MoveCast: Atomic Multicast Protocol with Object Migration

RQ: How to integrate **object migration** within an **atomic multicast protocol**

Author: Riad Zaid | r.zaid@student.tudelft.nl
Supervisor: Jérémie Decouchant

1. Background

1.1 Atomic Multicast

Multicast, is a mechanism for **groups of processes** in a distributed system to communicate with each other.

Unlike broadcast, where a message is sent to all processes, multicast enables the communication to a **subset of processes**.

Atomic Multicast refers to a multicast mechanism where **total order** is preserved between the groups of a destination.

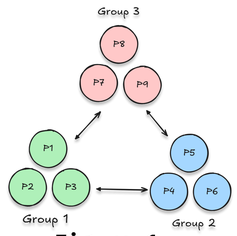

Figure 1

1.2 White-Box Protocol

- **Core Innovation:** Merges Paxos consensus directly into Skeen's algorithm to accelerate atomic multicast
- **Optimized Routing:** Group leaders intentionally bypass other leaders, sending ACCEPT messages directly to a quorum of followers across all groups
- **Global Ordering:** Once followers acknowledge (ACCEPT_ACK), leaders commit the message using a global timestamp : max(local_timestamps)
- **Performance Impact:** This direct-to-follower approach drastically reduces both failure-free and collision-free latency while maintaining strict safety.

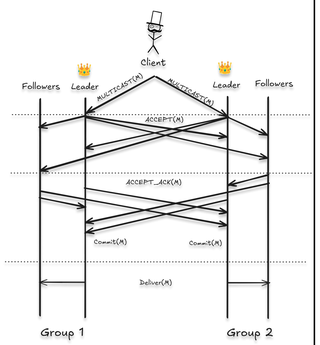

Figure 2

2. Problem Statement

Due to objects being spread out between shards, there are **often cross-shard transactions.**

The goal would be to **minimize** these cross-shard transaction to increase performance.

The **Shard Scheduler** paper has introduced an algorithm to decide when objects should be migrated in order **to distribute load and maximize performance.**

This research has integrated the Shard Scheduler ideas within the White-Box atomic multicast protocol.

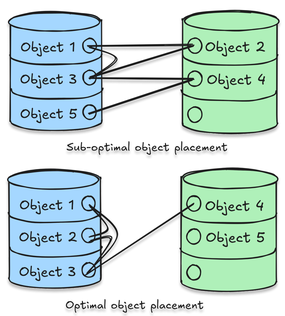

Figure 3

3. The Protocol : MoveCast

3.1 High-level overview of MoveCast

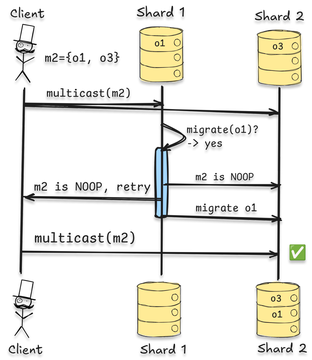

Figure 4

1. **Freeze:** A shard decides an object needs to migrate and freezes it.
2. **NOOP Intercept:** All new messages targeting the frozen object are safely marked as "NOOP" (no operation).
3. **Clear Queue:** The system waits until all in-flight, conflicting messages are fully delivered.
4. **Migrate:** The object is securely moved to its new destination shard.
5. **Retry:** The client is notified of the NOOP and can instantly retry the transaction at the optimal shard.

3.2 Messages overview of MoveCast

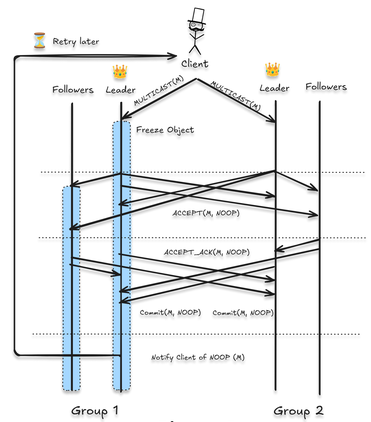

Figure 5

1. **Client Multicast:** The client initiates a transaction (M) that requires coordination between multiple groups.
2. **Piggybacked NOOP:** The leader of Group 1 decides to freeze the object and embeds a "NOOP" flag directly into the standard White-Box ACCEPT message.
3. **State Propagation:** Followers and other group leaders receive the message and acknowledge the NOOP status via ACCEPT_ACK.
4. **Commit & Notify:** The transaction is committed across the network as a NOOP, and the client is directly signaled to retry the transaction later.

4. Methodology

We benchmark White-Box against Movecast using an in-memory **Golang** simulator.

- Each process is its own thread.
- **8 groups** of 3 processes each.
- Intra-group latency = **1ms**
- Inter-group latency = **50ms**
- Client latency = **0ms**
- Adjusted **Sui** blockchain dataset with **600.000 transactions**
- **50.000 objects** are initially **randomly distributed**
- **We do not model crashes in the simulator.**

5. Results

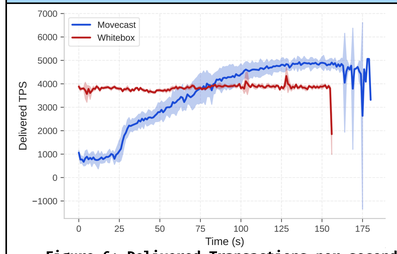

Figure 6: Delivered Transactions per second

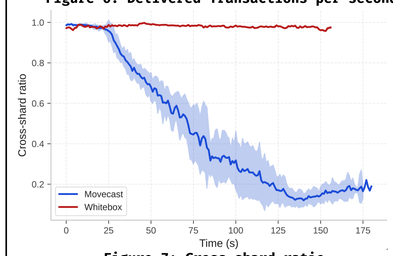

Figure 7: Cross-shard ratio

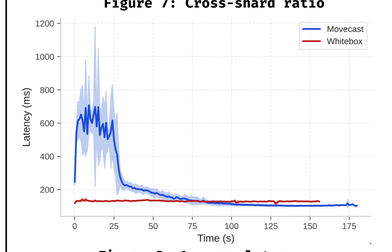

Figure 8: Average latency

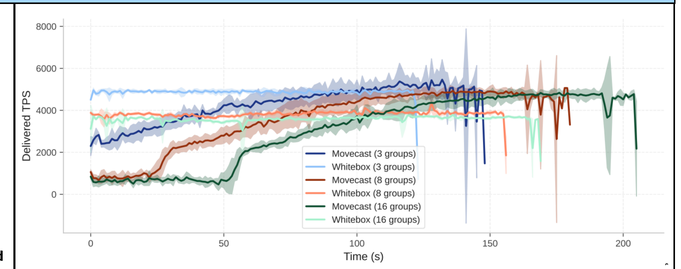

Figure 9: TPS comparison with different number of groups (50k objects)

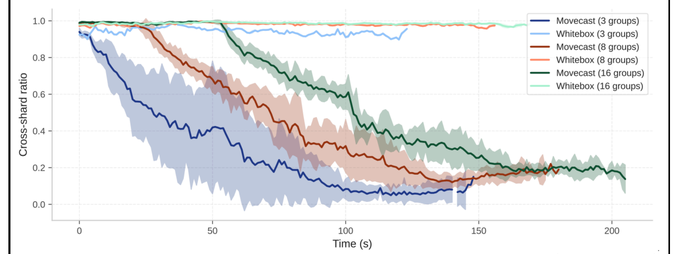

Figure 10: Cross-shard ratio with different number of groups (50k objects)

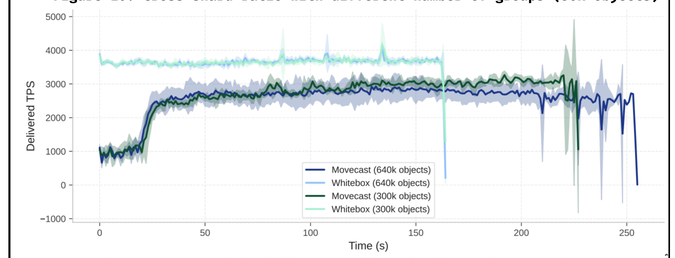

Figure 11: TPS with different number of objects (8 groups)

Transactions/second : 3800 → 4850
Average Latency : 130ms → 103ms
Cross-shard tx ratio: 0.98 → 0.2

6. Future Work

- Instead of a **stop-and-restart migration** approach, **an asynchronous migration** mechanism would decrease the freeze window. This would allow for large objects to migrate in the background, while the system keeps running.
- **Take advantage of the object set to deliver non-conflicting messages** in the delivery queue. If there are two messages with no overlap in object sets; it might be safe to deliver them without ordering them.