Performance Analysis

A)

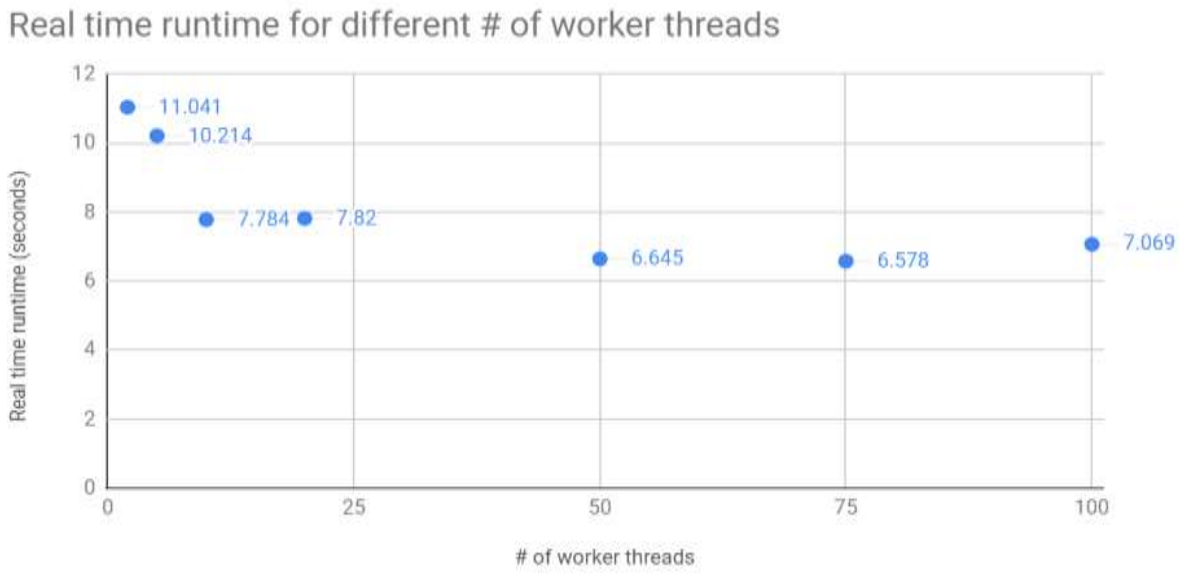
[2][1][5][/image/jpg/21.jpg][17772][4182 us][MISS]
[3][1][6][/image/jpg/21.jpg][17772][8 us][HIT]
[4][1][5][/image/jpg/23.jpg][17772][8138 us][MISS]
[5][1][6][/image/jpg/21.jpg][17772][8 us][HIT]
[6][1][5][/image/jpg/23.jpg][17772][18 us][HIT]

Above are the logs for several wget calls. The cache was empty and could hold 100 entries at the start. In this example there were 100 dispatcher threads, 100 worker threads, a queue length of 100, and no dynamic flags set.

As you can see, the cache “hits” are far less time consuming, taking between 8-18 microseconds. We noticed that cache hits slowed down a little bit after we changed our caching policy from First In-First Out (FIFO) to Least Recently Used (LRU). This is due to our LRU implementation doing up to cache_size array entry swaps in the cache on a hit. The cache “misses”, on the other hands, take 100s or 1000s of microseconds. Since cache “misses” must first check if the request is in the cache, this makes sense. Furthermore, once “misses” see that the entry is not in the cache, the program must then fetch the request from the disk. Accessing the disk is a long and time-consuming process, so the larger times make sense.

B) As you can see from the graph below, as the number of threads increases, the runtime decreases at a decreasing rate. While having more threads allows for greater concurrency, there will be diminishing returns. Threads have overhead and if there are more threads available than possible tasks, there will be slowdown due to queuing up threads.

As specified in the lab writeup, we used the Linux “time” command to measure the runtime and we reported real time. We ran the xargs test on the bigurls file as requested. After starting the server with the correct number of worker threads, we ran “`time cat <path _to_bigurls_file> | xargs -n 1 -P 8 wget`”.



Our results indicate that the optimal amount of threads appears to be around 75. This makes sense because there are 144 requests in the bigurls file. 75 threads would maximize the amount of concurrent work being done while limiting the amount of queuing.