CSCE-313

PA 2

Aakash Tyagi

Priyanshu Barnwal

10/8/2021

Starting off, we created a bounded buffer class to have a queue to push and pop threads. We put a mutex as a lock in this class. This means that any thread that would like to run, must be unlocked before it can do so. We also have two condition variables which notify one another of the unlocking of the mutex. The mutex and condition variable force the queue to check whether a pop or push is actually possible and they only allow the queue to perform the operation if it is possible so we don’t run in to errors.

The client’s main function is actually quite similar to the main function from PA1, a lot of the code can be carried over from there. However, there many new functions that must be considered, such as the histogram thread function, worker and patient thread functions, and the file function.

The worker thread function has a bounded buffer which contains all the worker threads. Same with the patient and histogram threads. Each thread performs a different task, such as collecting data, requesting data, and creating the histogram. W is the number of worker threads that will run at the same time, n is the number of data points we are requesting, p is the patient we are pulling the data from, and finally h is the number of histograms threads that will run to remove datapoints from the queue and put them into a histogram. Because of the condition variables we discussed before, we have these three threads perfectly synced up, avoiding any race conditions that may have occurred.

Finally, we have the file function. This function is very similar to PA1, with a few key differences. Firstly, it doesn’t need a conditional branch to run or a for loop, if we call it inside a for loop. We re-generate the file one by one using each thread to generate it. We use the buffer capacity to determine how many bytes to pull in each thread.

In the main function, we need to create the threads very carefully. One incorrect code placement and we run into horrific seg faults. We start by putting in the patients, then the worker, and then finally the histogram. This is because we want to request the data points, then store the datapoints, and finally create the histograms and un-store the datapoints. We also create a bunch of histograms and add it to an array with the histograms inside.

As we can see by the above data, it seems that the data cannot be reliably scaled. It seems like worker thread may be able to but it’s difficult to say, since there seems to be an outlier in the point where there are 100 threads. However, for data points, I believe it is reasonable to say that both the buffer cap and the worker threads can be scaled. However, histogram thread count is all over the place and it is highly unlikely that it can be scaled at all.

VIDEO PART 1: <http://youtu.be/MDcZ1ElPVUk?hd=1>

VIDEO PART 2: <https://youtu.be/DtbKq3Ba7dE>