order** governs the behavior of a state machine replica with respect to requests from various clients.

State Machine Tutorial Outline

- State machines
- Faults
- State Machine Replication
  - Agreement
  - Ordering
- Failures Outside the state machines
- Reconfiguring

Agreement

- "The transmitter" disseminates a value, then:
  - IC1: All non-faulty processors agree on the same value
  - IC2: If transmitter is non-faulty, agree on its value.
- Client can
  - be the transmitter
  - send request to one replica, who is transmitter

Outline

- State machines
- Faults
- State Machine Replication
  - Agreement
  - Ordering
- Failures Outside the state machines
- Reconfiguring

Ordering

- Unique identifier, uid on each request
- Total ordering on uid.
- Request, r is stable if
  - Cannot receive request with uid(r') < uid(r)
- Process a request once it is stable.
- Logical clocks can be the basis for unique id.
- Stability tests for logical clocks?
  - Byzantine faults?

Ordering

- Can use synchronized real-time clocks.
- Max one request at every tick.
- If clocks synchronized within δ,
  - Message delay > δ
- Stability tests?
- Potential Problems?
  - State Machine lag behind clients by Δ (test 1)
  - Never passed on crash failures (test 2)

More Ordering...

- Can the replicas generate uid's?
- Yes!
- Consensus is the key!
- State machines propose candidate id's.
- One of these selected, becomes unique id.

Constraints

- UID1: $cuid(sm_i, r) <= uid(r)$.
- UID2: If a request r' is seen by sm_i after r has been accepted by sm_i, then $uid(r') < cuid(sm_i, r')$.

How to generate uid's?

- Requirements:
  - UID1 and UID2 be satisfied
  - r != r' → *uid(r) != uid(r')*
  - Every request seen is eventually accepted.
- Define:
  - SEEN(i) = largest *cuid(*sm_i*,r) assigned to any request so far seen at* sm_i
  - *ACCEPT(i) =* largest *cuid(*sm_i*,r) assigned to any request so far accepted by* sm_i

Generating uid's....

- $cuid(sm_i, r) = max(SEEN(i), ACCEPT(i)) + 1 + i/N$.
- $uid(r) = max(cuid(sm_i, r))$
- Stability test?
- Potential Problems?
  - Could affect causality of requests
  - Client does not communicate until request is accepted.
- More or less communication needed?

Outline

- State machines
- Faults
- State Machine Replication
- Failures Outside the state machines
- Reconfiguring

Tolerating failures

- Failed output device or voter:
  - Replicate?
  - Use physical properties to tolerate failures, like the flaps example in the paper.
  - Add enough redundancy in fail-stop systems
- Client Failure:
  - Who cares?
  - If sharing processor, use that SM

Outline

- State machines
- Faults
- State Machine Replication
- Failures Outside the state machines
- Reconfiguring

Reconfiguration

- Would removing failed systems help us tolerate more faults?
- Yes, it seems!
- P(t) = total processor at time t
- F(t) = Failed Processors at time t
- Assume Combine function, P(t) – F(t) > Enuf
- Enuf = P(t)/2 for byzantine failures
- Enuf = 0 for fail-stop.

Reconfiguration

- F1: If Byzantine failures, then faulty machines are removed from the system before combining function is violated.
- F2: In any case, repaired processors are added before combining function is violated.
- Might actually improve system performance.
- Fewer messages, faster consensus.

Takeaways

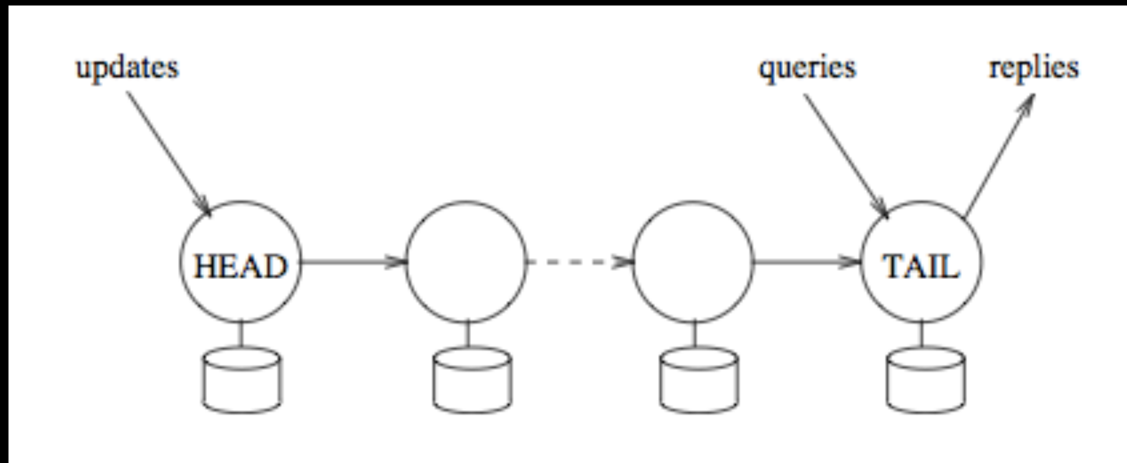
- The State Machine approach is flexible.
- Replication with consensus, given deterministic machines, provides fault tolerance.
- Depending on assumptions, may need more replications, may use different strategies.

Outline

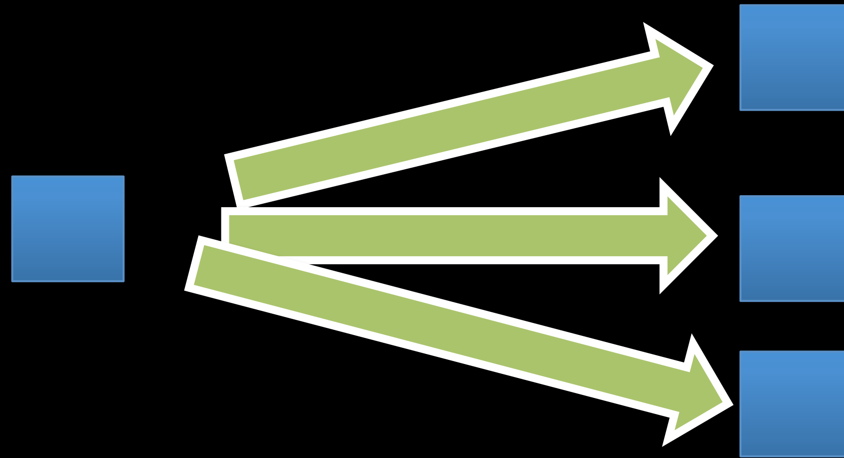
- Overview
- State Machine Tutorial
- Chain Replication

Keywords: Replication, fault tolerance

Chain Replication For Supporting High Throughput and Availability



Primary-Backup



- Serial version of State Machine Replication
- Only the primary does the processing
- Updates sent to the backups.

Chain Replication Assumes:

- No partition tolerance.
- Chain replication: Consistency, availability.
- A partitioned server == failed server.
- High Throughput.
- Fail-stop processors.
- A universally accessible, failure resistant or replicated Master, which can detect failures.

Reads and Writes

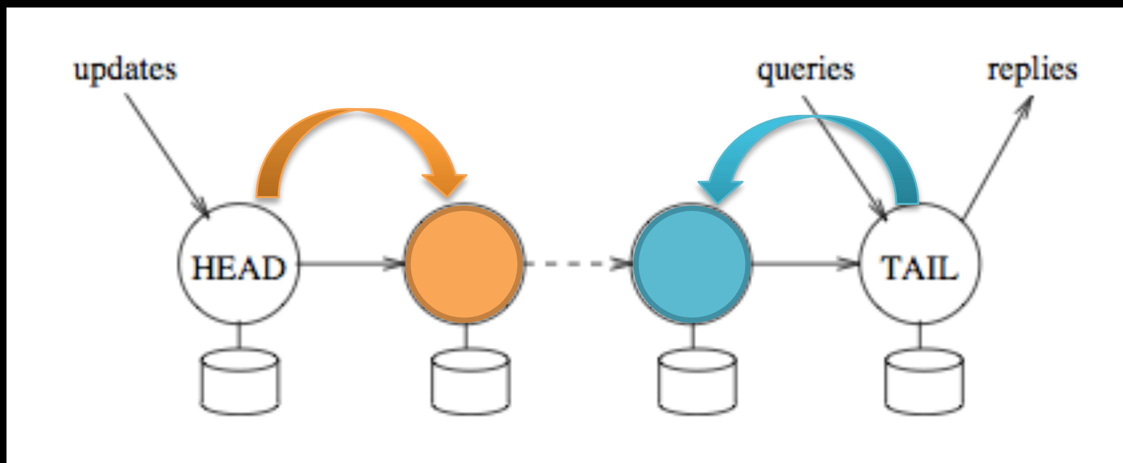
- Reads go to any non-faulty tail.
  - Just tail, 1 server per chain
- Writes propagate through all non-faulty servers.
  - t-1 severs per chain

God’s eye: Master

- Failures are detected by God/Master.
- On detecting failure, Master:
  - informs its predecessor or successor in the chain
  - informs each node its new neighbors
- Clients ask the master for information regarding the head and the tail.

Handling failures

- Assumed to never fail or replicated w/ Paxos
- Head fails?
- Tail fails?
- Other fails?
  - Remove, inform predecessor, handle ACKs



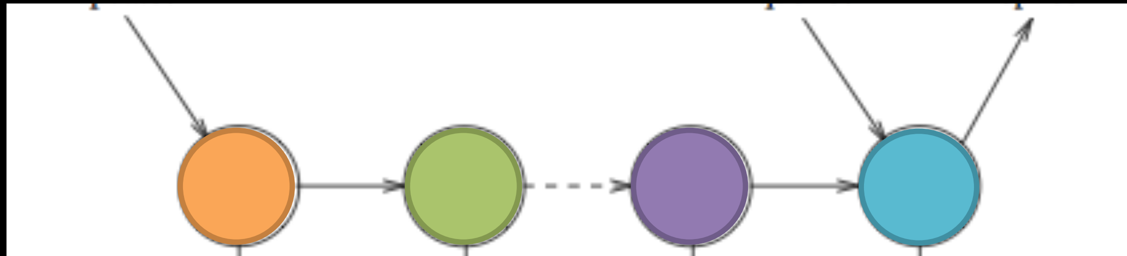
Adding a new replica

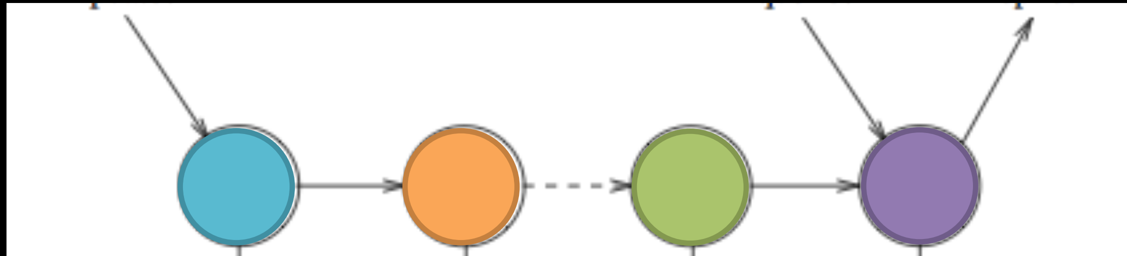
- Current tail, T notified it is no longer the tail.
- State, Un-ACK-ed requests now transmitted to the new tail.
- Master notified of the new tail.
- Clients notified of new tail.

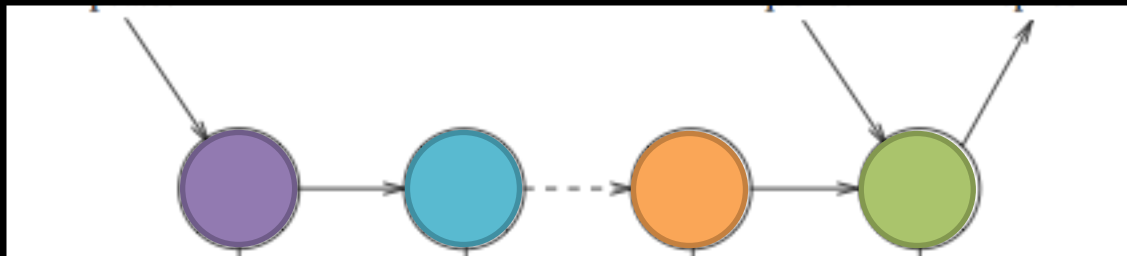
Unavailability

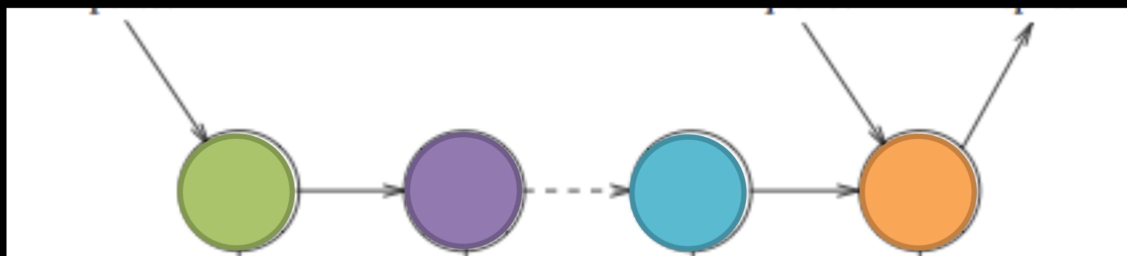
- Head failure:
  - Query processing uninterrupted,
  - update processing unavailable till new head takes on responsibility.
- Middle failure:
  - Query processing uninterrupted,
  - update processing might be delayed.
- Tail failure:
  - Query and update processing unavailable, until new tail takes over.

You Said Scaling?









Conclusions

- State Machine Replication probably would have less service interruption, but at a high cost, and usually more complex to implement.
- Primary-Backup easier to implement, reasonably high throughput and no need of consensus, so might be more efficient even in terms of network usage, but at the cost of lower availability.