File Caching Proxy

Senyu Tong

1. File Forms and Data Transferring Techniques:

A. *MyFile:* the file object stored in Proxy cache, containing:

- path: the original path name the client calls on; cachedPath: where it actually locate;
- lastModifiedTime: the version number, for checking if the file is up-to-date;
- readerCount: we only have one copy for every reader, 0 to evict the copy.
- Other file information and the actual random file to be operated by clients

B. FileInfo: used for data communication between proxy and server,

- Contains most information as MyFile does;
- Errno: proxy pass both path and open option to server, check for error at server side.

C. We have several path name transferring functions, as getting read/write copy, getting original master file name from the copies, and for subdirectory filename including '/' we flatten it to '_' .

D. Chunk structure named Bus, contains buffer and size for uploading and downloading files after writing down or adding to cache.

2. Proxy Cache Structure:

A. fMap and vMap, two hash maps with file path name as keys and MyFile instance or version number as values. More efficient for checking if a file is in cache and check for if it is out of date. Related function is "push", for importing info of files into cache and increasing cache size, "findVer" and "lookUp" for getting values.

B. The LRU queue with master copies as nodes, append to the head and delete from the tail. Related functions are "add", pushing file into the queue.

C. Function "hasSpace" check for if eviction is needed when new file from the server is to be downloaded or new copy to be pushed into cache. "MakeRoom" use LRU standard for eviction.

D. Function "cover" is for overwriting written copy to the master copy.

3. Caching Protocol:

- We use check-on-use protocol. When a client calls open operation, it sends file path to the server and server returns a FileInfo instance containing file Version, then the proxy check for the versionMap in the cache and decides if downloading for latests file is needed.
- If a client calls open with option "READ", after checking version, if there is a cache hit, then we check if there already exists a read copy. If so we add the reader number of that copy and just return that copy, so that we only have one copy for every reader.
- If client calls "open" with other options, then we just create a write copy, the path name is distinct for we associate the file descriptor number as suffix.

- If a client calls “close”, we are guaranteed that we are on a R/W copy. We check first the “open option” which is stored in the MyFile instance. If it is read-only, we are decrease the reader count by one, and if it hits 0, we delete the copy, but if the master copy is already evicted for reading session too long, we instead save it as the read copy; if it is not read-only, we update the new file to the server with the help of chunk “Bus”, and overwrite the original file. We update the file in the cache for this is the most recent file we have used.

4. Consistency Model

The whole operation is transparent to the client, since they are always operating on cache copies, but by our naming regulation (1C), and file structure (containing metadata and RandomFile itself), they will view these operations local.

5. Performance

- After a client done writing, though we have overwritten the master copy, but we did not update the version number. I noticed that I have passed the test, so I did not make other simpler structure for such tiny information, only the version number, transfer. To improve performance, this will be a great way.
- By our copying mechanism, only one read copy for a version of file, so we avoid filling the cache with too much identical data.
- In the cache, every read/write copies and master files are in the hash maps, makes it faster for checking version and getting file info, but only master files are in the LRU queue, since we will only evict those files that are not in use, but every copies are being using and they will evict themselves once they are done.
- To keep relatively good modularity, we use many java classes, as Cache.java and Bus.java, and we also define two file structures, with the difference in the one is for proxy-client, and the other for proxy-server.