**CSCI-GA.3033-017 Special Topic: Multicore Programming**
**Lab 4 Assignment (Part 3: A Complete Storage System)**

In Lab 3, you worked on understanding the performance of an in-memory key-value store. In this final lab, you'll expand your system to use the in-memory portion as a cache in front of an on-disk store. You'll have to make the following additions to your system:

1. If you didn't already do so, make your system continue to accept and handle requests from a single connection until the peer (ie, the client) has closed the connection. Your changes should be relatively minimal. You can test this change using httperf: just remove blank lines between httperf workload file requests to make them be sent in the same connection.
2. Implement an on-disk store. All this means is (for example) providing a directory as an argument to your server holding one file for each key-value pair your server is storing.
    a. The in-memory cache should store a strict subset of the key-value pairs on disk. If a file does not exist in the on-disk store directory with the name of a key being looked up, that key is definitely not in the store. If a file does exist, that key **may** or **may not** exist in the in-memory cache. Your in-memory cache must be limited to at most 128 key-value pairs.
    b. I will provide a new script to produce httperf workload files for this lab. Those workloads will use a Zipfian distribution: that is, they will focus activity on a few "hot" keys, with occasional access to "colder" keys.
    c. You don't need to bother trying to determine the key-value pairs already in that directory when the server starts. You should consider making the server wipe the directory clean when it starts. **TAKE GREAT CARE TO NOT WIPE YOUR CODE DIRECTORY!**
    d. The server should open, write, and close the relevant file each time an insert occurs.
    e. The server should attempt to read the relevant file each time a lookup occurs only if the in-memory cache does not currently hold the given key.
    f. The server should delete the relevant file, and if it exists, the relevant in-memory cache entry, when a remove operation occurs.
3. Your in-memory cache should intelligently choose when to add and remove a key-value pair to the cache (phrased another way, when a key should only be stored in the on-disk cache).
4. Provide measurements of server-side (only) performance with the cache + on-disk store, and with just the on-disk store (ie, simply make your code ignore the in-memory cache). You need not bother with client-side measurements for this lab.

Other than the completed, functional code, you should also add a short document describing the performance just using the on-disk store, and adding the in-memory cache in front of it.

**Measuring and Reporting Performance:** As in Lab 3, try to measure at least as many threads as your testing machines have cores (try to find machines with at least 4). Remember, the client and server should be run on different machines to avoid contending on CPU and I/O resources.

**Grading:** This lab will contribute at most 25% to your lab score. It will be graded on functionality, adherence to the specification, thread safety, code style, commenting, and thought put into the analysis. Proper thread safety and insightful analysis will be given an inordinate share of the grade in this lab, for obvious reasons.

**Getting Help:** If you're struggling, your first recourse should be the class mailing list. *Important:* you should solicit your fellow students for high-level help, such as "how can I iterate through a vector?". You should not ask for specific help with your code, such as "why doesn't the following code work?". If you're still stuck in a week, come to my office hours on Thursday. See the first lecture for the policy on collaboration with other students (tl;dr: don't, outside of the mailing list). As is university policy, instances of cheating will be taken very seriously. If you believe there's an omission or error in this document, you're welcome to email me directly.

**Due date:** December 8, 2016, by 11:59:59pm EDT. See the first lecture for the late policy. Do not save this lab until the last minute! You will need to be working on your final presentation by December 8.

**Submission:** Push your code to your **private** MulticoreProgramming repository in a folder entitled "lab4". Please also send me an email once you have committed and pushed your final submission with the tag (a 7-character hexadecimal string) of that commit; I will use the timestamp GitHub records for that commit to determine whether your submission is on-time or late.