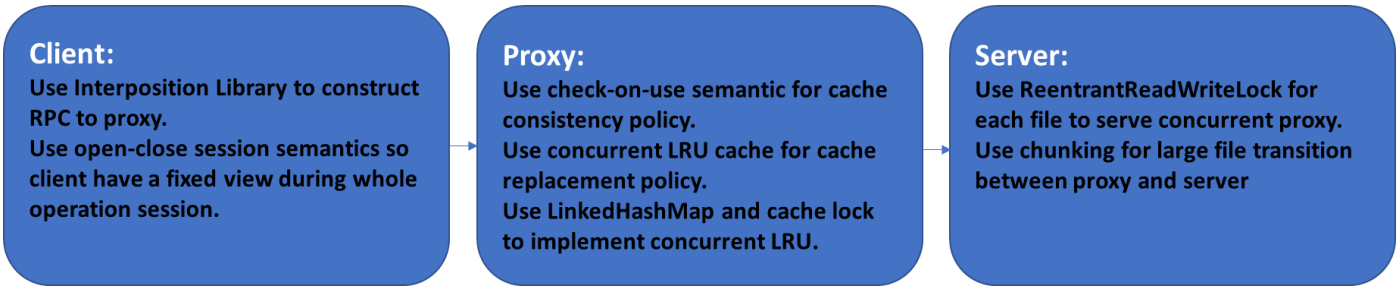
# Handout Note

Andrew ID : yunchens

**Architecture figure:**




1. **Cache consistency policy**

   The cache consistency policy in this project is check-on-use. For open function call from client, it will use RMI to check the file info at server, which includes file version, file exist, file is directory… etc, if there is an error, it will return an errno to client. If the file version at server matches the version at proxy, it will open proxy cache, otherwise it will send the current version to proxy and save as a cache.

   For a close call from client, the will close the proxy cache file and send the file back to server as the latest version, and then remove the file and reduce cache size at proxy if it is a write file. Each open file session will have its own version store at proxy, so a write to an open write session will not affect an open read session (assume the open read session open first). The modification will eventually show up at sever side when the written file is closed.

2. **Open-close semantics from client point of view**

   In this project, we can achieve open-close semantics. The open is like “git pull” from the remote server and open a local synced file, but each open create a new branch, so each open will not affect other. The close of a written file is like “git push” the current branch to the remote server, it will send the current version (branch) to server and becomes the latest version (HEAD). To achieve this structure, the proxy file structure is flatted and every file prefix is the server path name with each “/” replaced by “”, and the whole file name is : “path”+”_r_”+Tostring(version_info) for read file, and : “path”+”_w_”+ fd+”_w_”+Tostring(version_info) for write file.

   The client who opens a file will see the constant file content until it closes the file, the file will exist even if other client removes the file. This is achieved by different file name for each open call. If an open write file will be closed, the file will be sent to server and become the latest version. The latest closed file will become the final version no matter which file is opened first.

3. **Concurrent LRU cache at proxy**

   The LRU cache replacement policy is implemented by Java LinkedHashMap class and a cache lock for concurrency. The LinkedHashMap API is the same as HashMap but it maintains a sorted HashMap by using double LinkedList, the get(key) function call will move the key pair to the front so the HashMap

can remember who is recently used, and when the cache is full, it will evict the LRU one with a reference count of zero. Every entry in cache has three attributes: file name, file length and file reference count. When a new version of file is created, the older version will be deleted if its reference count becomes zero. Since the old version and new version has the same prefix name, this could be doned by comparing their verion_info. A cache lock is used for synchronized () to serialize all access and modification of cache object, it will reduce the concurrent performance but it will ensure the correctness of cache related operation for concurrent client.

4. **File object for transfer between proxy and server**

   MyData class and MyDataRead class is the file class for transferring data.
   MyData is used for transfer both data and metadata, including file_error, file_is_directory, file_error,…etc. The MyDataRead is for transferred data when reading from server or writing back. To avoid running out of heap space, chunking is used in the transfer.

5. **Other design choice**

   To handle concurrent proxy, the server uses ReentrantReadWriteLock for each file, so each file has a lock for read/ write operation, and concurrent read is enabled by share readlock.
   To handle concurrent client, not extra thing is needed since we already make unique file name for every open function call, all file will be deleted at close and send to server for update if needed.
   For each file in cache, the reference count is maintained, so the evict operation of LRU and update action by close function call can check the reference count before delete the file entry in cache.