# Project 2 Design Document

## Consistency Model

The system adopts a check-on-use approach introduced in AFS-1 to ensure whole-file session semantics at open-close granularity. In this approach, a server is stateless and only tracks the version of each file. All files on the server start with a version of 0 and the version number is strictly increasing. On the proxy side, there are two possible scenarios everytime a client tries to open a proxy file. If the proxy copy exists and its version is up-to-date, the client will directly operate on it. However, if the proxy file doesn't exist or is outdated, an RPC will be made to read the file contents on the server, copy them and create a new file on proxy, and update the corresponding file version. If an error occurred on the server side during this process, the server will send back the appropriate negative errno instead of the file contents.

Within an open-close session of a client, it sees a consistent view of the file even if a new version is fetched from the server. On the first write call to the file, the proxy will copy the original content and create a new file that is only visible to that client. Subsequent read/write/lseek calls on the file from this client will be directed to this new copy. When a client closes a file that has been previously written to, an RPC is invoked to update the file on the server. The server updates the version number and assigns it to the proxy file, before making the proxy copy visible to all clients.

This consistency model is efficient as no read/write lock is implemented on the server side. In case of write-write conflict on the same file, the last close after writing wins.

## Proxy-Server Communication Protocol

Three RPCs from proxy to server are implemented using Java RMI library.

- *ServerFile readServerFile( String path, OpenOption o, int proxyVersion, int offset)*
  - This RPC opens and reads the file at the specified offset if the version disagrees. A ServerFile class is created to encapsulate the return value. A ServerFile object has a boolean flag that indicates if open/read calls on the server file are successful. On success, It stores the actual file contents in bytes (or an empty byte array if proxy version is up-to-date), the server-side file version, and the file size to help proxy check whether any additional read RPCs are needed to fetch remaining file contents due to chunking. On failure, it stores the negative errno.
- *int writeServerFile( String path, byte[] content, int offset)*
  - This RPC writes the new content starting at a given offset to a server file. The new file version is returned to the client.
- *int unlinkServerFile( String path)*
  - This RPC deletes a file on the server and increments the file version. Lazy deletion is performed on the proxy as the proxy copy can be concurrently used by a client so it will be invalidated on the next open. Proxy can determine if the deletion is successful based on the return value.

Chunking is used to limit the data transfer of each RPC. The chunk size is chosen as 400,000 bytes as it empirically provides a good balance between latency and memory/bandwidth limitation on autolab test cases.

Notice that the read/write/lseek operations are not implemented as RPCs as they are performed on proxy copies under open-close session semantics.

## Cache/File Implementation

The proxy maintains a strictly increasing running integer and assigns them as file descriptors when new files are opened. A hashmap is used to maintain a mapping from a file descriptor to a RandomAccessFile object for subsequent read/write/lseek/close operations.

The proxy also implements a whole-file cache with LRU eviction policy. Each node in the cache is an abstract representation of a proxy file with its meta-data, such as the file name and version, the reference count (the number of current clients using the file), the size of the file, and whether the file is visible to only one client (a local write-copy) or all the clients. A doubly linked list is used to track the file access history for LRU eviction as it offers O(1) time complexity for single node reordering and removal. The cache also uses an internal hashmap to efficiently locate a file node, where the key is a server file name and the value is an array list of nodes corresponding to the server file's copies on the proxy. A traversal on the array list can retrieve the latest proxy copy of the server file. The proxy also uses a hashmap to associate each file descriptor with a file node in the cache.

To ensure cache freshness, all stale copies of a server file will be evicted from the proxy cache when a new version of the same file is fetched from the server on Open, or when a local write copy of the file is closed and its changes are propagated back to the server and made visible to all clients. A proxy file is stale if and only if it is outdated and not currently used by any clients (reference count = 0).

Additionally, cache eviction is needed when adding a new file or updating the size of a file exceeds the cache capacity limit. A file use is defined as a close on the file descriptor, so that the corresponding node can be relocated to the front of the linked list. Eviction is performed by traversing the doubly linked list in LRU order and deleting the file from the proxy if it is not currently in use. Although maintaining two separate lists (one for evictable and one for unevictable files) have potentially faster run time, this approach is chosen for its simplicity.

## Concurrency Control

All methods in the proxy class, the cache class and the server class are declared with the synchronized keyword to provide instance-level serialization of these methods. ConcurrentHashMap and ConcurrentLinkedQueue are used instead of HashMap and ArrayList to provide thread-safe and atomic operations on internal data structures such as the cache.

On the proxy side, the intrinsic lock of the cache object is used to handle possible race conditions and ensures that only one client can update the cache contents (e.g. adding or removing nodes) at a time on each proxy.