- Project 1: Report
    - Intro (Your understanding)
    - Flow of Control & Code Design
        - 1. The AMO Shim Layer
        - 2. Client Flow
        - 3. Server Flow
    - Design Decisions
    - Missing Components
    - References
    - Extra (Optional)

Project 1: Report

Intro (Your understanding)

This lab implements a distributed Key-Value store with "Exactly-Once" delivery semantics. The system is composed of three main parts:

1. **Application (`KVStore`):** A simple in-memory storage engine supporting `Get`, `Put`, and `Append` operations.
2. **AMO Application Shim (`AMOApplication`):** A wrapper around the `KVStore` that filters duplicate requests. It tracks the sequence number of the last executed command for each client address to ensure that commands are executed "At-Most-Once".
3. **Client (`SimpleClient`):** A network client that handles "At-Least-Once" reliability. It wraps commands in an `AMOCommand` (attaching a sequence number and client address) and retries them until a matching `AMOResult` is received.

Flow of Control & Code Design

1. The AMO Shim Layer

The core of the "Exactly-Once" guarantee lies in `AMOApplication.java`.

- **State:** It maintains a `Map<Address, AMOResult>` called `results`. This stores the result of the *most recently executed* command for each client.
- **Execution Logic:**
  1. When an `AMOCommand` arrives, the shim checks `alreadyExecuted()`.
  2. It compares the command's `sequenceNumber` against the stored result's `sequenceNumber` for that client address.
  3. **If new:** It passes the inner command to the underlying `KVStore`, wraps the `Result` in an `AMOResult`, stores it in the map, and returns it.
  4. **If duplicate:** It simply returns the cached `AMOResult` from the map without re-executing the command on the `KVStore`.

2. Client Flow

The client uses a "stop-and-wait" approach.

- **Sending:**
  - The client increments its local `sequenceNum`.
  - It creates an `AMOCommand` containing the `KVStore` command, the new sequence number, and its own `Address`.
  - It wraps this in a `Request` message and sends it.
  - It sets a `ClientTimer`.
- **Retrying:**
  - If the timer fires before a result is received, the client resends the *exact same* `Request` object. This preserves the sequence number, allowing the server's `AMOApplication` to identify it as a retry rather than a new command.

3. Server Flow

The server is simple because the complexity is delegated to the `AMOApplication`.

- **Initialization:** The server wraps the `KVStore` application inside an `AMOApplication` during construction.
- **Handling:** When a request arrives, the server passes the command to `app.execute()`. The `AMOApplication` handles the deduplication logic transparency. The server then replies with the `AMOResult`.

Design Decisions

- **Shim Layer Pattern:** "At-Most-Once" logic was implemented in a separate `AMOApplication` class rather than modifying the `KVStore` or `SimpleServer` directly. This separation of concerns allows the `KVStore` to remain purely focused on data storage and allows the deduplication logic to be reused for other applications.
- **Garbage Collection strategy:** The `AMOApplication` stores only the *latest* result per client address in a `HashMap`. Since the lab assumes clients have only one outstanding request at a time, overwriting the previous entry for a specific `Address` effectively garbage collects old metadata. This prevents memory from growing linearly with the number of requests per client.

Missing Components

Didn't have any missing components.

References

- **Lab 1 Handout:** Used to understand the requirements for `AMOCommand` structure and the specific interface for `AMOApplication`.
- **Lab 0 Handout**: Considered for understanding the architecture of the lab and various libraries that the code uses.
- **Java Map Interface:** Referenced documentation for various methods in `Map` interface in `Java`

Extra (Optional)