CSC443 A2

Jinman Zhao

Yitian Ding

Minjia Chen

We test on a 20000 records data size. (test2w.csv)

2.3

Since each record has 100 attributes, and each attributes contains 10 bytes. Then each record requires exactly 1000 bytes to serialize.

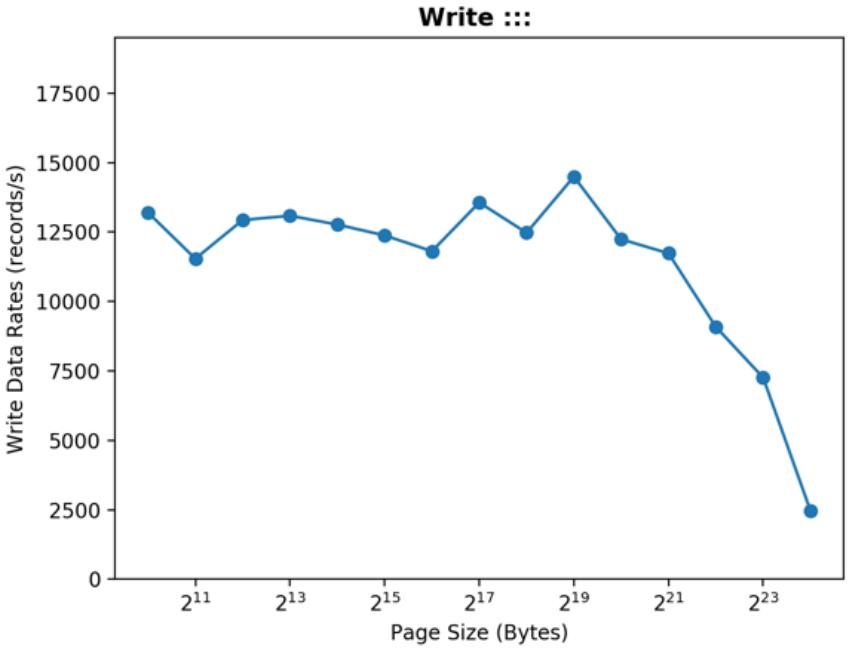
We initialized a record with 100 attributes, where each attribute contains 10.

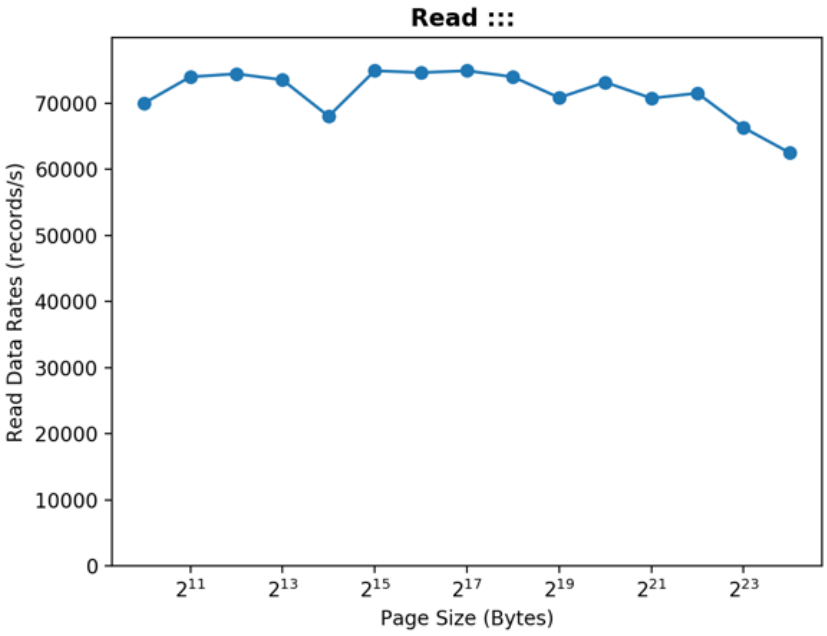
We use the following fixed_len_sizeof method to calculate the size. The result obtained was 1000, which verifies our calculation.

```
int fixed_len_sizeof(Record *record){
    return record->size()*10;
}
```

3.2

read/write





From the graphs, we find that the reading performance (using pagefile) is much faster than the writing performance (using csv).

Thus, storing records in Page format instead of CSV format is superior. We can read the entire page as well as the metadata easily and quickly. While it would cost much more if we parse through CSV deliminators and filter them line by line to get what we want.

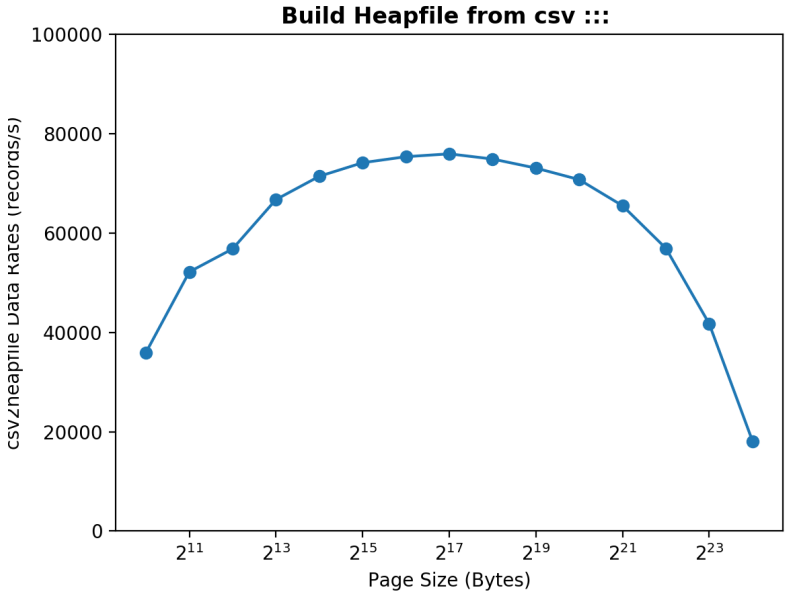
Page format also takes less space, since storing data as binary requires less storage than storing data as delimited strings.

Shortcomings of the way we organize pages:

Now we can't but scan the entire page block by block and try to see if there is an empty slots to insert new records. We could have stored more metadata such as bit-map of all the slots in order to make selection of records easier. Additionally, to make traversing pages easier, we can set the pointer to the next page in the page's metadata if we store multiple pages in the same block.

4.3

csv2heapfile

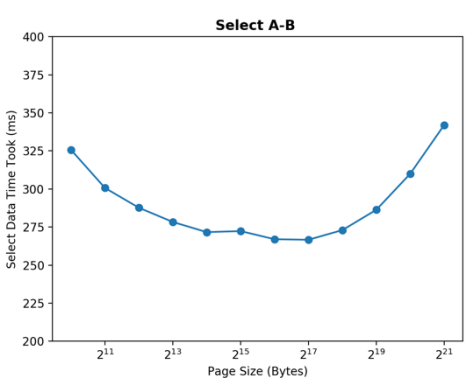


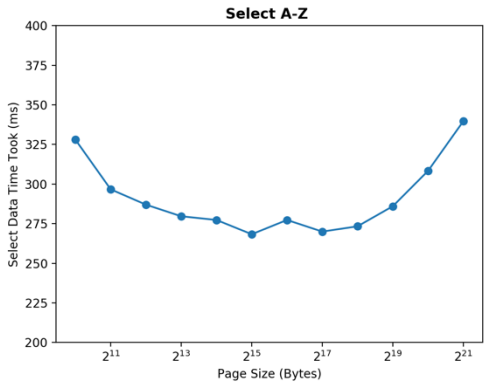
For csv2heapfile, the performance seems to be a fairly smooth concave-down curve. It dramatically increases at first as the page size grows from 2^10 = 1024 bytes, then slightly. At about Page size equals 2^16, the performance does not go up anymore and drops slightly, then significantly after about page size = 2^20 as page size grows.

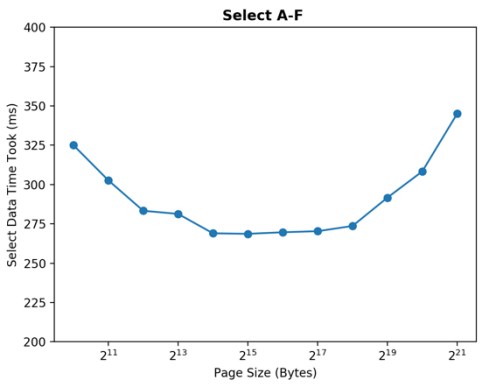
The reason of the concave performance is mainly because of the writing performance and reading times. When page size is very small, it has to frequently travel back and forth to read between the file and the disk, making the rates slow. Thus, time for csv2heap will decrease when page size gets larger(for 1kb to 128kb). Besides, because we use for loop to find empty space in page, write this page to disk when page is full. So if page size is too big, the time to look for empty space in page will increase the total time. This is why total time will not keep decreasing while page size increase. Total time will increase when it reaches the peak(in our case,peak is 128kb). In other words, records write to disk per second will increase while page size increase, and then it will reach maximum, then decline. When the page size is too large, traveling time is much less while writing to disk shows a dramatically decreasing efficiency, also causing to a huge drop in performance.

The plot supports that a page size between 16K to 128K might be the ideal interval for combined work of reading and writing to disk.

Select



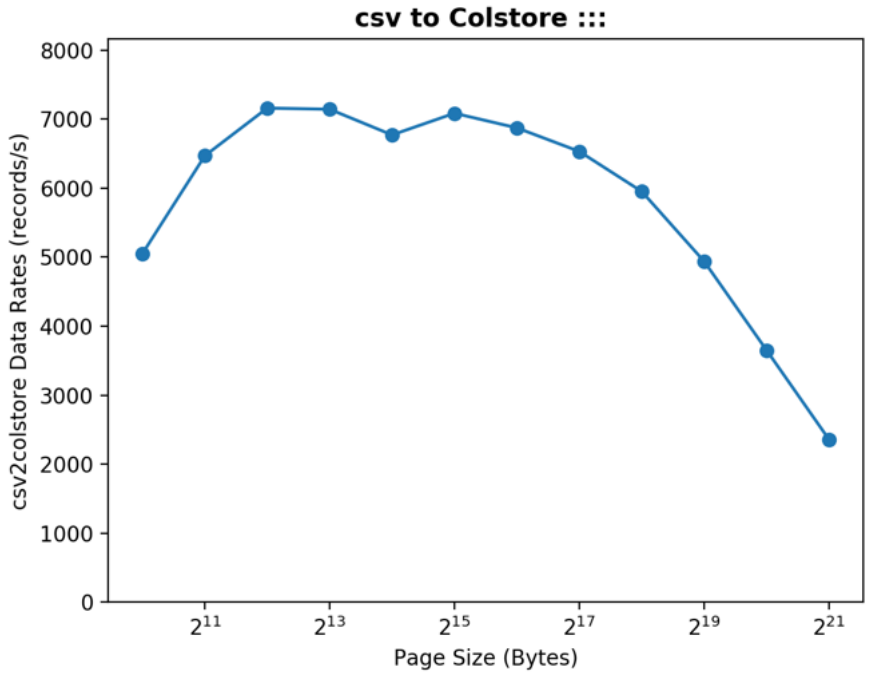




From select plots result, we find that there might be a slight difference between querying A-B, A-F, A-Z in the same heapfile. Smaller range query may lead to a faster operation since we need to take more time to compute if the query range is longer. The difference is very slight because we have to scan the whole heapfile no matter what range we want to query according to the poor schedule.

5.2

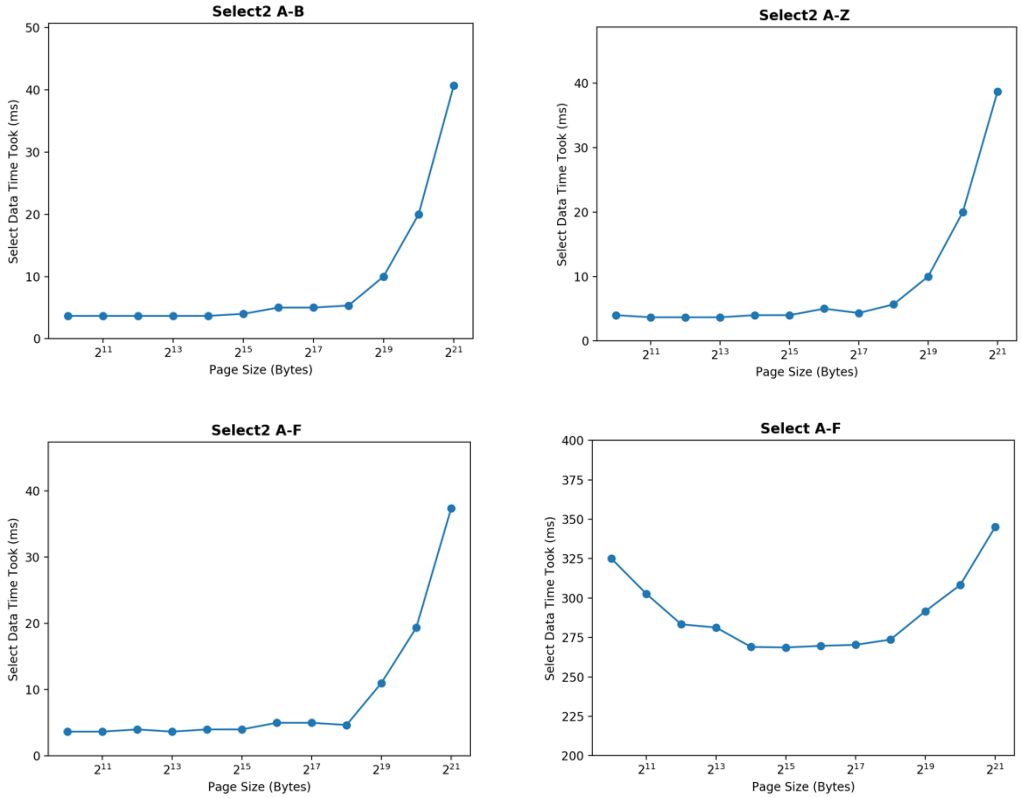
csv2colstore



The performance trend of csv2colstore is similar to that of csv2heapfile. After a obvious increase in rates at about 2^10 page size, we can see a stable performance between 2^12 to 2^15 for page size. Then the performance goes down as the page size grows. But notice that csv2colstore's rates are slower than csv2heapfile since we need to create many more heapfiles.

Same reason for causing this as csv2heapfile since we mainly do reading and writing work when experiment on csv2colstore.

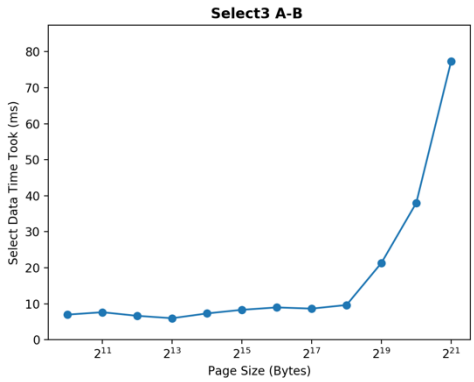
Select2 vs Select

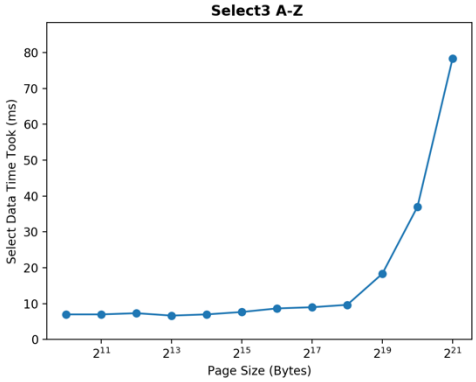


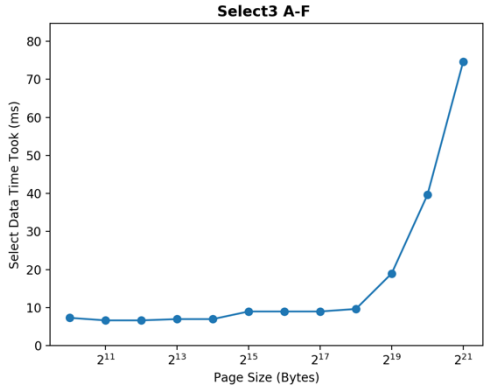
No obvious runtime difference about vaious query range for select2.

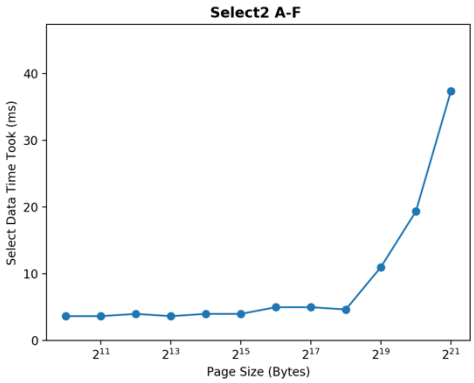
By observing the lower 2 pics (select2 vs select), we find that select2 took much less time than select when the page size is normal (between 2^10 to 2^14). For example, select takes about 300 ms while select2 only takes about 5 ms when page size is 2^11. We conclude that querying the colstore would be much more efficient than querying the heapfile. This is because we don't need to read through the other 99 attributes, for select2, we only need to read 1 attribute file from disk, this size of this file is only 1/100 of the file we need to read for select. Thus, select2 can save lots of time.

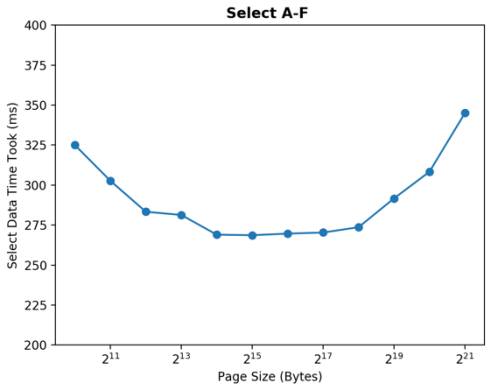
Select3 VS Select /Select2











The performance tread of select3 is very similar to that of select2 and much faster than select for the same reason as select2. By looking at the 2 middle plots (Select3A-F vs Select2A-F), there is a slightly more time used for select3, since select3 need to read 2 attribute' files (attribute and attribute_to_return) from disk and select2 only need to read 1 file.

Different query range has the similar runtime for select3, which is similar to the results in select2.

The difference of A-B, A-F,A-Z for select3 is very slight because we have to scan the whole heapfile for 2 columns no matter what range we want.