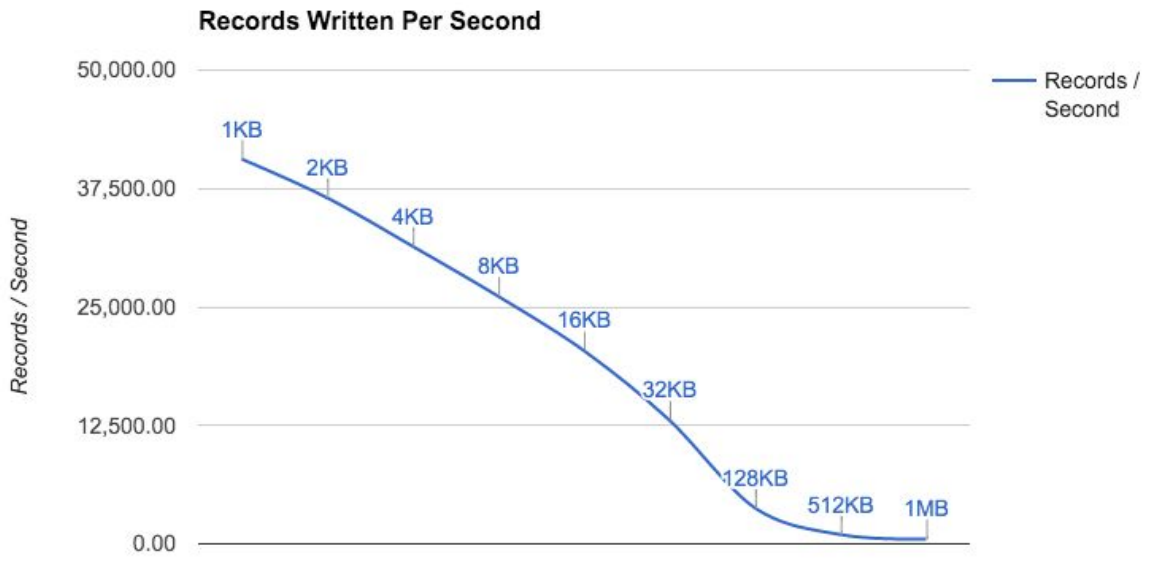
CSC443 Assignment 2

Data Layout

November 18, 2015

Binuri Walpitagamage - 999141780
Ammar Javed - 999058273

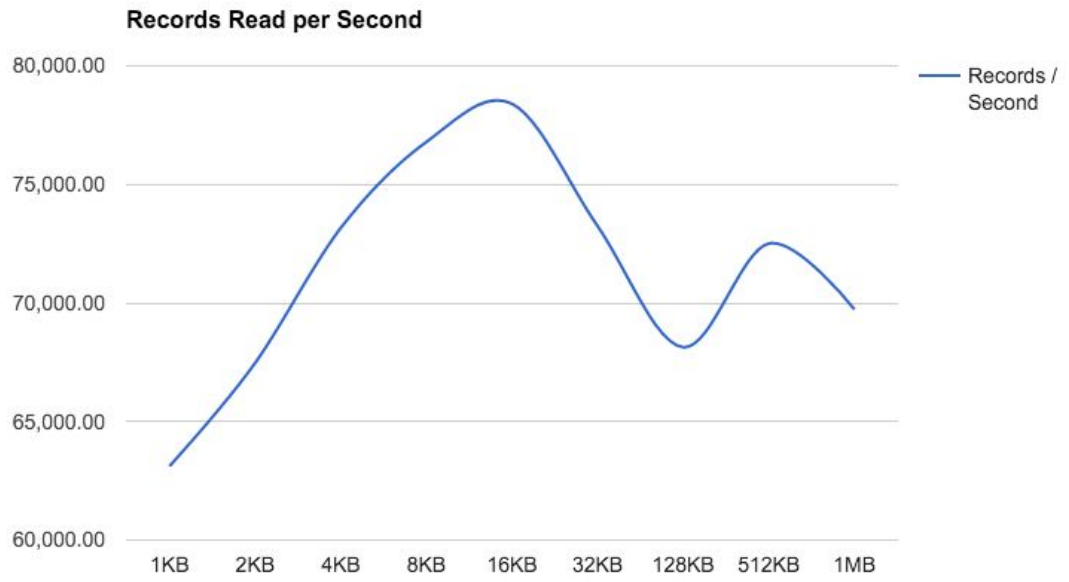
Section 3: Page Layout



Graph 3.1

This graph depicts the number of records written per second using different block sizes. We took the average time of 3 runs to ensure consistency.

We started timing as soon as the csv_file file pointer was open to read from and the page_file pointer was open to write to. The code path timed included reading in individual records from the CSV file, initializing a page and filling it with that record, plus as many others which would fit, and once full, writing it to the disk.



Graph 3.2

This graph depicts the number of records read per second using different block sizes. We took the average time of 3 runs to ensure consistency.

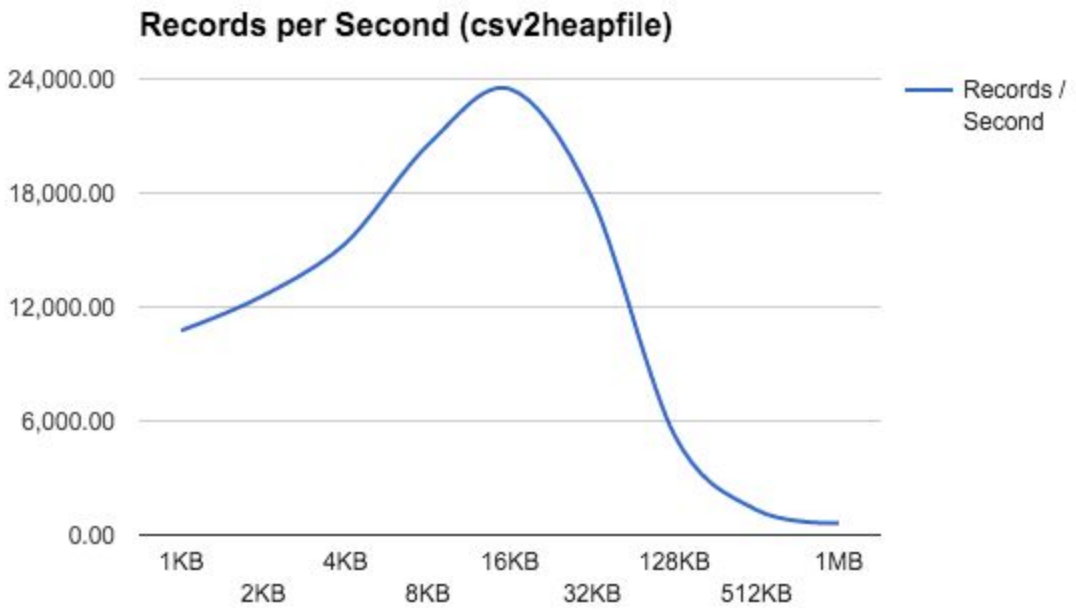
The code path we timed for reading included reading in the metadata for the first page (to bring back consistent pages as they were updated), calculating the page data from the meta data, and reading in the amount of records we had stored in that page. We did not include the time it took to dump the pages to the screen, as we were only interested in how long it took to read the pages and records.

Our experiments confirm that the page based format of handling data is superior to the CSV format, especially in the desk writing. Since we know that writing to the disk is the most expensive operation, as per assignment one, we can see how smaller block sizes being written to the disk would perform much faster than the larger block sizes, which tend to emulate the entire CSV file being written out of memory and onto the disk. Many chunks of small write operations are very fast compared to a few large write operations; and Graph 3.1 confirms that you can see that; as the block sizes get larger, the write efficiency decreases rapidly.

Graph 3.2 confirms the same is true for reading the records; there is a peak we hit with certain block sizes at which we can read in a large amount of records in a second, however as we start going through to the page sizes, you see performance dropping again. The read is much faster than the write, however multiple small calls to it are still faster than fewer longer operations (to fill up the larger block).

Some of the shortcomings of organizing pages on disk this way are still that without meta data, you are still forced to scan through the entire page block by block to try and see where there might be room to insert a new record into, or find a certain RID. We took steps to relieve that pain a bit by also writing out the page meta data, so we can read less data and decide to skip the page block on disk or not, however you may still end up iterating over and reading almost the pages before finding the one you need. This is the pain the heapfile addresses.

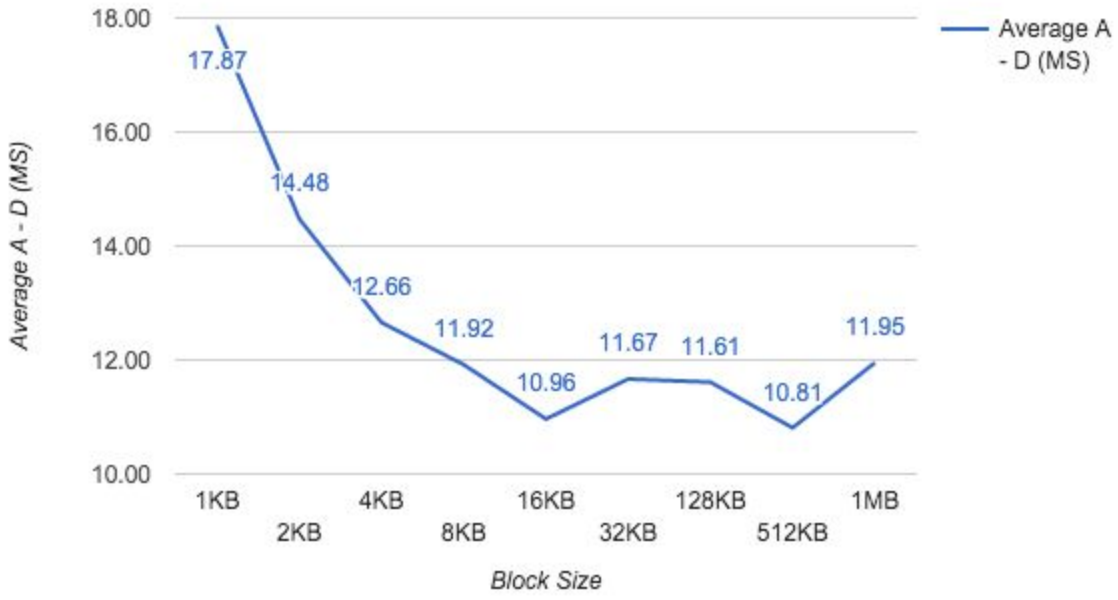
Section 4: Heap File



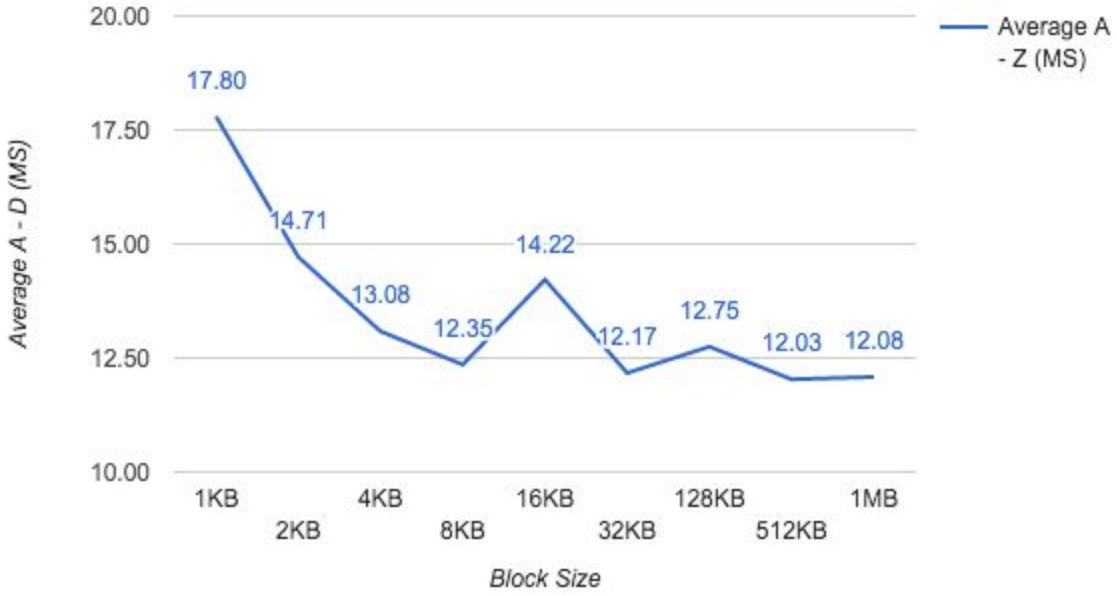
Graph 4.1
The records per second throughput of the csv2heapfile function using multiple different page_sizes

It was interesting to see a curve which resembled both the read and write to disk operations we tested earlier. As expected for writing, the records per second dropped significantly as we got to the larger page_sizes. What's interesting to note however is that because we are doing significantly more reads, since we have to travel back and forth in the file to update the heap directories and write out both the data and [additional] directory pages, we notice a peak in our performance around the same page_sizes as we saw in Graph 3.2 . It seems like anything between 8KB to 32KB is a sweet spot for reading from the disk in terms of page_size.

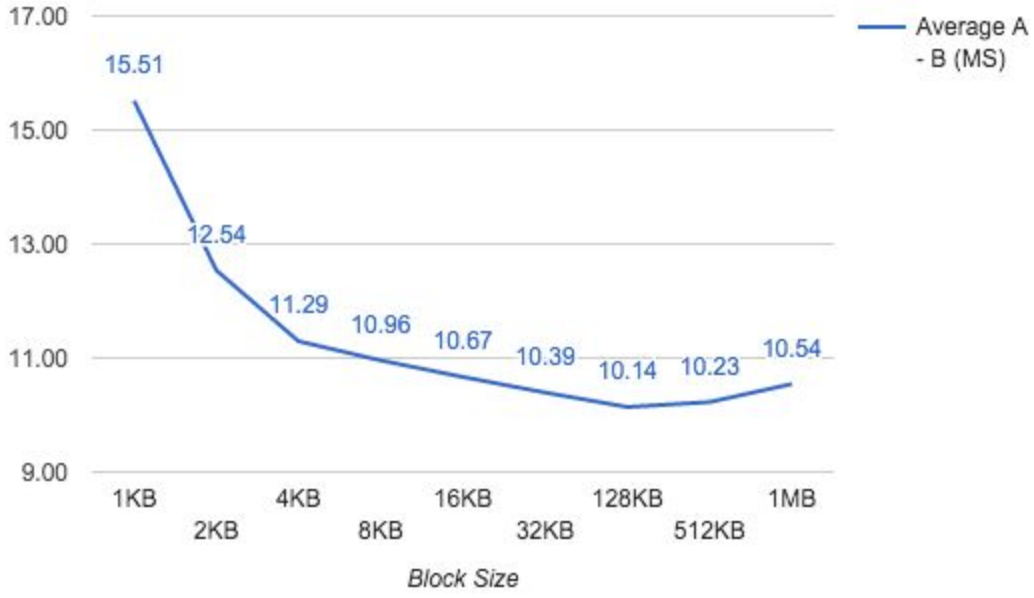
Select



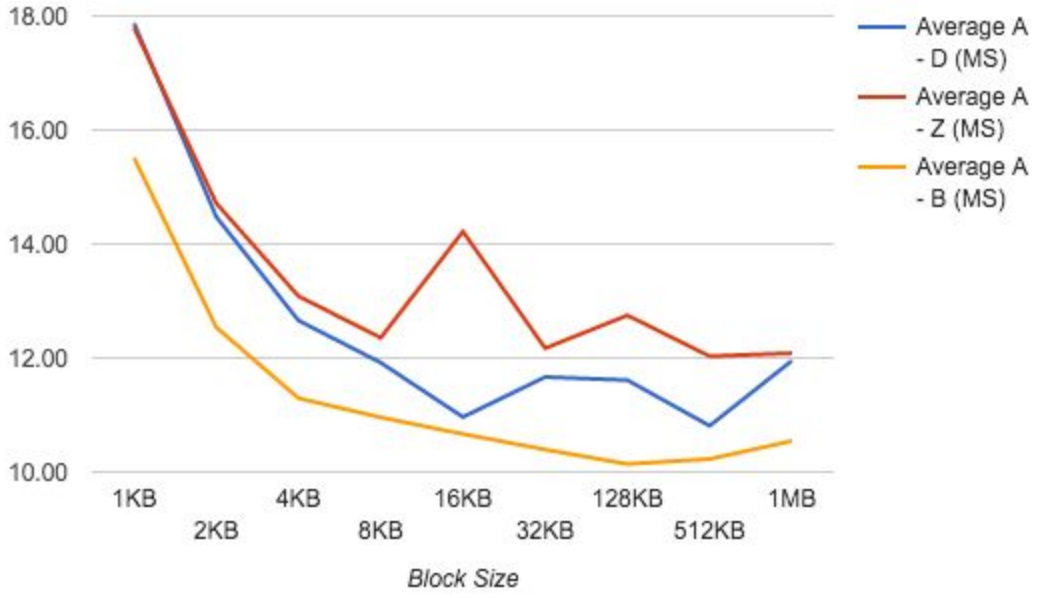
Graph 4.2
Start = A End = D



Graph 4.3
Start = A End = Z



Graph 4.4
Start = A End = B



Graph 4.5

The Graphs 4.2, 4.3, 4.4, 4.5 measure the runtime of the select query in milliseconds. A couple of things we noticed were that the run times of the query got increasingly faster as your page sizes got bigger. Looking at Graph 4.5 however, which shows the runtimes of different block sizes along with different Start and End criteria, you see that the more ‘correct’ attributes you had to output, the slower your query ran, because of the increase in the number of calculations you had to do. So the lower your Start-End range was, the faster your query ran, which only makes sense.

This reading of the pages however is not similar to the Graph 4.1 which showed us slowing down with larger page sizes however because now with each page we read in, the more computations you have to do, and then another disk access for the next page. In the larger sizes however, you are reading in a larger chunk of the table to do your in-memory computations (such as string comparisons and printing) and making fewer disk access operations. The overhead of disk access is clearly visible in Graph 4.5 as you see how much longer the smaller buffer sizes take to do the same reads with some added in-memory computation in between.