Name: Zijie Huang

Andrew ID: zijieh

1. **Protocol Between Server and Client**

   The communication protocol between the server and the client is designed around Remote Procedure Calls (RPCs), particularly for file operations like open and close. The initial contact begins with the client requesting to **open/close** a file, which triggers the proxy to call the RPC's **download/upload** method.

   a) **Download Method**: When a client requests a file, it first attempts to fetch the first chunk of the file from the server. The server, upon receiving a request, simulates the file opening scenario and returns appropriate error codes or opening modes. The first chunk is marked with its validity status. If valid, it's tagged with the opening mode; if not, it's tagged with an error code. Information on the file's existence, the size of the first chunk, and the total file size are also provided.

   b) **Upload Method**: When a client uploads a written file to server, he will first fetch the latest version of the file and increment this version by one, and chunk the whole file in to different pieces for uploading. It will update the cache and server at the same time.

2. **File Management**

   a) **RPCFile Class Hierarchy**: Files during the RPC process are represented through a hierarchy of classes. The base class is RPCFile, with three subclasses: **TmpFile, CacheFile, and ServerFile**. TmpFile represents a temporary file created during write operations. CacheFile represents a file currently stored in the cache. ServerFile represents a file stored on the server. Each class contains attributes like path, size, version, reference parameters, and expiration status to manage the file's state throughout its lifecycle.

3. **Consistency Model/Cache Mechanism**

   a) **Consistency Model:** The design aims to ensure that multiple clients can fetch file information concurrently without interfering with each other's

operations. This is achieved through: Read Locks on the server's download method to allow multiple clients to read concurrently. Write Locks on the server's upload method to ensure that only one client can perform write operations at a time.

b) **Cache Mechanism:** The cache is managed through a priority queue, leveraging a min-heap for cache eviction based on two criteria: Reference count, with files having a count of zero prioritized for eviction. Least Recently Used (LRU) metric, with less recently used files being evicted first.

c) **Cache Implementation:** Implemented using a priority queue that automatically sorts CacheFile instances based on reference count and LRU metrics. To maintain cache freshness, the system checks for stale CacheFile instances marked as expired and without references during open and close operations, evicting them promptly.

4. **Concurrency Handling**

a) **Concurrency in Cache:** The system distinguishes between readers and writers to ensure that read operations do not interfere with write operations. Readers directly read from the cache without additional copying, optimizing space and performance. Writers copy the content to a TmpFile before operating, ensuring that ongoing read operations are not affected.

b) **Concurrency in Server:** Concurrency is managed through the use of read locks in the server's download method and write locks in the upload method. This ensures that multiple clients can concurrently access file information while restricting file downloads to one client at a time.