Performance Analysis Report

CSCI 4061, Fall 2018, Project 3

Lee Wintervold, Jack Bartels, Joe Keene
University of Minnesota
12 December 2018

Cache Analysis

In order to observe the relative performance of the cache, we will need to wget a variety of files that result in both cache hits and misses. For this portion of the analysis, the ‘urls’ file within the testing directory will be used.

While the hypothesized performance benefit of a cache is relatively significant, the difference between cache hit performance and cache miss performance can be nearly imperceptible when measuring worker time in milliseconds. As a result, we will be observing cache performance’s impact on worker efficiency at the microsecond scale. For testing purposes, the cache size has been set to 1 for a better mix of hits and misses.

Cache Hit vs Miss Time (urls; 100 1 0 1 1)						
	Run 1 (µs)	**Run 2 (µs)**	**Run 3 (µs)**	**Run 4 (µs)**	**Run 5 (µs)**	**Mean (µs)**
Total Time	20961	17921	20171	22627	20241	20384
Mean Miss Time	76	68	75	85	77	76
Mean Hit Time	4	4	3	4	4	4

It is immediately evident that retrieving a request from the cache is much faster than retrieving it from the disk. On average, the cache hit operations required only approximately 5% of the time that the cache miss operations did. The cache holds the requests in memory, resulting in fewer system calls and less latency.

The resultant hypothesis from this data is that—assuming there are some identical requests—the overall time to process the ‘urls’ requests will decrease and the proportion of cache hits will increase when the program is provided with a larger cache.

Because the input file is the same, the Hit : Miss ratio does not change between runs; rather, it only changes when the size of the cache changes in an impactful way relative to the input file.

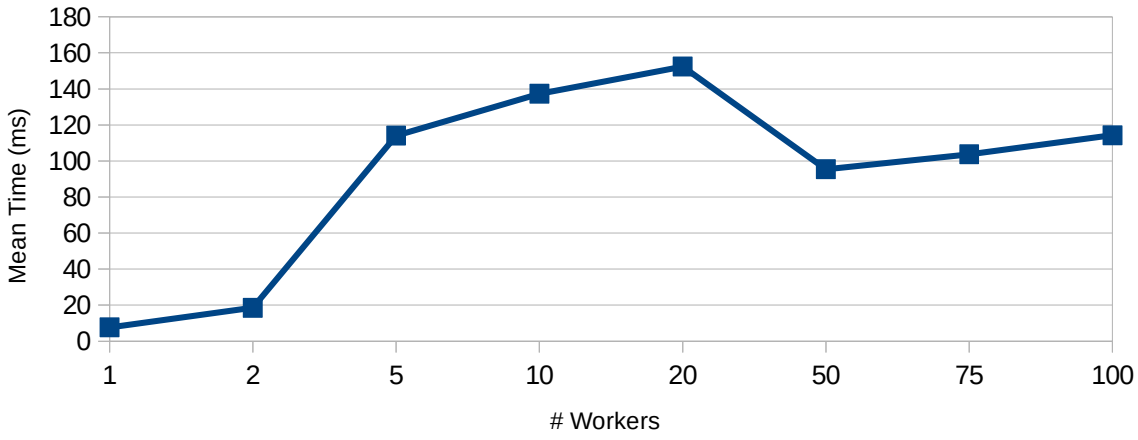
Small vs. Large Cache Performance		
	Small Cache [1]	**Large Cache [100]**
Overall Time (5 run mean)	20384µs	11480µs
Hit : Miss Ratio	0.58 : 1	2.51 : 1

As expected, increasing the size of the cache allowed for a greater ratio of hits to misses. Because the mean hit time is so much lower than the mean miss time, this hit proportion increase resulted in a large decrease in overall time.

‘bigurls’ Analysis

The ‘bigurls’ file contains requests for the same file over and over again. The expected behavior, then, is that the file will be cached upon the first request and then retrieved from the cache when subsequent requests are made. Having the data cached right away could take away a lot of the potential benefit from threads, making additional threads approach unnecessary overhead territory really quickly.

Mean Runtime vs. Number of Worker Threads (bigurls; 100 X 0 10 10)						
# Workers [X]	**Run 1 (ms)**	**Run 2 (ms)**	**Run 3 (ms)**	**Run 4 (ms)**	**Run 5 (ms)**	**Mean (ms)**
1	6.0	11.2	7.3	7.3	6.8	7.7
2	13.5	25.9	22.0	15.4	16.2	18.6
5	125.4	104.8	114.2	132.3	94.0	114.1
10	144.7	64.5	141.7	113.5	222.3	137.3
20	256.8	214.1	80.7	40.6	170.1	152.4
50	97.4	125.3	56.9	109.5	87.9	95.4
75	21.4	36.2	115.6	81.8	263.7	103.7
100	170.2	49.7	244.6	56.4	50.6	114.3



It can be seen that any multi-threading leads to loss of performance. This is likely because the overhead is greater than the benefit of concurrency. In the case of ‘bigurls’ especially, the cache is already pulling most of the weight in terms of delivering better performance. Switching between threads is only adding to overall time. The overall time sharply increases after two threads, leveling out at about 120ms with 5-100 threads. This plateauing is caused by too many additional threads to switch between, thus not adding any extra overhead. Variance increases dramatically with thread count, another indicator that the randomness of thread switching is heavily involved in the time outcomes.

Dynamic Thread Analysis

Due to the speed of the worker threads and the overhead of creating dispatcher threads, it is difficult to create circumstances under which dynamic threads can really shine. In this analysis, three dispatcher threads were used with one worker thread, a cache size of one, and a queue size of ten so that dispatchers are not bottle-necked by the queue.

Dynamic Worker Threads On vs. Off (xargs -n 2 -P 8; 'bigurls', 3 1 X 10 1)						
	Run 1 (ms)	**Run 2 (ms)**	**Run 3 (ms)**	**Run 4 (ms)**	**Run 5 (ms)**	**Mean (ms)**
Dynamic Threads On	6.4	7.2	7.2	7.6	7.3	7.1
Dynamic Threads Off	12.6	7.9	15.4	16.8	11.8	12.9

In this case, having dynamic worker threads enabled resulted in a 45% decrease in runtime. With three dispatcher threads and only one worker thread at initialization, the queue quickly fills, causing additional worker threads to be created. This additional thread or threads can then process the requests in the queue faster than the single worker threaded in a static execution.