Major design aspects of File-Caching Proxy

jinxi@andrew.cmu.edu

(1) master-copy and write-copy

(a)When to create copies:

if it is write operation, always create a private copy called write-copy. First check the master-copy in cache, if not cache hit, delete the expired master-copy, fetch the new master-copy from server, then copy the master-copy as write-copy. This write-copy will only be used by this write operation, never shared with others. When the write operation finished, upload this write-copy to server and delete the write-copy in cache.

If it is read operation, there exists situation that several read operation share the master-copy. First check the master-copy in cache, if not cache hit and no one is using it, delete the expired master-copy and fetch new master-copy from server.

(b)When to delete copies:

As for write-copy, when it is going to close, always delete the write-copy in cache.

As for master-copy, when it is going to close and find this process is the last one who are using the master-copy, then check whether the version of this master-copy is the newest version from the view of proxy. The current version in server is not visible. If not, then delete this master-copy, otherwise save.

(2) LRU replacement:

LRU maintain a list of all the current files in the cache. Most recently used file is near the head of the list. In the open() and close() function, put current being used file to the head of the mru-list. When we need to fetch file from server and create write-copy, we need to check the freesize of cache, if there is not enough space, we need to delete the file in the rear(making sure no one is using it).

(3) Protocol between server and client:

Every time client accesses th open() function, it will go to server and fetch all the info about the specific file.

```
class MetaDataofFile implements Serializable{
      long version;
      boolean isDir;
      boolean isExist;
      boolean isHit;
      long filesize;
      boolean isPermitted;
}
```

Then client holds the metaData of file, it can do the all the decisions, no need to go to server again. Only in the situation that cache miss, client will access server again and fetch the newest file. So in the last latency test, can keep the RTT as less as possible.

Test: Latency test *Got results: 1 315 605 102 102*
read small file: 3 RTT
read medium file: 6 RTT
read small file again: 1 RTT
read medium file again: 1 RTT

(4) Transfer huge file between proxy and server

(a)Upload huge file from proxy:

assume the file named “foo”,every time transfer chunk size (1MB) of 'foo' and generate 'foo.001','foo .002'.... files in the server, then merge the chunk files 'foo .001','foo .002'... in server to a

huge file 'foo', delete the chunk files.
(b)download huge file from server:

assume the file named “foo”, split the file into several chunk files 'foo.001','foo .002'.. with the size same as chunk in the server, then transfer the 'foo.001','foo .002'.. to proxy, finally merge the file in proxy and delete the chunk files in server.

(5) How to record the users and newest master-copy

(a)Set a hashmap named counter as below to record the current user of the specific file, when open(), counter +=1; when close(), counter -= 1;

```
public static Map<String, Integer> counter = new HashMap<String, Integer>();
```

(b)Set a hashmap named most new version as below to record the current newest version of file in cache.

When download new version from server, put it into hashmap.

```
public static Map<String, Long> mostNewVersion = new HashMap<String, Long>();
```