Everett Periman
October 27, 2021
EEL 5737 – EDGE
Professor Renato Figueiredo

# Homework 3

**- Race Free Condition Free Operation**

In order to control the race conditions between the clients a lock is Acquired on the server using the RPC RSM method anytime that an input is entered. The input will then be processed as usual and then upon the return of the corresponding input method the lock is Released on the server, by using the RPC Put method, which inserts the unlock state into the server.

I tested race conditions by launching one client will no delay in releasing its lock, and then launching another client with a 5 second delay in releasing its lock. When I enter a command in the delayed terminal first and then try and execute a command in the nondelayed terminal the nondelayed terminal sits in a "Spinning" state until the lock is released.

**- Cache Operation**

Two new methods were created to manage the cache of the client as well as to indicate to other servers that they have a stale cache. To implement this caching system a cache timestamp was added to the server to indicate the last time that anything on the server was updated. Given this the first method that is responsible for clearing its own cache will first check the servers cache timestamp vs its own cache timestamp. This first method is ran at the end of each Acquire to ensure that the cache is always up to date. If these do not correspond then the client voids its own cache and updates its timestamp. The second method that was added allowed for the client to invalidate the cache timestamp of the server. This method is only called on methods that would create modifications of the server. For example of operation, on operations such as "cat", the ServerTimestampUpdate method is not called but the client does use the ServerTimestampCompare method to decide whether the "cat" method should read from cache or dump the cache and read from the server.

**-Exponential Back Off Spin Lock**

When utilizing the original spin lock it would continuously bombard the server and even caused reduced performance of the operation of the second client. After implementing the exponential back off spin lock I ran an experiment to assess the rate of access attempts in the new spin lock vs the old spin lock. By utilizing time.time() recording in the acquire and release stages I was able to obtain this data. For a delayed time period of 5 seconds using the original spin lock condition the client issued 13421 requests to the server to obtain the lock, vs the exponential back off spin lock only made 4 requests for the lock. The biggest issue with the exponential back off is the time utilized is much greater. The original spin lock was able to take control almost immediately after the lock was released whereas the exponential spin lock generally required more time depending on how long the original lock was in place.