Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering

NOPaxos

Li, Michael, Sharma, Szekeres, Ports

Presentation by: Steven Schulte and Samantha Silveira
Motivation

- Paxos makes transactions slow
- Every request has large overhead
- Can we improve latency and throughput?

YES!
NOPaxos: Network Ordered Paxos

- **Key insight:**
  - Divide responsibility
    - **Network:** guarantee message ordering (OUM)
    - **Application:** handle dropped packets

- **Result:**
  - Normal case avoids coordination entirely
  - Easier than ordering requests in application
Ordered Unreliable Multicast

- Network guarantees messages received in order
- Route client requests through sequencer
- Sequencer adds monotonically increasing sequence number
- Sequence number establishes order of messages
- Sequence number gaps imply dropped messages
An absolute temporal ordering is not what you want in a distributed system anyway.

Leslie Lamport
OUM - Sequencer

1. On data center switches
2. On a middlebox
3. On a dedicated server
OUM - Practical

- Members of group identified by a distinct address
- For a particular OUM group
  - All packages routed through a sequencer
  - Routing done by network controller
  - 88% cases serialization added no extra latency
  - 99th percentile only 5µs extra latency
NOPaxos - Overview

- **Model:**
  - Asynchronous network
  - Up to $f$ crash failures
  - $2f + 1$ replicas
  - Still safe if more than $f$ failures

- Requires ordered but unreliable message delivery

- Provides linearizability of client requests
Failed non-leader processes

- Easily handle $f$ non-leader replicas crashing
- Client will still receive $f + 1$ responses
Dropped packets

- Detect by gap in sequence number
- Algorithm receives DROP-NOTIFICATION
- Non-leaders ask leader what request was
- Leader gets agreement to drop that request
  - Insert noop in log
  - Send gap-commit to other replicas
- Optimization: leader asks replicas before inserting noop
Gap-commit - leader drops packet

- Leader
  - Leader inserts noop in log
  - Sends <GAP-COMMIT, log-slot> to all replicas
  - Waits for f <GAP-COMMIT-REP, log-slot>
  - Retries if necessary

- Replicas
  - Insert noop at log-slot, possibly overwriting

- Ensures noop decision persists if leader fails
View change

- A view:
  - Same leader
  - Same session number

- View change: Leader failures or session termination
  - Ensures progress
  - Ensures replicas start new view consistently
  - Successful operations from last view retained
View Change: Initiation

- **Initiation:**
  - Replica detects leader failure OR
  - Replica receives a VIEW-CHANGE request OR
  - Replica receives a SESSION-TERMINATED message

- Set status to ‘ViewChange’

- Replica broadcasts VIEW-CHANGE-REQ to other replicas

- Replica sends VIEW-CHANGE notification to new leader
VIEW-CHANGE

VIEW-CHANGE-REQ
View Change: New Leader

- New leader receives VIEW-CHANGE messages
  - View id of last 'good' view
  - Log of that view
  - ID of new proposed view
- New leader waits for f+1 matching messages
- Unifies logs
  - Noop if any log had a noop
  - Request if any log had a req
- Sends out START-VIEW
View Change: Start View

- Replicas send reply to client to any new requests on merged log
- New leader must execute all requests on log
- Replicas ignore any START-VIEW with a lower view-id than the current

Problem?
Synchronization

- A required 'optimization'
- Non leader's logs may contain speculative requests
  - Slots may be changed to noops
- Periodic synchronization in background
  - Ensures replicas have stable logs up to sync point
  - Replicas can then execute commands up to sync point in background
Forked State

What if a correct leader is incorrectly replaced?

Requires support for transferring application state
What about the sequencer?

Could it fail?

Yes, but we can handle it just like leader failures
Evaluation

- Compared to:
  - Paxos
  - Fast Paxos
  - Paxos with Batching
  - Speculative Paxos
  - Unreplicated system

- 3-level fat-tree network
- 5 replicas
Evaluation

Figure 5: Latency vs. throughput comparison for testbed deployment of NOPaxos and other protocols.
Evaluation

Figure 7: Maximum throughput with simulated packet dropping.

Figure 8: Maximum throughput with increasing number of replicas.
Strengths

- Common-case as costly as without replication
- Safe in any number of failures
Strengths and weaknesses

● Designed for single data center
● Sending logs is slow
● Requests build up without synchronization
● Requires way to transfer application state
● Security concern
Thanks!

Questions?