Brian Quinn

Multicore Programming

Professor Mitchell

13 April 2017

Lab 3 Analysis

The purpose of this lab program is to create and analyze a multithreaded server utilizing a threadpool in order to process incoming requests. This is achieved by creating a ThreadSafeKVStore in order to store information from incoming requests, creating a ThreadSafeListenerQueue in order to pass connections to the threadpool, and creating a ThreadPoolServer in order to set up and begin the server/threadpool. The implementation for these files is found in the lab3 folder. Analyzing the efficiency of the threadpool server provided the following server-side results:

Server Side	Global Lock (25 connections/threads)	RW_Lock (25 connections/threads)	RW_Lock (25/5 connections/threads)
Mean	12812.36 ms	12748.84 ms	18984.56 ms
Median	12825 ms	12794 ms	19033 ms
Maximum	13097 ms	12951 ms	19248 ms
Minimum	12273 ms	12228 ms	18389 ms

Running HTTPerf to create connection requests provided the following client-side results:

Client Side	Global Lock (25 connections/threads)	RW_Lock (25 connections/threads)	RW_Lock (25/5 connections/threads)
Mean	12813.1 ms	12748.8 ms	18734.4 ms
Median	12828.5 ms	12794.5 ms	18947.5 ms
Maximum	13097.2 ms	12951.0 ms	19107.1 ms
Minimum	12267.7 ms	12228.8 ms	17794.2 ms

Through the analysis of the two tables, a small discrepancy between use of a global lock and the use of read-write locks. The read-write locks are an average of 63.520 milliseconds faster server-side and an average of 64.300 milliseconds faster client-side. However, this average difference is likely bottlenecked by the tracking of the data. In order to properly record the server-side statistics, a number must be incremented for each lookup, insert, and delete. Since the ++operator is not atomic, we must lock to increment this number, which means a small critical region even for lookups.

Furthermore, comparing the server and client side connection times show that the times are nearly the same as long as the number of threads exceeds or equals the number of connections. This would be expected as the server will start the connection slightly after the client when it receives the connection request, however, it will also close slightly after. Moreover, when the number of threads is less than the number of connections, the connection time for the server was an average of 250.160 milliseconds longer than that of the client. The extended server connection time is a result that the threadpool picks up one connection at a time and processes it. Due to this, when the client closes a connection, records its connection time, and sends a message closing the socket, this message will not immediately be processed by the threads because they may be handling other requests. As a result, the client has already closed the connection, but the server does not process this until the file descriptor is picked up by the threadpool again.