Jiarui Li

**Data requests performance:**

Graph of runtime vs # of worker threads (with request buffer size fixed to 20)

(Other param info: p = 5, m = MAX\_MESSAGE)

In the above chart, the request buffer size is fixed, and I changed the number of worker threads varying from 20 up to 600.

I observe that when number of threads is less than 100, every minor increase in the number of worker threads (in the scale of 10) will significantly decrease the runtime. And it seemly follows a exponential scale but with negative coefficient because the runtime will not decrease so significantly as the number is close to 100. I believe the reason why the runtime decreases a lot in change of # of worker threads is that at this point, the number of data requests that are ready to pop in the bounded buffer is out pacing what the worker threads can handle, since only limited number of worker threads will make the buffer always be full or nearly full.

But as the # of worker threads increase, from 100 to 600, I hardly observe any change in the runtime and it is stable around 1.5 seconds. This is because now the request speed is faster than the filling speed of patient threads, and that makes worker threads wait for pushes to the buffer. And this makes the runtime be maintained at a minimal value.

Graph of runtime vs # of buffer size (with worker threads fixed to 100)

(Other param info: p = 5, m = MAX\_MESSAGE)

When changing the buffer size, I hardly observe any significant decrease in runtime, and I tried out sizes from 1 up to 80.

On a large scale, there is no significant change but for smaller buffer sizes (below 20), it seems that it is exponentially associated with the runtime. The reason is that smaller buffer sizes require more bound checks and waits and those operations slightly slow down the runtime.

**File requests performance:**

Graph of runtime vs # of worker threads (with request buffer size fixed to 20)

(Other param info: p = 5, m = MAX\_MESSAGE, file size = 10M)

(Other param info: p = 5, m = MAX\_MESSAGE, file size = 100M)

To figure out the performance of file request runtime vs threads, I created two graphs. The first is using small file size (10M) and small thread numbers (5 to 50). And I did not get a good relationship of the two variables. I thought it was the file size that was too small and that was not letting the program exhibiting its full power. So later I tried out using large file size (100M) and large thread numbers (50 to 1000). But still get the same result: # of threads doesn’t really affect the runtime of a file request.

I think this is because the time-consuming part of file requesting is writing to the file and the number of times a file is opened. That is a system call and requesting the I/O devices, which the speed cannot be easily changed even using threading.

Graph of runtime vs buffer size m (with request buffer size fixed to 20)

(Other param info: p = 5, m = MAX\_MESSAGE, file size = 100M, w = 100)

I tried out m = 256, 512, 1024, 2048, 4096, 8192 and observed a significant improvement of runtime when buffer size is doubled. To be precise, the runtime is nearly halved as m doubles.

This is because increasing the buffer size is namely increasing the window size, and that allows one fetch with one open file operation to get more data than before. Essentially increasing the buffer size is decreasing the amount of operations wasted on opening files, and that’s the main reason why we can observe such a big improvement in runtime.