CMU 15-640 Project 2
Design Document

Xiaoyun Ye
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
xye2@andrew.cmu.edu

System description

This is a remote file operation system with whole file caching. It includes a server supporting remote read, modify, and delete operations in local file space, and a proxy which handles the RPC file operations from the client interposition library, fetches whole files from the server and caches them locally.

Caching Protocol between proxy and server

The proxy provides whole file caching based on open-close session semantics. The server keeps the master copies of every file and the latest version number of files.

When the proxy calls `open` function, the proxy first gets some information about the file from the server before retrieving the file from cache or from server, which includes the file existence, directory, the latest file version number and the file size.

For large file transmission, the server and proxy will divide the file into chunks and only when all chunks has been transmitted will the operation be marked as done.

Fetch policy: On-demand caching
The cache only caches a file when the client requests it, in this case, when `open` function is called.

Update propagation policy: Check-on-use
The cache checks the master copy before each use and fetches the file conditionally if the current version in the cache is stale. The check is implemented by comparing the version number of the local copy and the master copy.

Cache replacement policy: LRU based fresh copy
The least-recently-used file is evicted from cache when more space is needed. The cache also actively deletes stale copies once being detected, instead of waiting until the storage is full.

Consistency model

When `open` function is called, the proxy will fetch the file requested from cache or from server(when cache misses or version stale) and provide the local copy for client to access. To be consistent, the proxy uses `synchronized (sync_object)` to synchronize among multiple threads before assigning the file descriptor for the new file.

Any operations called between `open` and `close` will only be valid on the local copy, having no effects on the master copy. And any operations on the file from other concurrent clients during this time period will not be valid on the local copy in order to keep it consistent.

Only when `close` function is called, will all the client operations be updated to the server. In order to keep consistency, for `write` and `unlink` operations, the system also uses `synchronized (sync_object)` to synchronize for multiple threads, implementing the operations from concurrent clients in an ordered way.

LRU replacement implementation

The system uses a double-linked list to arrange the cache. Every file in the cache is represented as a node in the double-linked list, with one pointer to its previous node and one pointer to its next node. The system maps a file's path to the a corresponding cache node in the list.

Whenever the system wants to cache a new file/version, it initializes a new cache node, which will be added to the tail of the linked list, so the spatial order in the list represents the temporal order of file use.

The LRU replacement will take place when the cache needs to free some space for new cache files. To evict the least-recently used file cache, the system keeps removing the head node of the list until enough space is released. The LRU replacement also evicts the stale version files from the cache, which is done by mapping the file path to the cache node and removing the node from the list.

Cache freshness maintenance

Every time the proxy wants to open a file, it first gets some file information from the server, and compares the version numbers of the local copy and that of the master copy. If they don't match, the proxy will delete the local copy and fetch the file from the server. The proxy also keeps checking whether there are stale versions of a file when opening and closing the file and deletes the stale copies immediately.