SpecPaxos
James Connolly & Harrison Davis
Overview

● **Background**
  ○ Fast Paxos
  ○ Traditional Paxos Implementations
  ○ Data Centers

● **Mostly-Ordered-Multicast**
  ○ Network layer

● **Speculative Paxos Protocol**
  ○ Application layer
Recall Fast Paxos

Client

Proposer

ANY

m

R0

R1

R2

R3

Learner
Are we done?
Only two message delays?
Recall Fast Paxos

THREE MESSAGE DELAYS??

Client

Proposer

ANY

m

R0

R1

R2

R3

Learner
Typical Paxos Implementations

- Designed independently from the underlying network
- Assumptions about the network are worst-case
  - Asynchronous
  - Packets are arbitrarily delayed, dropped, or reordered
- Useful for the Internet
  - Little is known about the underlying network
  - Would you trust a logo like this?
- NO
- We have access to something better: Data Centers
Data Centers

● Attributes
  ○ Predictable
    ■ Known network topologies
  ○ Reliable
    ■ Congestion losses unlikely with Quality of Service (QoS)
  ○ Extensible
    ■ Software defined networks (SDNs) allows for novel routing protocols

● Goal
  ○ Design a multicast protocol that aims for ordered and synchronous message delivery
Motivation

What would Fast Paxos look like with ordered multicast on a synchronous network?

That's it. Not always achievable… but is in the common case.
Traditional Multicast

- Not Ordered
- Different path lengths between nodes
- Different levels of congestion along different paths
Mostly Ordered Multicast (MoM)

- **Step 1:** ensure all messages travel the same length
- All messages routed through root switches
  - S1 or S2
  - SDN routes MoM traffic here
- One point of serialization
- Load balance using different roots
- If S1 fails, route traffic through S2
  - SDN routes according to failures
Mostly Ordered Multicast (MoM)

- **Step 2:** prioritize client to node messages

- Quality of Service (QoS) to prioritize MoM traffic
- QoS uses strict-priority hardware queues
  - Ensures MoM messages are sent before other types of traffic

⭐ Actual quality 99.9%
MoM Evaluation

Test Setup
SpecPaxos: Speculative Operation

Low latency AND High Throughput
SpecPaxos

- Aka faster FastPaxos
  - Clients broadcast requests
  - However, now all replicas ONLY respond to client
    - Reduces leader bottleneck

- Fault Tolerance
  - Same as Paxos: up to f benign failures
  - Still 2f+1 replicas
SpecPaxos: Subprotocols

- **Speculative Processing:** *normal operation*
  - less than f/2 failures
  - replicas speculatively commit

- **Synchronization:** *occasional, leader initiates*
  - To verify we speculate correctly

- **Reconciliation:** *divergence*
  - When synchronization fails
  - Or between f/2 and f failures.
SpecPaxos: Replica State

- Each Replica has its own
  - State
    - Normal, Reconciliation, Recovery, Reconfiguration
  - View number
    - Designates the current leader \((v \mod n)\)
  - Log
SpecPaxos: Replica Log

- Sequence of executed requests
  - Each request within the log contains
    - Sequence number
    - State
      - Speculative (Triangle)
      - Committed (Circle)
    - Unique Hash of Previous Log
We must wait for $f + f/2 + 1$ responses that match.
SpecPaxos: Synchronization

Leader

SYNC(v,s)

R0

SYNCREPLY(v,s,h(s))

R1

COMMIT(v,s,h(s))

R2

R3

- v := view number
- s := highest seqno
- h(s) := hash correlated with seqno s

We must wait for $f + f/2 + 1$ responses that match
Replica Log of $R_n$

$\text{COMMT}(v,3,h(3))$
SpecPaxos: Reconciliation

- Transfer into this state when
  - A client receives less than $f + \frac{f}{2} + 1$ matching specreplies()
  - Inconsistent state detected during synchronization by leader or replica
  - Timeouts
    - Synchronization
    - Specreply to client

- Analogous to collision recovery…
  - Also can be used as a ‘classic round’ to allow progress when more than $f/2$ replicas fail
wait for f

StartRecon(v,s)

Reconcile(v,v_L log)

v_L := highest view number of when a replica was in normal operation

R_n

wait for f
Log Merging -- $F + 1$

- Consider log of size 3
  - $F$ failed replicas
  - Clients don’t require super quorum
    - May recover into a different state
    - Client already accepted this, now our system is inconsistent
    - Precisely the same issue with Fast Paxos

Client’s Perspective
Log Merging -- Super Quorum

Client’s Perspective
Log Merging

- Well described within the paper.
  - Log merging takes longer for more speculative states
  - If a speculative action is not in the same sequence within a majority of the received views, it is appended to the new log arbitrarily
Log Merging -- Reconciliation Pt.2

Client(s)

\[ R_n \]

\[ R_0 \]

\[ R_1 \]

\[ R_2 \]

\[ R_3 \]

Start-View(v,log)

Commit(v,s,h(s))

In-View(v,log)

Return to Speculative Execution
SpecPaxos: Progress with f failures

- Log merging allows non-super quorum actions to be appended arbitrarily
  - The commit message sent after the In-view message allows these messages to be committed
  - If we finish reconciliation, and a new process immediately executes a new request
    - F failures still exist
    - Immediately go back into reconciliation
    - Slow!
SpecPaxos: Evaluation

- $f = 1$ (3 replicas)
- SP: MoM
- FP: MoM
- P: IP multicast
SpecPaxos: Evaluation

- Message reordering was artificially increased
  - SpecPaxos must perform reconciliation
  - Performance drops
- SpecPaxos still outperforms Paxos until 0.1% reorderings
Conclusion

- Data Center networks are predictable, reliable, and extensible
  - Use this to design MoM
- MoM achieves greater than 99.9% of ordered messages
- SpecPaxos uses MoM to speculatively commit requests
  - Higher throughput: ~100,000 req/s vs ~38,000 req/s
  - Lower latency: 135 µs vs 171 µs
Pros and Cons

Pros:

- Lower latency due to fewer messages delays 2 vs (3 or 4)
- Higher throughput due to reduced bottle

Cons:

- Slightly higher latency for some messages
  - Due to routing through root switches
- Prioritizing MoM adds latency to other messages
  - If everyone has priority no one has priority
- Evaluation with only \( f = 1 \)
- Reconciliation is very expensive
Open Questions?

- How well does this scale?
  - If many systems are using MoM in a Data Center, what happens to ordering?
- Any others?

- Can we do better?
  - NO… Paxos
Thank you!
1. Increment $v$, send Start-Reconciliation($v$) to all replicas
2. After receiving $f$ Start-Reconciliation($v$) messages, send Reconcile($v$, $v_L$, $log$) to the new leader
3. New leader receives $f$ Reconcile($v$, $v_L$, $log$) messages, and begins the log merging process
1. New leader sends `Start-View(v, newlog)` to all replicas
2. Each replica sends `In-View(v)` back to leader
3. After receiving `f In-View(v)` messages, leader sends Commit messages to all replicas, and to any relevant clients