1 Introduction

In this project we implement a **Sharded Key-Value Store** that supports shard reconfiguration and transactions over multiple shards. Shards are replicated across multiple servers in a "Shard-Group" which use Paxos to reach consensus regarding the order of operations on the shards so that linearizablity of operations is maintained.

1.1 Terms and Definitions

Here are some terms that we will be using throughout our implementation.

- **Config Controller**: The testing framework uses the Config-Controller to create new configurations in which different shards belong to different groups.
- **Shard Master**: The ShardMaster receives data from the Config-Controller and tries to balance all the shards across the available Shard Groups. It keeps track of all the configurations and is Queried by ShardStore Servers and Clients to obtain new configurations.
- **Shard Group**: A replica group which contains multiple shards, each having it's own Key-Value Store.
- **Transaction**: An operation across multiple shards which maybe present across multiple Shard-Groups. These are implement using a 2-Phase Commit protocol. This implementation supports two types of Transactions - *MultiGets* and *MultiPuts*.

2 Flow of Control and Code Design

2.1 Reconfigurations

There are 3 types of Reconfigurations possible - **Joins, Leaves** and **Moves**. All these three follow the same flow of control shown in Fig. 1.

1. The ConfigController sends the new configuration to the ShardMaster.
2. The ShardMaster divides the shards among the shard groups *(If needed)* and creates the new configuration, i.e, a mapping between ShardGroups and Shards.
3. Servers from both Shard Groups query the ShardMaster for the configuration succeding their corresponding Current Configurations.
4. Once a new configuration is received, it is validated by the server and sent through Paxos, so that all the servers in the Group can process the reply in the same sequence.
5. In our example, Group 1 needs to transfer shards to Group 2 and instantiates a ShardMove.
6. A server in Group 2 validates the ShardMove, pushes it into Paxos and once processed, all servers copy the KV Stores into the corresponding Shards.
7. A server from Group 2 sends a ShardMoveAck to Group 1.

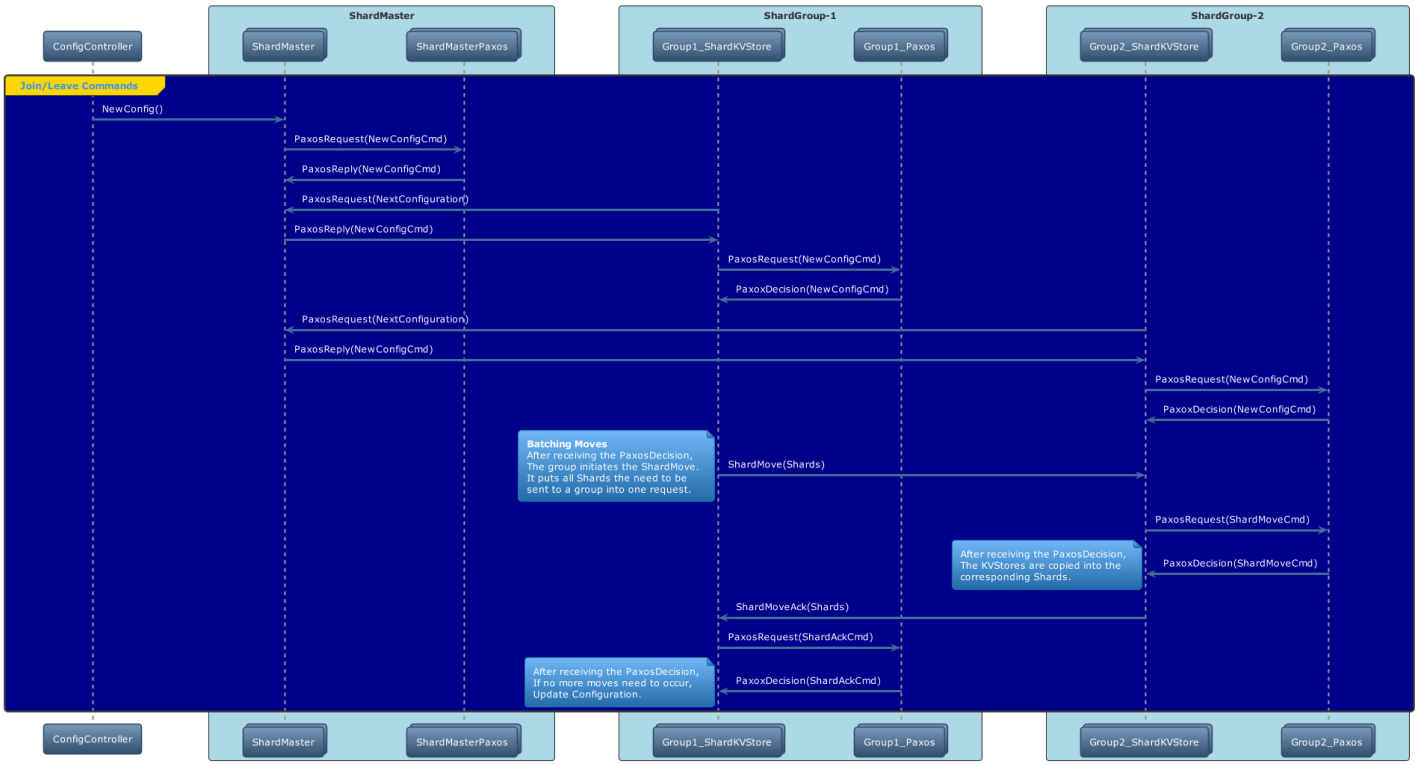

Figure 1: Reconfigurations across 2 Shard-Groups

8. The ShardMoveAck is validated, pushed into Paxos and once a decision is received, all the servers update their configurations. *(If no more Acks or Shards need to be received by the Group)*

While ShardMoves are being performed, if a single key operation, operating on a shard that is being sent, is received, it is ignored. If a transaction is received, it is also ignored.

2.2 Single Shard Operations

Single Shard Operations are fairly simple to handle as only one Shard and therfore one group is involved in the operation. Therefore, consensus does not need to be reached across multiple groups regard which operation needs to be performed. The flow of control is shown in Fig. 2.

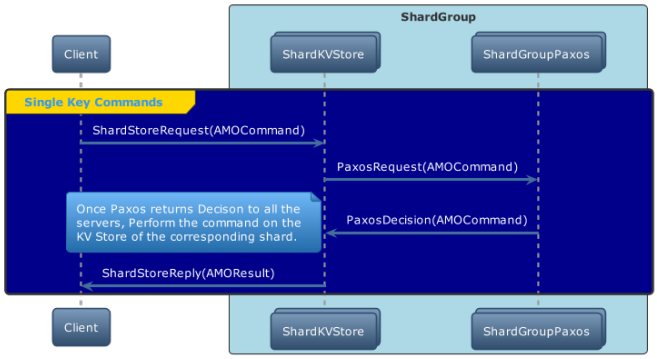

Figure 2: Single Shard Operations

2.3 Cross Shard Transactions

Our implementation allows each Shard-Group to perform at most one transaction at a time. As shown in Fig. 3, there may be cases when one of Shard-Groups required to service a Transaction is unavailable and in such cases, to avoid deadlocks, we need to abort the current transaction and release all locks that have been acquired for this transaction. The abort is then propagated to the Client which retries the request with a higher *Sequence Number*.

A Cross-Group Transaction is performed using a 2 Phase Commit. A coordinator is chosen by the Client for a given Transaction in a deterministic manner. The Coordinator, on receiving the request and processing it, locks the required shards and sends a prepare request to all the required groups, asking them to lock their shards as well. Once a shard is locked, no operation can be performed on the shard. The groups then lock the shards and send a PrepareAck to the coordinator. The coordinator then sends a CommitRequest, asking the Groups to perform the required operations on their shards. Once performed, the groups unlock the shards and send a CommitAck to the coordinator. The coordinator then performs the required operations on its shards, gathers all the results and send it result of the transaction to the client. The flow of control is shown in Fig. 3.

2.4 Timers and Failures

We use 7 timers in this implementation - *ConfigRefreshTimer, BufferTimer, ShardMoveTimer, PrepareTimer, CommitTimer, AbortTimer and ClientTimer*

- **ClientTimer**: This timer is used to resend client requests if the client has not received a response for awhile. It helps with dealing with dropped client requests. The client also queries the ShardMaster for the latest configuration when this timer fires.
- **ConfigRefreshTimer**: This timer is used by the ShardServers to obtain new configurations from the ShardMaster periodically.
- **BufferTimer**: This timer is used to execute buffered ShardMove requests, if the request was received before the new configuration was processed by a ShardServer.
- **ShardMoveTimer**: This timer is used to resend Shard Moves that may have been dropped while the system is proceeding to a new configuration. A balance between this timer and the BufferTimer was important as ShardMove Requests are very bulky and having the buffer timer ensures that once received, they are not rejected because of a configuration mismatch. This indirectly allows the ShardMoveTimer to retry with a lower frequency to reduce network congestion.
- **PrepareTimer**: This timer is used to resend dropped Prepare Requests.
- **CommitTimer**: This timer is used to resend dropped Commit Requests.
- **AbortTimer**: This timer is used to resend dropped Abort Requests.

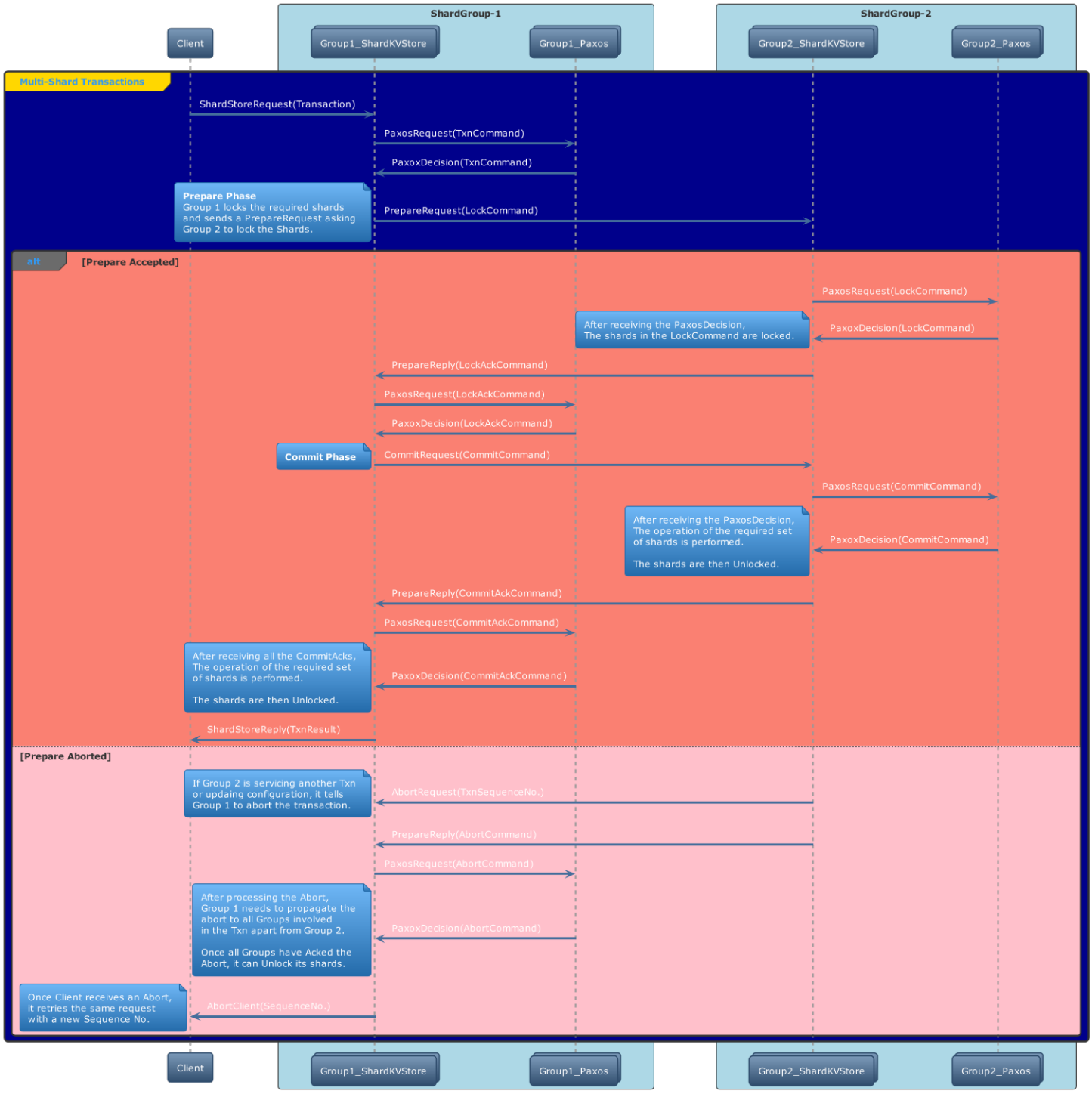

Figure 3: Multi Shard Operations

3 Design Decisions

- **Batching ShardMoves and ShardMoveAcks**
  Instead of sending multiple ShardMoves and ShardAcks to the same group, Each group sends only one ShardMove Message to each of the Groups to which it needs to send some shards. Similarly, instead of sending multiple ShardAcks to the same group, they are also batched. This helps because there are lesser messages exchanged to perform reconfigurations.

- **Buffering ShardMoves**
  Once a ShardMove is received, there is a chance that the Group has not yet processed the next configuration from the ShardMaster. In such a situation, the ShardMove is buffered so that once the new config is processed, it can be applied. This affords us to have a higher retry timeout for the ShardMoves being sent from other groups and helps to decongest the network.

- **Handling Aborts**
  Aborts are handled by asking the Client to retry the same command with a higher sequence number. The means that we don't have to keep track of which requests were aborted on the server side and that helps in reducing the complexity of the server-side logic.