Project I - Task 3

Yawen Hou

March 24, 2019

Notes

- All the execution time are measured in milliseconds (ms).
- For Vector-at-a-time execution model, the vector size is fixed to 10 000.
- For PAX data layout, the number of tuples per page is fixed to 100.
- Big datasets are used to measure the performance of each query.
- All the joins are implemented using the Hash-Join algorithm.
- Each combination of the data layout and the execution model produces the exact same results for the same query.
- For each query, selections are performed as early as possible.
- *Store creation time* measures the execution time of Init(), the time for creating the Store and for loading the data from file.
- *Execution time* measures the execution time of the query, from the beginning of the scan till the fetch of each query result (i.e. until the last operator returns EOF).
- The measurements were conducted on a Windows 10 System using Eclipse, with Intel Core i7 @ 2.30GHz CPU and 16GB of RAM.

1 Selection and Projection

```
SELECT L.L_PARTKEY, L.L_LINESTATUS, L_RECEIPTDATE
FROM lineitem L
WHERE L.L_QUANTITY >= 20
```

1.1 Result

Data Layout	NSM	PAX	DSM		
Execution Model	Tuple		Column (Early)	Column (Late)	Vector
Execution time	122	163	326	**119**	511
Store creation time	6623	4679	4409	8041	8624

1.2 Analysis

The combination of **DSM data layout using Column-at-a-time late-materialization** performs the best, since most of the attributes are of a fixed length data type (except the attributes with string) and the values of a single attribute is stored contiguously in memory. When making a Select, only the corresponding column as a whole is called out, which implies a much higher I/O performance and cache exploitation than the Tuple-at-a-time execution model. Even though the vector-at-a-time is implemented in a similar way (in the sense that it uses DBColumns instead of DBTuples), it has to make multiple next() calls in order to finish filtering the corresponding columns, which increases the workload. At last, the late-materialization works better than the early-materialization, because every attribute value is reduced to a Integer representation, which takes much less space. We only materialize in this case the one column on which we need to perform filter.

2 Join

```
SELECT L.L_TAX, O.O_COMMENT
FROM orders O, lineitem L
WHERE O.O_ORDERKEY = L.L_ORDERKEY
AND O.O_SHIPPRIORITY = 0
```

2.1 Result

Data Layout	NSM	PAX	DSM		
Execution Model	Tuple		Column (Early)	Column (Late)	Vector
Execution time	3278	2956	654	**286**	4358
Store creation time	3125	2308	5413	5126	6153

2.2 Analysis

Again, the combination of **DSM data layout using Column-at-a-time late-materialization** outperforms all the other for the same reason cited as above: high I/O performance and efficient cache utilization, by loading Integers (small data type) indices instead of the large data values, and by materializing only at the necessary moment. We can especially observe the efficiency of the column-at-a-time execution model versus the other two models: column-at-a-time outperforms by almost 10 times tuple-at-a-time model, and by roughly 15 times vector-at-a-time model. We can see that since column vector makes less functions calls (multiple next() vs 1 execute() call) and it brings data store contiguously, it takes much less time to construct the data store and to execute the query than the tuple-at-a-time model. The performance comparison between vector-at-a-time and column-at-a-time depends on the vector size processed at each next() call, so we cannot conclude anything decisive on the number (15x), but column-at-a-time certainly outperforms vector-at-a-time for the reasons explained in 1.2.

3 Aggregation

```
SELECT MIN(L.L_DISCOUNT)
FROM lineitem L
WHERE L.L_QUANTITY < 30
```

3.1 Result

Data Layout	NSM	PAX	DSM		
Execution Model	Tuple		Column (Early)	Column (Late)	Vector
Execution time	228	383	371	**121**	392
Store creation time	3092	2193	2427	6981	5980

3.2 Analysis

Again, the combination of **DSM data layout using Column-at-a-time late-materialization** outperforms all the other models, for the same reasons stated above. Here we can observe that without a join, the tuple-at-a-time and vector-at-a-time execution time decreases significantly, and the differences between the different combination of data lyout and execution model are small. At the aggregation step, each combination has to materialize and go through each value of the selected attribute, which explains why this small difference.

4 Additional Analysis

4.1 Column-at-a-time: Early vs Late Materialization

For every query, we can observe that the late materialization execution model takes roughly half of the time taken by the early materialization execution model for the query execution time. For the reason mentioned above (better I/O efficiency with indices and materialize as late as possible), we can see that there is indeed a cost to bring all the data early in the stage of the query.

4.2 Tuple-at-a-time: NSM vs PAX

Although NSM is implemented using DBTuple and PAX is implemented using DBColumn, we can see that there is not much difference in the query times between the two data layout. Indeed, the PAX query time varies depending on the numbers of tuples stored by page, but since they both bring one tuple at a time, they have to make equal amount of next() calls, and we can see that this is much less efficient that the column-at-a-time execution model.

4.3 Vector-at-a-time: Variation in vector size

		Vector size			
		100	1000	10 000	100 000
Selection, Project	**Execution time**	3938	626	511	563
	Store creation time	5911	8317	8624	8343
Join	**Execution time**	3895	3549	4358	20690
	Store creation time	5695	5440	6153	5720
Aggregation	**Execution time**	482	506	392	443
	Store creation time	5534	4750	5980	5699

We can see that as the vector size increases, the time for selection and projection decrease, due to less calling to next() and more filtering per next() call; but the execution time for join increases drastically. It takes 5 times more times to perform join between 10 000 tuples at a time than between 100 000 tuples at a time. It is possible that the garbage collector takes more time freeing up the space for next tuples, but it is also possible that this time difference is due to the implementation. I could not find a clear explanation to this question. Finally, we can see that the aggregation time does not vary much, since at the end, we achieve to the same (small) number of tuples that the operator needs to go through.

4.4 PAX: Variation in page size

		Number of tuples per page			
		10	100	1000	10 000
Selection, Project	**Execution time**	199	163	159	161
	Store creation time	10461	4679	8440	8875
Join	**Execution time**	5061	2956	537	582
	Store creation time	4287	2308	4460	4514
Aggregation	**Execution time**	465	383	441	445
	Store creation time	3835	2193	3445	3180

We can see that as the number of tuples per page augments, the execution time for join decreases and stabilizes around 550 ms. The execution times for Selection, Projection and Aggregation do not vary much. Another interesting fact to notice is that generally, the store creation time decreases as we progress through each query; this is due to the fact that the data are already loaded in cache after the first query (Selection, Project), thus it is much faster for the 2nd and the 3rd query to reconstruct the same store each time.