CSCI-GA.3033-017 Special Topic: Multicore Programming
Lab 3 Assignment (Part 2: Performance)

In Lab 2, you created an HTTP-accessible, multithreaded key-value store that worked. In this lab, you'll check how fast it is, and add additional instrumentation to understand how each of the techniques you implement help its performance. In this lab, you'll do the following:

1. Store (and print) statistics about the requests received.
2. Determine the mean, median, minimum, and maximum request time, as viewed from the server and from the client.
3. Attempt to get the best performance possible Evaluate performance from 1 to N threads with a single global state lock, with the implementation you delivered for Lab 2, and with any additional multithreaded optimizations you can come up with.
4. Chart and report your results. You should explain in two or three short paragraphs any interesting phenomena you notice. A *non-exhaustive* list of examples you *may* want to consider are the relationship between the number of threads and the per-request performance, the relationship between mean and median request time, the relationship between the client- and server-reported statistics (where did the time go?), and why each of your optimizations yields the speedup it does.

Storing and Printing Statistics: Add a way to record and print the total number of insert, delete, and lookup operations performed by your system over its lifetime. Figure out how to report them: for example, you could catch SIGUSR1, SIGINT, or make the statistics be returned as the "value" for a special key in your store (or even simply store the three numbers in actual key-value pairs in your KV store).

Determining Mean, Median, Minimum, and Maximum Request Times: For the server-side measurements, this is less trivial than it sounds: you want to compute the overall statistics for all of your threads together. You'll find that this requires some synchronization, especially to compute the median. On the client side, you can use httperf's output directly.

Measuring and Reporting Performance: 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: November 17, 2016, by 11:59:59pm EDT. See the first lecture for the late policy.

Submission: Push your code to your **private** MulticoreProgramming repository in a folder entitled "lab3". 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.