Name: Ming Yao AndrewID: mingyao

15-640 Distributed Systems Project 2 Report

1. proxy-server protocol

The server provides several remote functions to be called at proxy. These functions are defined at interface FileService and implemented in the Server class. The remote functions are following:

Function name	Usage
sendFile	The server send a file into a byte array and transfer to proxy
receiveFile	The server read a byte array transferred from proxy and write it into a file on server
fileExist	Judge whether a file is exist on the server
createFile	To create a empty file on the server given a path
deleteFile	To delete a file on server
metadata	Collect metadata information of a file on server, include a file's version, file length and if this file exist

To reduce the latency between the proxy and server, the metadata function is designed to collect several information of a file and transfer to proxy only use 1 RTT.

To support read by chunk, the sever send the content of a file into a byte array, which has a chunk size limit, I set the chunk size around 1 MB. If a file is larger than the chunk size limit, we will split the file into chunks and call sendFile function several times to send the file to proxy. Similarly, the server also receives file contents from proxy by chunk.

2. LRU cache

I used a linkedlist to implement the LRU cache. The elements of this linkedlist are instances of a class called cacheElement which contains a file's path, how many clients are using it, and whether it should be deleted after close (when it's a obsolete copy but someone still using it).

When a new file is cached, we add the corresponding cacheElement to the head of the linkedlist. When a file is opened or closed, we update the linkedlist by move the element to the head. In this design, the head element is the most recently used file and the tail is the last recently used one.

When the size of the cache is not enough and we should evict a file, we start from the tail and choose the first one which no one is using to evict from cache. As stated in part 1, there is a attribute in cache element to record number of users using this file. So every time a file is closed, the number of users will be decreased one, only when the number of users is 0 can the file be deleted.

To ensure the cache freshness, every time when a file is opened with write/create/create_new option, the private copies should be deleted after the file is closed. And every time when we open a file, if the master copy is obsolete, a new version should be fetched from server, the original master copy should be deleted, but if some users are still using it, we can't use it directly, instead I add a Boolean attribute to record whether it should be deleted. When close a file, if the this attribute is true and no one using it, then we delete it immediately.

Because a linked list is used to implement the LRU replacement policy, when we want to find a specific file element for the purpose of move it to head or delete it, we will go through the list, which the complexity is acceptable in this project. If a linked list is too long, just using the list may be not so efficient, in this case I can use a hashmap to record a file and its corresponding cache element, thus can find a file in constant time.