Morales Gonzalez Mikael 234747 18.03.2018

Execute queries over different execution models and data layouts

These are the four types of queries that I executed on the pairs of data layout-execution model, these correspond to the four types of queries given in the project statement.

1.
   ```
   SELECT L.orderkey, L.partkey
   FROM lineitem_big L
   WHERE L.orderkey > 100;
   ```

2.
   ```
   SELECT L.orderkey, L.partkey
   FROM lineitem_big L, orders_big O
   WHERE  L.orderkey = O.orderkey;
   ```

3.
   ```
   SELECT MAX(L.orderkey)
   FROM lineitem_big L
   WHERE L.orderkey > 3;
   ```

4.
   ```
   SELECT COUNT(L.orderkey)
   FROM lineitem_big L, orders_big O
   WHERE L.orderkey = O.orderkey;
   ```

The execution time of these queries are given as an annex, and all the queries have been performed on the **big** datasets that were given.

Analysis of the queries:

• Query 1:
In my implementation, the first query is faster in the NSM data-layout with tuple-at-a-time execution model.
In theory this type of query should be executed faster in a DSM data-layout, since we are filtering on a specific attribut, and then merging only two attributs using the record ids. If we were to project all the attributs of the table at the end, this would be costly using DSM, since we would have to look for the records ids in all the attributs.
But given that we are performing early materialization, after filtering the attribut on the given operation, we are constructing a new table with all the columns, which require more time.
NSM can directly access the other attributs to reconstruct the new table much faster than DSM in this case.

• Query 2:
In my implementation, the second query is faster in the NSM/PAX data-layout with tuple-at-a-time execution model, these two data layouts perform the query roughly at the same speed.
In theory this type of query should be executed faster in a DSM data-layout, since we are joining two table by comparing two attributs, and then recovering the other attributs given the result of the join (using the record ids).
But again, given that we perform early materialization, the join create new filtered table with all attributs of the table `lineitem` and `orders` concatenated. This add a lot of computation time and overwhelm the initial advantage of DSM when joining two attributs.

• Query 3:
In my implementation, the third query is faster in the NSM/PAX data-layout with tuple-at-a-time execution model, these two data layouts perform the query roughly at the same speed.
In theory this is the typical type of query where a DSM data-layout should outperform NSM/PAX, given that we are filtering and aggregating the same column, we don't need to read any other attributs of the table. But again, since we are performing early materialization, filtering the attribut produce a new table which contain all the attributs, this cancel entirely the performance of DSM in this case.
Hence NSM/PAX perform better since no intermediate table is constructed.

- Query 4:

In my implementation, the fourth query is faster using PAX data-layout. Like previous answers, in theory, this query would perform better in a DSM data-layout since the aggregation is done on the same column where the join is performed. But again, early materialization constraint us to create an entire intermediate table with all the attributs of `lineitem` and `orders` concatenated. This add too much time complexity, hence this is why NSM/PAX perform better.

In general, it is worth to notice that these times fluctuate between every execution. A more rigorous benchmark system would be necessary to really estimate the performance of the models.
Another important factor is that data-layouts, like PAX, don't perform that well given that its suppose to be cache friendly, which should decrease the time to recover the attributs of a tuple, but given that this project is written in JAVA and that the dataset are in memory as opposed to the disk, we can't see that much the advantages of PAX given that we are not reading pages from the disk, hence we are not using as much as we could the mini-pages and the cache locality that they provide.
Furthermore, we can clearly see the negative impact of early materialization on a NSM data-layout, and the execution times would vary significantly using late materialization.

Annex

Query 1:

```
SELECT L.orderkey, L.partkey
FROM lineitem_big L
WHERE L.orderkey > 100;
```

Query 2:

```
SELECT L.orderkey, L.partkey
FROM lineitem_big L, orders_big O
WHERE  L.orderkey = O.orderkey;
```

Query 3:

```
SELECT MAX(L.orderkey)
FROM lineitem_big L
WHERE L.orderkey > 3;
```

Query 4:

```
SELECT COUNT(L.orderkey)
FROM lineitem_big L, orders_big O
WHERE L.orderkey = O.orderkey;
```

- NSM: Tuple at a time:
  - Query 1: 60 ms
  - Query 2: 171 ms
  - Query 3: 74 ms
  - Query 4: 1002 ms

- PAX: Tuple at a time (with 400 tuples per page)
  - Query 1: 169 ms
  - Query 2: 210 ms
  - Query 3: 92 ms
  - Query 4: 871 ms

- DSM: Column at a time
  - Query 1: 286 ms
  - Query 2: 2692 ms
  - Query 3: 517 ms
  - Query 4: 2605 ms

- DSM: Vector at a time (with a vector size of 400)
  - Query 1: 69 ms
  - Query 2: 370 ms
  - Query 3: 411 ms
  - Query 4: 3506 ms

Note: These execution times can be seen by running the Main class of my project.