Writeup

Testing for this assignment was really painful and tedious because there were tons of things going wrong that I could not figure out. Even with unit testing, I made sure to test each function one by one using many print statements along the way to determine where I could possibly be facing bugs. The thing about multithreading debugging is that it's hard to pinpoint where exactly my problems were coming from. Sometimes they would happen, sometimes they wouldn't, which would tell me that I was facing race conditions and that my threads were somehow accessing each other's memory. So, that led to a major debug of my multithreading code. For logging, I was having the most issues calculating how much each offset should be and accounting for all the edge cases. I would test each offset by working on a small file first so that I could manually determine how large it had to be then printing out the values after each incrementation of an offset to make sure it was on track. After tediously doing this for a few days, I finally managed to get correct offsets.

1.Start eight separate instances of the client at the same time, one GETting each of the files and measure (using time(1)) how long it takes to get the files. The best way to do this is to write a simple shell script (command file) that starts eight copies of the client program in the background, by using & at the end.

It takes my program roughly ~4s to GET 8 400MB files.

2.Repeat the same experiment after you implement multi-threading. Is there any difference in performance? What is the observed speedup?

Yes, there is a pretty big improvement in performance because of multi-threading. Multiple threads were able to handle multiple GET requests at once and not do it sequentially. I observed around a ~.25x in speedup.

3.What is likely to be the bottleneck in your system? How much concurrency is available in various parts, such as dispatch, worker, logging? Can you increase concurrency in any of these areas and, if so, how?

I feel like the biggest bottleneck in my system is when logging is enabled. I know I didn't make the most efficient use of my implementation for logging because I do it line by line instead of filling up my entire buffer and writing it into the log file that way. The slow down happens because I use a lot more pwrite operations (I/O operations) than I would like which definitely would worsen my performance as my program would spend a lot more time writing compared to other implementations. There is concurrency throughout all my functions since they need to be able to work with each other without stomping on each other's memory and data. I don't think I would be able to increase concurrency in my program because if I tried lowering the amount of

code I currently have locked, then I would run into many more bugs and race conditions, at least from my experience.

4.For this assignment you are logging the entire contents of files. In real life, we would not do that. Why?

We would not want to log entire contents of a file in the real world, because in the real world we could potentially be dealing with much larger files. 400MB is nothing compared to 400GB. In this case, the slowdowns of logging very big files like that would not be worth logging in the real world.