Report

Yuhui Pan
Raymond Li

2.3

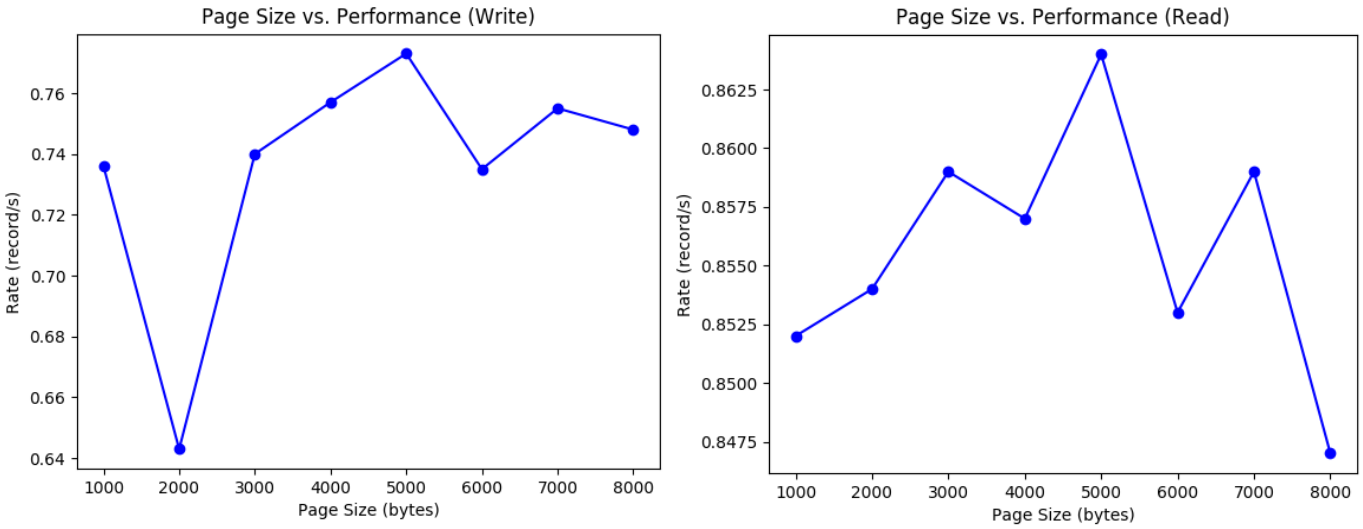
Since each record has 100 attributes, and each attributes contains 10 bytes. Then each record requires exactly 1000 bytes to serialize. We don't need to serialize the null terminator since each the record and attribute length are all fixed.

I initialized a record with 100 attributes, where each attribute contains 10 characters, using the following code.

```
Record record;
for (int i = 0; i < 100; i++){
   char attribute[11] = "1234567890";
   V v = attribute;
   record.push_back(v);
}
std::cout << fixed_len_sizeof(&record) << std::endl;
```

The result obtained was 1000, which verifies my calculation.

3.2



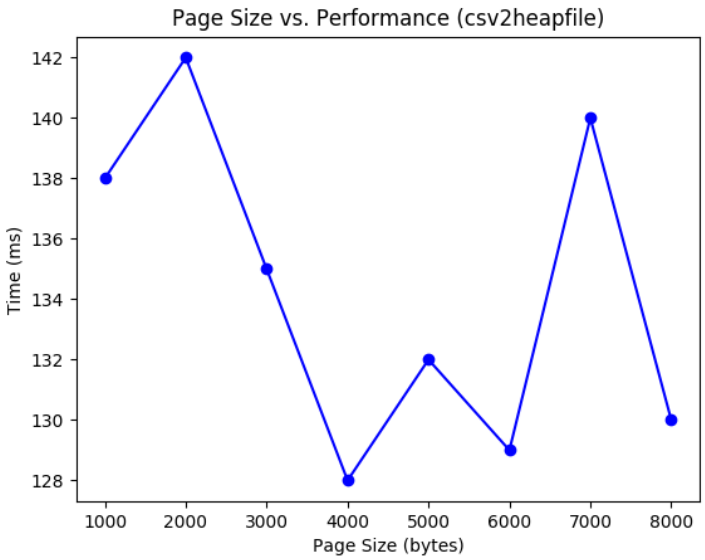
Storing records in Page format instead of CSV format is superior because we can read the entire page as well as the metadata, without having to parse through CSV deliminators to get the records we want. It's also more space efficient, since storing data as binary requires less storage than storing data as delimited strings.

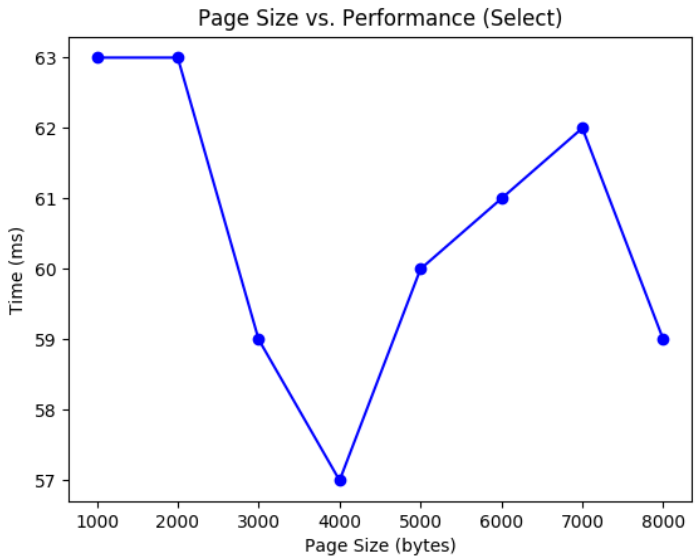
Shortcomings:
We could have stored more metadata, for example bit-map of all the slots, to make selection of records easier. In addition, if we stored multiple pages in the same location, we can store the pointer to the next page in the page's metadata to make traversing pages easier for selection queries.

4.3

Performance:
Since we implemented page-oriented I/O, we achieved slightly better performance when the page size was a multiple of 4K, which is almost equal to Linux system default size of 4096 bytes.

The performance is illustrated by the plots below.

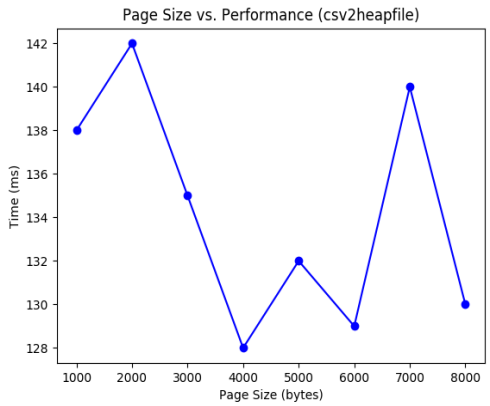


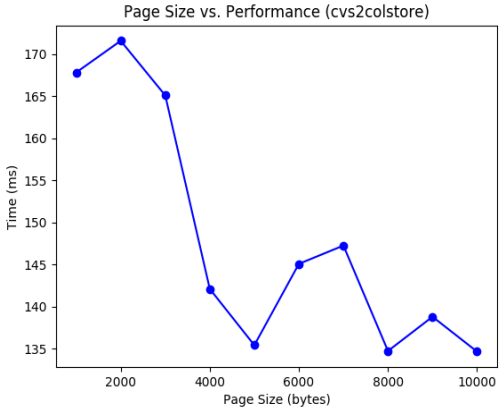


Since our heapfiles consists of no metadata, a 4K page consists exactly 4000 bytes of data. Therefore, it makes sense that page_size of 4000 has better performance (took less time).

The value range of END and START had no effect in the performance of select. Since heapfiles contains no index, we need to iterate through all records regardless of the value in the WHERE clause.

5.

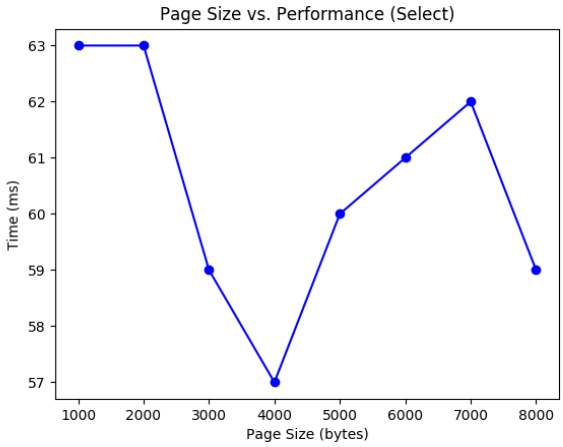


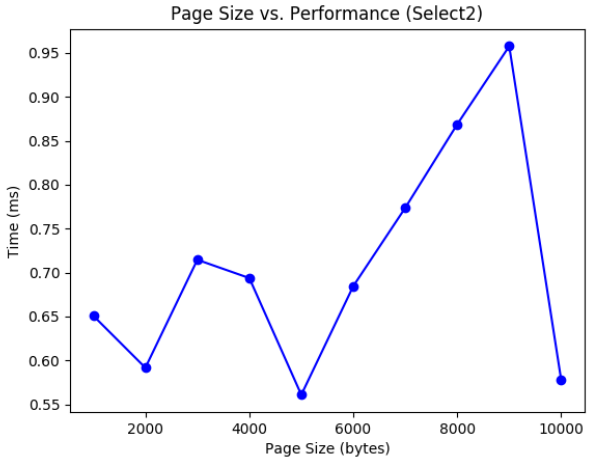


Comparison with cvs2heapfile

The performance is as above (left side is for csv2heapfile right side is cvs2colstore), since we are storing all CSV as heap file, it has same benefits as mentioned in Section 4. The same reasoning from Section 4 applies here. Column based does not take extra space. It is just another way of storing the CSV data. The graph reaches its best performance around 4K and 8K which is exactly as expected. The slight difference between them is the process time since in order to get each attribute value and store them in one single heap file, it would take extra time to get them in one run and push them into the file instead of just sequential storing all the data as csv2heapfile does.

Comparison with select2

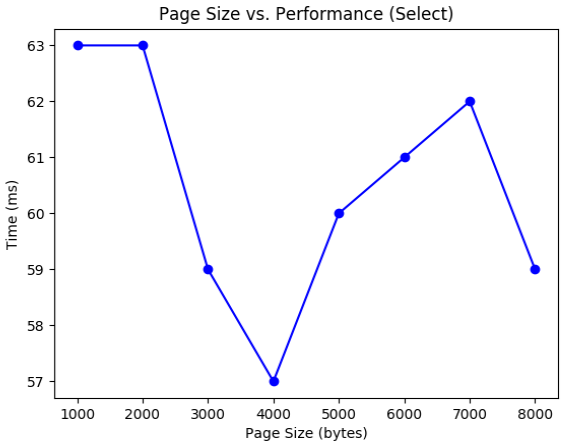


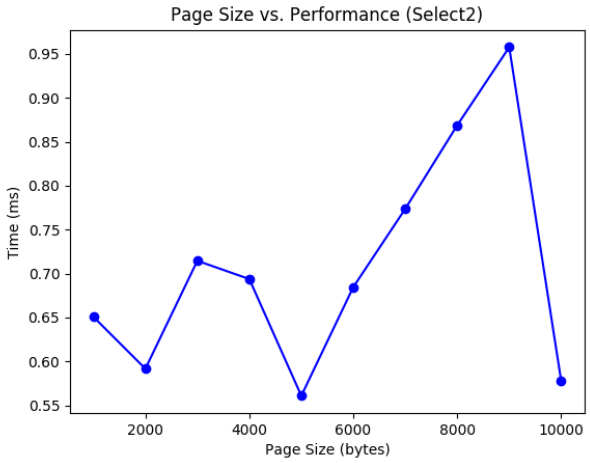


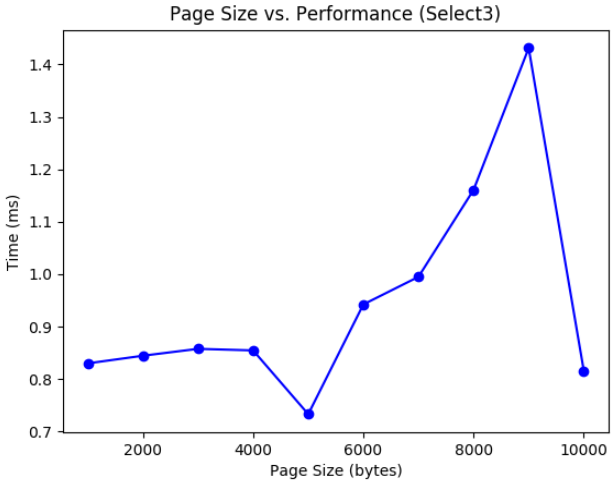
Result is as above (left side is for select and right side is select2). The difference between select and select2 is that select2 takes much less time than select to process the data. This is mainly due to the way how we access the data. We specifically choose one attribute for both comparison and projection which enable the program to only manipulate in one specific file corresponding to that row. This is significant since I/O time would have a lot influence on the performance of a program. Here, by the way we select

the data and the fact that the records are stored as column based, we gain a lot improvement over simply storing them as normal heap file.

Comparison with select3







Result is as above (top left side is for select, top right side is select2 and bottom is select3). Same reasoning applies here when comparing select3 and select. But select3 does something more than select2 which unavoidably reduces the performance by a little bit. Select3 is trying to compare one attribute and projecting on the other attributes. This leads to one extra I/O for opening the other heap file and reading data from it. Thus, increase the time because one more I/O is needed here. But still takes much less time than select since select would traverse through all files in order to get the correct result.

Not plotted here, we also noticed that, with select3, we experimented with different start and end values. The result would remain comparatively stable while select fluctuate a lot. The way how these two kinds of files are stored drives this kinds of behavior.