Brian Quinn

Multicore Programming

Professor Mitchell

5 May 2017

Lab 4 Analysis

The purpose of this lab program is to analyze the differences between on-disk memory and cache storage using the previous multithreaded server implemented via a threadpool. In this implementation every insert will be written to disk. The same ThreadSafeKVStore is used in the server. However, an additional queue has been added to the data type in order to keep track of older elements. The values inserted will also have an additional count variable that will be incremented with any access to that variable. When a new item is inserted, the ‘oldest’ item will be popped from the queue. The associated count for this item will then check how many other times the item has been accessed and pushed to the queue. If this number is less than a certain percentage of total items in the queue, which is the percent usage of this item, then it will be removed from cache memory. Otherwise the count is decremented and the next item is popped and the process starts over. Different percentages will be tested in the test cases as well as being compared against complete on-disk storage.

Running HTTPerf provided the following server-side results for connection times:

Server Side	<1% removals	<5% removals	<10% removals	Disk-Only
Mean	23.672 min	23.258 min	23.704 min	23.767 min
Median	23.676 min	23.261 min	23.702 min	23.762 min
Maximum	23.683 min	23.266 min	23.723 min	23.781 min
Minimum	23.653 min	23.238 min	23.677 min	23.750 min

Table 1: Measured in duration time

Running HTTPerf provided the following client-side results for connection times:

Client Side	<1% removals	<5% removals	<10% removals	Disk-Only
Mean	23.671 min	23.251 min	23.699 min	23.764 min
Median	N/A	N/A	N/A	N/A
Maximum	23.682 min	23.262 min	23.722 min	23.781 min
Minimum	23.652 min	23.234 min	23.676 min	23.749 min
Total Duration	23.690 min	23.273 min	23.725 min	23.788 min

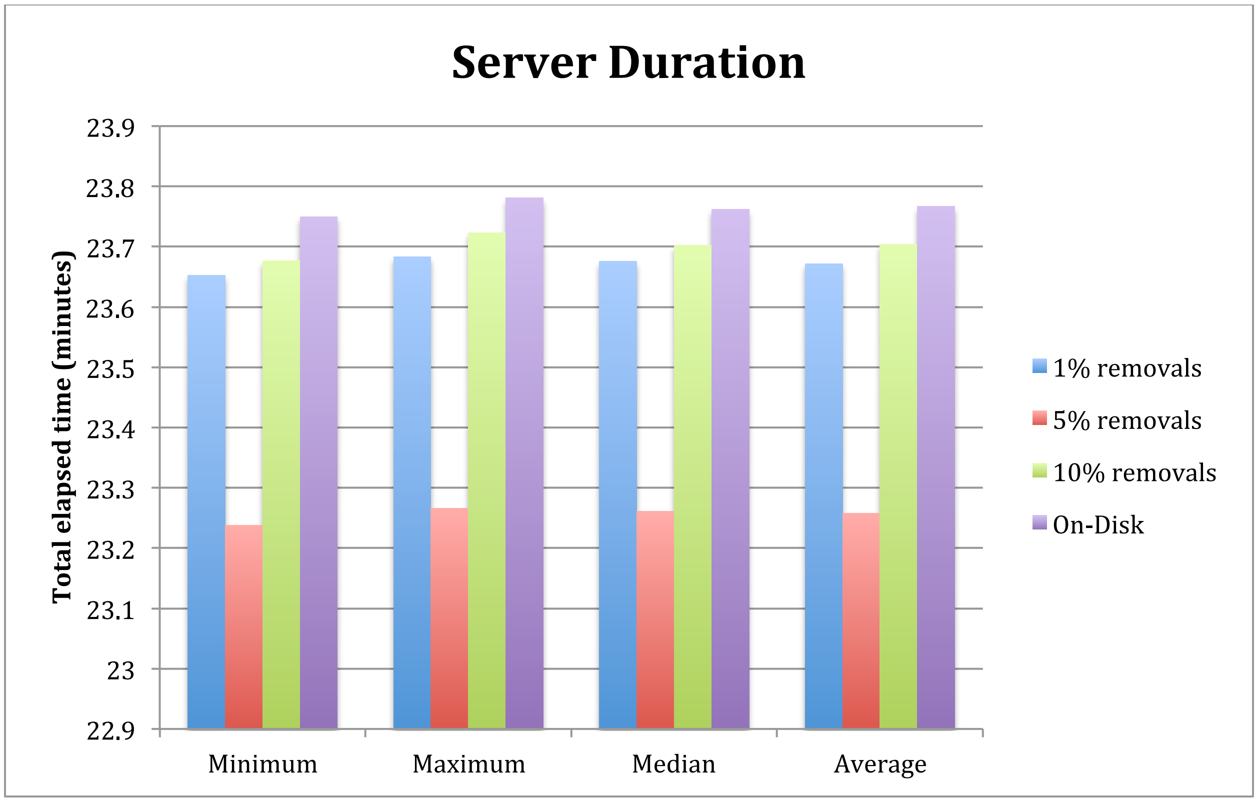
Table 2: Measured in duration time

The following server-side results for reply rates were recorded:

Client Side	<1% removals	<5% removals	<10% removals	Disk-Only
Mean	45.7 r/sec	46.6 r/sec	47.0 r/sec	45.5 r/sec
Maximum	47.5 r/sec	49.3 r/sec	45.7 r/sec	49.0 r/sec
Minimum	38.0 r/sec	42.8 r/sec	42.5 r/sec	33.8 r/sec
Range	9.5 r/sec	6.5 r/sec	3.2 r/sec	15.2 r/sec

Table 3: Measured in replies per second

The reply rate table (Table 3) shows that the Disk-Only memory had a much smaller minimum than the other three categories. This is due to the fact that the accesses to disk for every operation are going to take longer and thus result in a slower response. Alternatively, the range differences between 1% removals, 5% removals, and 10% removals are likely due to the nature of the eviction policy. 10% removals most likely has the lowest range because it is doing the most evictions as most items will not have more than 10% of the requests. 1% ranges the most because it has the strictest eviction policy and therefore, it will likely have to iterate multiple times to find a valid eviction candidate. However, if it does not it will have to iterate at worst *n* times to find the next candidate.



Graph 1

The above graph is derived from the server-side data in Table 1. The results provide a proof of concept in the efficiency of cache memory as all cache memory eviction policies ran faster than on-disk storage. However, there are also interesting observations to be made based on the differences in runtimes of the different eviction policies. Both 1% and 10% eviction policies ran significantly slower than the 5% cache policy. This is likely due to the test load and the most beneficial eviction policy will be dependent on the type of data being transmitted. In this case, it is likely that each of the ‘hot keys’ were inserted/looked-up a little over 5% of the time. Due to this the 5% policy would keep the keys in cache memory while finding others to evict. The 10% was likely less efficient because it would basically be evicting the oldest element even if it was recently referenced. As such, the cache memory was less likely to hold relevant keys and more disk accesses would have to be made. Alternatively, the 1% policy may make the least disk memory accesses, but would have to repeatedly pop from the tracking queue and decrement the corresponding element until it found/created and eligible eviction candidate.