(Joe Meyer hw6)

Because my networked cache doesn't work and the ostensibly working networked cache provided to me by other students does not seem to work when I run their tests, I benchmarked my non-networked cache instead (because that at least works).

Major bottlenecks in my code included getting an element size, which relied on iterating through the eviction vector to find the element, and inserting/deleting from eviction queue, which iterated through the vector eviction\_queue to ensure each item is in the queue only once on insert, and not in the queue on deletion (yes, these checks are redundant). Certain safeguards also wasted resources, verifying facts despite only ever being called when those facts hold (i.e. checking that arguments to functions are valid when that function is only ever called with valid arguments, as with the eviction function).

To remedy these inefficiencies, I changed the getsize function to maintain a hashtable mapping from keys to value sizes, so it no longer iterates over the entire eviction-queue. I also deleted the code which checks that each item is in the eviction queue exactly once on insert and not at all on deletion, and instead check that next\_evict is in the hashtable on eviction, looping the evictor until next\_evict is in the hashtable (this new code is in "set"). I also changed the evictor from largest-first to LRU, sparing the resources used to sort the eviction queue on insert. Finally, I deleted redundant safeguards.

My changes left most operation times relatively unchanged in my benchmark (perhaps because my tests all use tiny caches to make filling them easier so asymptotic improvements might be obscured by the tiny workloads). Non-overwriting sets seem to be slower in my revised cache (i.e. 1200 instead of 1050 nanosecs), and overwriting sets seem to be faster (i.e. 1430 instead of 1580 nanosecs). Although the latency of deleting and getting absent elements is unchanged, getting deleted elements and deleting present elements is significantly faster in my revised cache (1210 and 1250 nanoseconds, respectively) than in my original cache (1630 and 1600 nanoseconds, respectively).

Overall, I improved performance slightly for overwrites, deletes, and getting deleted elements at the expense of a slight delay in non-overwriting sets. Other operation times were not affected by my changes. The improvements likely stem primarily from the reduced runtimes of getting a cached element's size and lazy eviction queue maintenance. The extra set delay could be due to the extra cost of maintaining a hashtable of each element's size and updating it each set.

My throughput improved from .73 operations per millisecond to .88 operations. This is because the workload is mostly gets, and my revised cache takes the same amount of time to get a present element, and less time to get an absent element.

ex benchmark output:

vagrant@ubuntu:/vagrant/hw6$ g++ benchmark.cc cache.cc

vagrant@ubuntu:/vagrant/hw6$ ./a.out

Running test\_set\_insert() avg nanonanosecs per op: 1040.6 PASS

Running test\_set\_insert\_full() avg nanonanosecs per op: 1107.77 PASS

Running test\_set\_overwrite() avg nanonanosecs per op: 1581.65 PASS

Running test\_set\_overwrite\_dif\_size() avg nanonanosecs per op: 1566.6 PASS

Running test\_get\_present() avg nanonanosecs per op: 633.29 PASS

Running test\_get\_absent() avg nanonanosecs per op: 654.22 PASS

Running test\_get\_deleted() avg nanonanosecs per op: 1626.09 PASS

Running test\_delete() avg nanonanosecs per op: 1597.34 PASS

Running test\_delete\_absent() avg nanonanosecs per op: 658.95 PASS

Running test\_space\_used\_empty() avg nanonanosecs per op: 474.84 PASS

Running test\_space\_used\_full() avg nanonanosecs per op: 502.77 PASS

Running superscript avg operations per millisec: .73

vagrant@ubuntu:/vagrant/hw6$ g++ benchmark.cc efficient\_cache.cc

vagrant@ubuntu:/vagrant/hw6$ ./a.out

Running test\_set\_insert() avg nanonanosecs per op: 1209.25 PASS

Running test\_set\_insert\_full() avg nanonanosecs per op: 1280.83 PASS

Running test\_set\_overwrite() avg nanonanosecs per op: 1427.99 PASS

Running test\_set\_overwrite\_dif\_size() avg nanonanosecs per op: 1432.12 PASS

Running test\_get\_present() avg nanonanosecs per op: 613.55 PASS

Running test\_get\_absent() avg nanonanosecs per op: 642.83 PASS

Running test\_get\_deleted() avg nanonanosecs per op: 1214.54 PASS

Running test\_delete() avg nanonanosecs per op: 1247.01 PASS

Running test\_delete\_absent() avg nanonanosecs per op: 666.71 PASS

Running test\_space\_used\_empty() avg nanonanosecs per op: 471.75 PASS

Running test\_space\_used\_full() avg nanonanosecs per op: 466.91 PASS

Running superscript avg operations per millisec: .88