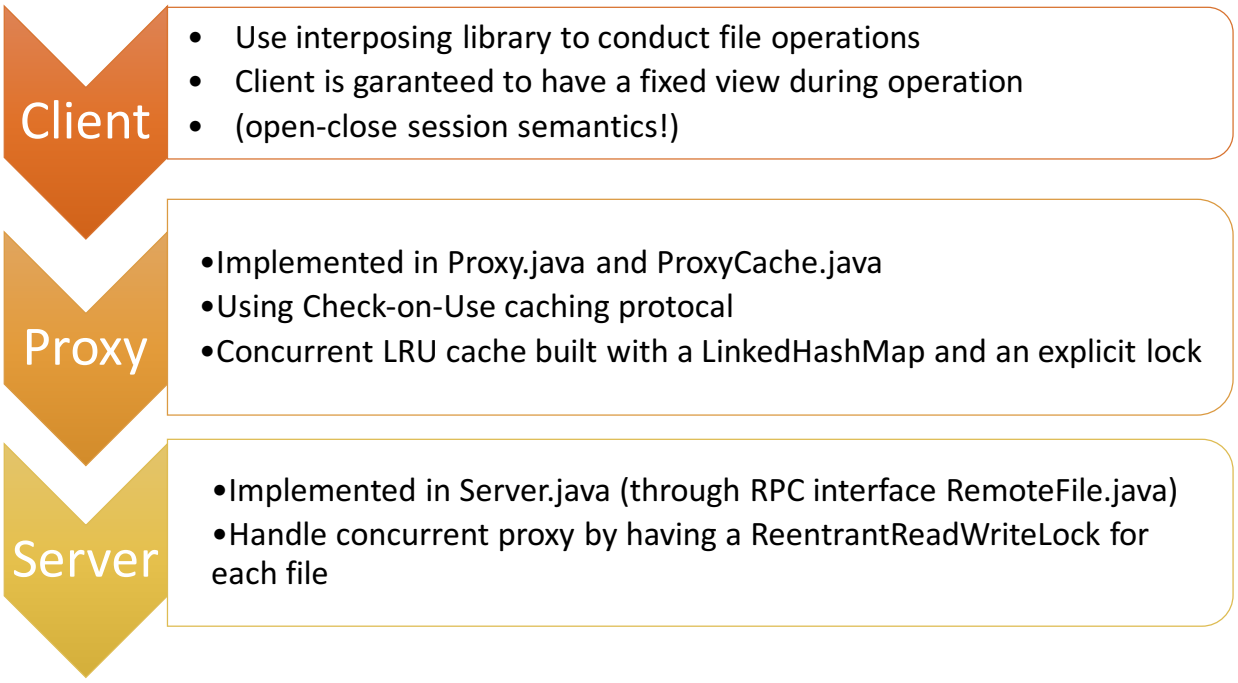
Design Notes

Yuqi Liu <yuqil@andrew.cmu.edu>



Overview of the design

1. Caching Protocol between Server and Proxy

In this Project, “Check on use” is used as the caching protocol between server and proxy. Every time a client tries to open a file, it firstly checks with the server for the file’s latest metadata. If some error occurs, return the errno. If the file is the newest in cache, then it reads from cache. If the newer version version being detected or a cache miss happens, the file would be transferred through chunks.

If a write operation closes the file, the proxy write back the data to server in chunks. To avoid race condition, the proxy will firstly write to a shallow copy then overwrite the master copy.

The file structure is different from server to proxy. In proxy, the file is maintained in a flat structure in the cache directory without any subdiretory. This is achieved by mapping “/” in file path to a very rare string “%’%”.

The different file version is represented by the last modified timestamp of the file (which is a long number). The read cached file is named as path_r_timestamp. The write cached file is named as path_w_fd_w_timestamp.

2. Consistency Model of the Client

In this project, the client sees a fixed view during the entire operation. So ‘’open-close” session is used as the consistency model. After close, if the file is updated then the data will be written back to server. Last writer to the file wins meaning that the final version would be determined by the last writer.

3. LRU Cache Implementation

LRU cache is implemented in ProxyCache.java. It is implemented by a LinkedHashMap which will automatically maintain LRU order in the list. Every entry in the cache has three attributes: file name, file length and file reference count. When the cache is full, it will evict the LRU one with a reference count of zero.

Also, when a new version of file being detected, the old version will be deleted once its reference count becomes zero. Since the old version and new version has the same prefix name, so this could be detected by comparing their timestamp.

To achieve concurrent operation on LRU cache, an explicit lock object is used to synchronize all the cache operations. This will decrease the system performance but we have to ensure that every operation to the linked list is sequential.

4. File Transfer Handling

File transfer data format is handled in FileData.java and ReadFileData.java. FileData.java is used to transfer file’s metadata while ReadFileData class is used to transfer real file content when reading or writing back.

To avoid running out heap space, chunking is used in the transfer with maximum length of 409600 bytes. Concurrency during chunking is handled by making a shallow copy of the file.

5. Other design decisions

(1) At server side, concurrent operation is handled by a ReentrantReadWriteLock per file. So we can guarantee that the file operation is atomic.

(2) At client side, since every writer its own file. No lock is needed. Reference count is maintained for a read copy.