SpecPaxos
Jacob Thoma
SpecPaxos

- **Paxos**: 4 message delays
  - Client → Leader → Replicas → Leader → Client
- **FastPaxos**: 3 message delays
  - Client → Replicas → Leader → Client
- **SpecPaxos**: 2 message delays!
  - Client → Replicas → Client

- **How?**
  - Push ordering responsibility to the network
    - Data center networks are more predictable, reliable, and extensible
  - Speculatively execute client requests
  - Beats Paxos/FastPaxos in latency and throughput when reordering is rare
Fast Paxos is 3 message delays? I thought Lamport said it was 2!

**FastPaxos:**
- SpecPaxos Authors:
  - Client ➔ Replicas ➔ Leader ➔ Client (3)
- Lamport:
  - Proposer ➔ Acceptors ➔ Learners (2)

**SpecPaxos:**
- SpecPaxos Authors:
  - Client ➔ Replicas ➔ Client (2)
Mostly-Ordered Multicast

• Topology-aware multicast
• High-priority multicast
• In-network serialization
Existing Datacenter Multicast

C1 multicasts m to N1, N2, N3
Existing Datacenter Multicast

```
t=0
```

- C1
- S3
- N1
- S1
- S2
- S4
- S5
- N2
- C2
- N3
Existing Datacenter Multicast

- $S_1$ to $S_2$: $<m, N2>$, $<m, N3>$
- $S_3$ to $S_1$: $<m, N1>$
- $S_4$: $N2$
- $S_5$: $C_2$, $N3$

$t=1$
Existing Datacenter Multicast

S1

S3
  C1
  N1
  \langle m, N1 \rangle
  rcvd at t=2

S1

S4
  N2

S2

S5
  C2
  N3

\langle m, N2 \rangle
\langle m, N3 \rangle

\text{t=2}
Existing Datacenter Multicast

\[ <m, N1> \] rcvd at \( t=2 \)

\[ <m, N2> \]

\[ <m, N3> \]
Existing Datacenter Multicast
Topology-aware Multicast

- Route multicast messages through switches that are equidistant from all nodes in the target multicast group
- Can cause increased latency for some recipients of each multicast message
  - Max latency and average latency not significantly impacted
- Each multicast group has a single address
  - Target switches convert message to a true multicast packet and send it downward along replica-facing ports
C1 wants to multicast m to N1, N2, N3
Can choose to route to S1 or S2
Topology-aware Multicast

t=0

<m, S2>
Topology-aware Multicast

t=1
Topology-aware Multicast

C1 N1 N2 N3 C2

S1 <m, N1> <m, N2> <m, N3>

S2

S3

S4

S5

C1 N1 N2 N3

t=2
Topology-aware Multicast
Topology-aware Multicast

T=4

C1

S3

S1

S4

S2

S5

C2

N1

N2

N3

<m, N1> rcvd at t=4

<m, N2> rcvd at t=4

<m, N3> rcvd at t=4
High-Priority Multicast

• Topology-aware Multicast is impacted by cross traffic
• Assign a higher priority to MOM traffic using Quality of Service mechanisms
  • Mitigates, but does not eliminate reordering
Topography-Aware Multicast

Different paths may have different queueing delays due to background traffic.

$t = 0.0$: C1 multicasts $m_1$ through S1

$t = 0.1$: C2 multicasts $m_2$ through S2
Topology-Aware Multicast

S3

C1 N1

S1

S4

<\textit{m1} > rcvd

N2

S2

C2 N3

S5

<\textit{m2} > rcvd
Topology-Aware Multicast

N2 and N3 receive messages in different orders
High-Priority Multicast

Skip to the front of the queue using QoS priority, mitigating cross-traffic impact.

$t = 0.0$: C1 multicasts $m1$ through S1
$t = 0.1$: C2 multicasts $m2$ through S2
With high probability, N2 and N3 receive messages in the same order.
In-network serialization

• Guarantee correct ordering of messages
• Route all multicast operations to a given group through the same switch
  • Acts as a serialization point
  • Ensures that messages to a given group node traverse same path
In-network Serialization

Anyone wants to multicast to N1, N2, N3:
Can ONLY route through S2
MOM Evaluation
MOM Evaluation

Packet reordering frequency

Multicast sending rate (packets/sec)

- Multicast
- MOMs
- MOMs + QoS
- MOMs + NS

0%
0.01%
0.1%
1%
10%

20000
40000
60000
80000
100000
SpecPaxos

• **Speculative processing**
  • Executes requests efficiently in normal case where messages are ordered and < f/2 replicas have failed

• **Synchronization**
  • Periodically verifies that replicas have speculatively executed the same requests in the same order

• **Reconciliation**
  • Ensures progress when requests are delivered out of order or when between f/2 and f nodes have failed
Replica State

Each replica maintains:

- **Status**
  - NORMAL, RECONCILIATION, RECOVERY

- **Log**
  - Sequence of operations executed by replica
  - Each entry tagged with:
    - Sequence number
    - COMMITED/SPECULATIVE tag
    - Summary hash, $\text{summary}_n = H(\text{summary}_{n-1} \ | \ | \ \text{operation}_n)$

- **View number**
  - The leader is only used to coordinate synchronization and reconciliation, **not** normal-case speculative processing
Speculative Processing

- Client sends a **REQUEST** to all replicas
  - Includes operation requested, identity of client, unique client identifier
  - MOM primitives imply that replicas are likely to receive concurrent requests in same order
- Replicas receive **REQUEST**
  - If in **NORMAL** state:
    - Immediately speculatively execute
    - Append request to log with **SPECULATIVE** tag
    - Send **SPECULATIVE-REPLY** to client
- Client waits for matching **SPECULATIVE-REPLY**’s from a superquorum of $f + \lceil f/2 \rceil + 1$ replicas
Speculative Processing

Client -> <REQUEST> -> Replica 1 -> Replica 2 -> Replica 3 -> Replica 4 -> Replica 5
Speculative Processing

Client

<REQUEST>

Replica 1: specexec()

Replica 2: specexec()

Replica 3: specexec()

Replica 4: specexec()

Replica 5: specexec()
Speculative Processing

Client

<REQUEST>

<SPECULATIVE-REPLY>

match?

Replica 1

specexec()

Replica 2

specexec()

Replica 3

specexec()

Replica 4

specexec()

Replica 5

specexec()
Why do we need a superquorum of responses?

- Correct replicas can still receive operations from different clients in inconsistent orders
Synchronization

- Replicas still don’t know the outcome of their committed operations
- Periodically the leader sends <SYNC, v, s> message to other replicas
  - v: current view number
  - s: highest sequence number in its log
- Replicas reply with <SYNC-REPLY, v, s, h(s)>
  - h(s): summary hash(entry s in its log)
- When leader has received f + [f/2] + 1 matching SYNC-REPLY’s for sequence number s
  - sends <COMMIT, v, s, h(s)> to other replicas
Synchronization

Replica 1 (leader)
Replica 2
Replica 3
Replica 4
Replica 5

<SYNC, v, s>
Synchronization

Replica 1 (leader)
Replica 2
Replica 3
Replica 4
Replica 5

<SYNC, v, s>

<SYNC-REPLY, v, s, h(s)>
Synchronization

Replica 1 (leader)
Replica 2
Replica 3
Replica 4
Replica 5

\(<\text{SYNC, } v, s>\>

\(<\text{SYNC-REPLY, } v, s, h(s)>\>

f + \lceil f/2 \rceil + 1

\text{matching}

h(s): \(<\text{COMMIT, } v, s, h(s)>\>

\text{tag entries in log up to } s \text{ with COMMIT}
Reconciliation

• Replica state diverges
  • If replicas fail
  • If leader receives **SYNC-REPLY** messages that *do not match*
  • If replica receives a **COMMIT** message with a hash different than its current log entry

• Initiate a reconciliation
  • Increment \(v\)

• Sends a <**START-RECONCILIATION**, \(v\)> message to the other replicas
  • This causes other replicas to also initiate reconciliation

• Once a replica has received **START-RECONCILIATION** for view \(v\) from \(f\) other replicas it sends <**RECONCILE**, \(v\), \(v_i\), log> to leader of new view
Reconciliation

increment \( v \),
\(<\text{START-RECON, } v>\)
Reconciliation

Increment \( v \), \(<\text{START-RECON}, v>\)

Replica 1 (leader)
Replica 2
Replica 3
Replica 4
Replica 5
Reconciliation

Replica 1 (old leader)

Replica 2 (new leader)

Replica 3

Replica 4

Replica 5

increment \( v \), \(<\text{START-RECON}, v>\)

increment \( v \), \(<\text{START-RECON}, v>\)

\(<\text{RECONCILE}, v, v_l, \log>\)
Reconciliation

• Once new leader receives **RECONCILE** from f other replicas, it merges their logs
• Merges all logs with the highest \( v_i \) and produces combined log with properties:
  • If same prefix appears in a majority of logs, then those entries appear in the combined log in the same position (for safety)
  • Every operation in any of the input logs appears in the output log (for liveness)
• Sends a **<START-VIEW, v, log>** message to all replicas
• Replicas receive **START-VIEW**
  • roll back any speculative operations that do not match new log
  • execute any new operations
Reconciliation

Replica 1 (old leader)
increment \( v \), \(<\text{START-RECON, } v>\)

Replica 2 (new leader)
increment \( v \), \(<\text{START-RECON, } v>\)
increment \( v \), \(<\text{RECONCILE, } v, v_i, \text{log}>\)

Replica 3

Replica 4

Replica 5

merge logs
Reconciliation

increment $v$, <START-RECON, $v$>
increment $v$, <START-RECON, $v$>
<RECONCILE, $v$, $v'_l$, log>
<START-VIEW, $v$, log>

Replica 1
(old leader)

Replica 2
(new leader)

Replica 3

Replica 4

Replica 5

merge logs

install new log
Correctness

• Any successful operation always survives in the same serial order
• Only one operation can succeed for a given sequence number
SpecPaxos Evaluation
SpecPaxos Evaluation
Not so good

• If \( \lceil f/2 \rceil + 1 \) replicas have failed, SpecPaxos must perform reconciliation every time

• Requires reordering rate < 0.1%
  • (Only applicable in datacenters with MOM mechanism applied)
Summary

• **SpecPaxos**: 2 message delays!
  • Client → Replicas → Client

• Push ordering responsibility to the network using MOM mechanisms
  • Data center networks are more predictable, reliable, and extensible

• Speculatively execute client requests

• 20% lower latency than FastPaxos and 60% lower latency than Paxos under low-medium request rates

• Beats Paxos/FastPaxos in throughput when **reordering rate is < 0.1%**
Question?