15640 p2 implementation design

author: QIYANG HE AndrewID: qiyangh

Basic design idea

- This file caching proxy is designed based on session-semantics, use check-on-use to ensure validity of caching files.
- Client will see consistency of files during the whole file operation. Write copy will be created for each write user. If concurrent write happens, files will be changed based on last-write-win principle.
- Protocols between server and proxy:
    - In most cases they transfer file content/metadata such as file version, file length, operation error messages within a self-made class.
    - For chunk read-write, a different class is used to transfer file data, content offset as well as file length between server and proxy.

Server implementation:

- The server uses multiple concurrent methods to ensure handling concurrent file operation successfully.
- Data structure:
    - A concurrent hashmap is used to store filename and related reentrant read-write lock object.
        - We also used an explicit lock object to handle insert/remove elements from the list.
        - When handling read action, assign read lock for files, multiple read lock can be assigned in one time.
        - When handling write actions such as push new file version/delete file, assign a write lock. Only one write-lock can be delivered at one time.
    - A concurrent hashmap is used to store temp file which is created for handling chunk read-write requests from proxy
        - Key is the temp file name, value is the status of this temp file
        - This design helps to ensure there will be no concurrent conflict happen when doing chunk read-write, and ensure other operations happen concurrently because the whole structure will not be locked.
- Handle chunk read:
    - When bytes being read is too large, server will create a temp file on server (copy origin content to a temp file) and assign a file link back to the client
    - Client call read recursively to get full bytes, until all bytes has been received
- Handle chunk write

- When proxy push modification to server, and content length exceeds max chunk size, chunk write happens
- An rpc call will be made to create a blank temp write file on server to receive bytes
- Client recursively call write to server, until all bytes has been transferred
- When it is done, an rpc call will be made to push temp file on server to replace origin version. This temp file will also be deleted

Proxy implementation:

- Proxy run a cache to serve client files based on LRU rule, it creates a cache when first time initialized.
- For each client there will be three session hashmap storing assigned file descriptor with Random Access File object, file operation permission(r/rw), and related file paths.
- File descriptors are assigned based on session principle so there will be no conflict between fds for different clients.
- Cache design
    - Basically, a file access number is used to indicate the file operations that are executed on this file. If there is no operation on this file, the accessNum should be zero.
    - Data structure:
      There will be two hashmap storing file information, each of them serves for different purposes.
    - A linked hashmap is used to maintain LRU rule for original copies
        - Each entry has key: filename, value: a class storing their own status
        - An example of cache entry is:
            - <key: /cachedir/a.txt>
            - <value: class { file name: /cachedir/a.txt__1678__r,
              file version: 1678,
              file size: 30,
              file accessNum: 6, (total count of r/w access)}
        - An explicit lock object is used to synchronize structure when insert/delete elements.
        - For each file access(open/close) the access number will be adjusted, the file object will be removed from structure and re-insert to hashmap indicating it is a most recent used object.
    - A concurrent hashmap is used to store temporary files such as temp write file and read file that is not most updated but still be used
        - Each entry has key: real file name in memory, value: a class storing their own status
        - An example of temp file entry will be
            - <key: /cachedir/a.txt__1333_w> //a write temp file
            - <value: class { file name: /cachedir/a.txt__1333__w,

file version: 1678,
file size: 29,
file access number: 1,
dup: class {file name: /cachedir/a.txt,
file version: 1678} }

- A dup class is used to storing the origin copy info that the temp file is cloned from. This is used for purpose that we have to adjust access number of these origin file after we close these temp files.
- When a new file has to be pushed (e.g. a write happens on a cached file), but the file is still being use. We have to remove origin file status class into temp file list, then create new class in linked hashmap to provide most updated file version for users.

- Evict rule
  - Any file in temp file hashmap cannot be evicted, these files will automatically be deleted after all operations are done (e.g. access number became zero)
  - For files in linked hashmap, only if files with zero access number can be deleted/evicted.
  - Each open/close call on file is counted as a file access. These files will be removed from Hashmap first to adjust their status and then re-insert into the data structure. This implementation could ensure we are evicting file with least recent used property when evicting files by looping through the linked hashmap.