NOPAXOS: REPLACING CONSENSUS WITH NETWORK ORDERING

Authors: J. Li, E. Michael, N. K. Sharma, A. Szekeres, D. R. K. Ports
Presenter: Wei Lee
MOTIVATION
PAXOS IN A SLIDE

Proposers, acceptors and learners are all collocated on $2f + 1$ replicas

Expensive Coordination!

Always stays safe, achieves liveness only when system behaves synchronously
Paxos Assumption

- Communication Model: completely asynchronous and unordered communication

- Failure Model: crash failures $\Rightarrow 2f+1$ replicas

But Data Center Networks are more predictable and more reliable
To ensure Linearizability

Two properties needed:

- **Ordering**: if some replica processes request A before B, no other process B before A

- **Reliable Delivery**: Every request submitted by a client is either processed by all or none

Can we move some of the responsibilities to network layers?
PAXOS

Ordering
Reliable delivery

CONSENSUS

Application
Transport
Network
Data Link
Physical
What about... 

Atomic Broadcast
(as hard as consensus)
Reliable delivery
Ordering

Application
Transport
Network
Data Link
Physical
OR WHAT ABOUT…

Reliable delivery

- Application
- Transport
- Network
- Data Link
- Physical

Ordering
**Relaxed Assumption**

- **Communication Model**: completely asynchronous and **ordered** communication

- **Failure Model**: crash failures $\Rightarrow 2f+1$ replicas

Powered by **Ordered Unreliable Multicast**
**Key Difference**

- Speculative Paxos (MOM)
  - Mostly ordered
- Paxos (OUM)
  - Always ordered

<table>
<thead>
<tr>
<th></th>
<th>Speculative Paxos [53]</th>
<th>NOPaxos</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network ordering</strong></td>
<td>Best-effort</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Messages at bottleneck</strong></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Quorum size</strong></td>
<td>&gt; 3n/4</td>
<td>&gt; n/2</td>
</tr>
<tr>
<td><strong>Reordering/Dropped packet penalty</strong></td>
<td>high</td>
<td>low</td>
</tr>
</tbody>
</table>
ORDERED UNRELIABLE MULTICAST (OUM)
Ordered Unreliable Multicast

Properties:

- Asynchrony
- Unreliability
- **Ordered Multicast**: if multicast m and m’ to processes, all receive in the same order
**Ordered Unreliable Multicast**

Properties:

- **Multicast Drop Detection:**
  
  if multicast m to processes, either:
  
  - Every process receives m or drop notification, or
  
  - No process receives anything
Use **Sequence number** to ensure ordering

- **OUM session**: intervals in which OUM guarantees to hold

- **OUM group**: a set of receivers identified by an IP address

Share sequencer, session, seq_num
OUM IMPLEMENTATION

Must serialize request through sequencers for each OUM group, so:

- **Software-defined networking (SDN)** needed
  - **controller**: install forwarding rules, reconfigure
  - **sequencer**: switch, middlebox or end-host

*Figure 1: Architecture of NOPaxos.*
PROTOCOL
OVERVIEW

• Messages ordered but can be dropped
  • New problem: binary consensus
    • Replicas agree on which request to execute, which to ignore
  • Run consensus only when Drop_Notification received

• 2 roles and 4 subprotocols
  • leader and normal replica

• View based
  • view_id = <leader_num, session_num>
4 SubProtocols

- Normal Operations
  - Process client requests
- Gap Agreement
  - Drop-Notification received
- View Change
  - Leader fails or session terminates
- Synchronization
  - Leader syncs log with replicas (optimization)
NORMAL OPERATION

- Client multicasts to replicas

![Diagram showing a client multicasting to replicas](image-url)
NORMAL OPERATION

- Receiving replicas save operation to log and reply

Must receive f+1 REPLY (including leader)
NORMAL OPERATION

- If fails, client retries

Must receive $f+1$ REPLYS (including leader)

- <REPLY, view_id, log_slot_num, req_id, result>

Give me gum gum!!!

Give him gum gum
Gap Agreement

- If replica get Drop_notification, ask leader

Client

leader

Give me dum dum!

replica

replica

DROP!!
Gap Agreement

- If replica get Drop_notification, ask leader

What happened last time?

We gave gum gum
Gap Agreement

- If leader get Drop_notification, everyone replace with NO OP

Roger that

leader

DROP!!

No, I didn’t give.

Roger that

replica

<GAP_COMMIT, log_slot>

replica

<GAP_COMMIT_REP, log_slot>
View Change

- A replica initiate View Change either if
  - Suspects leader is dead
  - Detects end of an OUM session
  - Receives VIEW_CHANGE_REQ or VIEW_CHANGE from other with higher leader_num or session_num
**VIEW CHANGE**

- `view_id=<leader_num, session_num>`
  - Old leader
  - `<VIEW_CHG_REQ, view_id>`

- Replica
  - `He's dead`
  - `<VIEW_CHG, view_id, old_view_id, log>`

- Replica
  - `My new leader, here's my log!`

- Replica
  - `He's dead`
View Change

- Once collects $f+1$ VIEW_CHG (including self), becomes new leader, execute extra requests on log

client

old leader

replica

leader

Aye!

I'm your new leader, follow my order!

<START_VIEW, view_id, log>

Merge most Recent logs

execute
**Synchronization**

- View Change can be expensive! (make-up works)
  - Leader sync its log with replicas
  - Replicas execute new operations locally
  - Everyone updates `sync_point`
EVALUATION
Latency vs Throughput

- Paxos
- Fast Paxos
- Batching
- SpecPaxos
- NOPaxos
- Unreplicated
ROBUST AGAINST UNRELIABILITY

The diagram illustrates the throughput performance of different consensus algorithms under varying simulated drop rates. The x-axis represents the simulated drop rate, ranging from 0.001% to 1%. The y-axis shows throughput in operations per second (ops/sec), ranging from 0K to 350K.

- **Paxos**: Shows a stable performance across different drop rates.
- **Fast Paxos**: Similar to Paxos, maintaining consistent throughput.
- **Batching**: Exhibits a slight decrease in throughput with increasing drop rates.
- **SpecPaxos**: Maintains a relatively stable throughput.
- **NOPaxos**: Displays a significant decrease in throughput as the drop rate increases.
- **Unreplicated**: Shows the lowest throughput, significantly dropping as the drop rate increases.

The graph highlights the robustness of consensus algorithms against unreliability, with Paxos and Fast Paxos proving to be the most resilient.
View Change can be expensive! (make-up works)
ROBUST AGAINST SEQUENCER FAILURE
CONCLUSION
• Strength

  • Identifies features of data center network

  • **Further** leverages network capability to reduce overhead for coordination

  • Efficient, scalable and robust most of the time

• Weakness

  • Lacks evaluation on controller failure and synchronization

  • Only apply to 1 data center

• Future work

  • Apply OUM on **Byzantine failure**
QUESTIONS?