Dan Cheng
15440-P2

Protocol between proxy and server

Java RMI to implement communications between proxy and server

Server

It has the following functions

- **downloadFile**
- **uploadFile**
- **initVersionNum**
- **getVersionNum**
- **getClientID**
- **setClientID:** Everytime a client connects to a server, it gets a unique clientID
- **rmFile**

It also has a **version map** which stores the most recent updated version number of the original file.

Proxy:

It has a *LRU* class to ensure cache freshness.
It has a *FileInfo* class to keep track of all the fd, make sure a client get unique fd for different files to support concurrency. It also stores the information of whether the file is a private cache copy, and the path of original file and path of the cache.

Workflow:

1. When a proxy get a path, it first simplify the path(remove any ".." or ".").
2. Check whether there is a cache for this path,
    a. if there is, it is a hit, we compare the version number of the cache with the original file from the server, if the version number matches, we use the cache, else we download the latest version from the server to the cache.
    b. If there is not, we create a cache, and download the content of the original path file from the server to the cache.
3. Create a private copy for all the open options except READ, (named as filename with clientID) of the cache, READ can share a file, and write will modify the file, and in order to support concurrent process, we need a private copy.
4. Mark the cache as being used in open, so other concurrent process cannot delete it.
5. In close,
    a. If using a private copy, we update the cache with the private copy and upload the cache to the server to the original file, and also updates the version number of the file.
    b. If not using a private copy, just upload the cache to the server to the original file, and also updates the version number of the file.
6. In unlink, we delete the file and delete the cache if it is not being used.

LRU Implementation

For LRU class, I used linkedlist called *cacheList* to keep track of LRU and MRU.
I used a *cacheMap* which maps the original path to a class called *CacheInfo*, which stores pathname of the cache, size of the cache, version number of the cache , and whether is cache is modified by write.

Dan Cheng
15440-P2

1. Every open, I add the cache to LRU, mark the cache as being used in *useMap*.
2. Every close, I put the cache at the beginning of the least(freshest), and mark the cache as not being used in *useMap*.
3. In unlink, if the cache is not marked as being used in *useMap,* we delete the cache.

Eviction implementation:

Every time we add a cache to the *cacheMap*, we compare the cachesize with the cachesize limit, if it exceeds the limit, we evict the LRU(iterate from the end of the linkedlist cacheList and get the LRU which is not marked as being used in *useMap*) until the cachesize is smaller than the limit, if we cannot make space for the cache that we want to add(either because we cannot evict cache that are being used, or the newest one is bigger than cachesize limit), I return a memory error.