CS422 - Database Systems - Task 3

Baris Sevilmis
Sciper Number: 306798

1 Overview

In database project 1, aim is to implement various database execution models with different data layout combinations. In each execution model, there are mainly 6 classes to complete: Scan, Filter, Project, Aggregate, Join & Sort. Testing of each model-data combination are done by provided tests. Purpose of this report is to explain briefly and precisely each of these combinations according to their implementation and execution times. Table 1 depicts execution time results for each model and system properties for test environment are provided below Table 1. These results will be discussed over the next section. For the convenience, given results are average of 10 distinct executions.

Table 1: Execution & Data Models vs. Execution Times

METHODS	*Execution Times*
Volcano(Row_Store)	16s 498ms
Volcano(Column_Store)	16s 419ms
Volcano(PAX_Store)	14s 112ms
Overall_Volcano	47s 029ms
OperatorAtATime(Row_Store)	14s 185ms
OperatorAtATime(Column_Store)	13s 948ms
OperatorAtATime(PAX_Store)	12s 821ms
Overall_OperatorAtATime	40s 954ms
BlockAtATime(Row_Store)	15s 162ms
BlockAtATime(Column_Store)	15s 228ms
BlockAtATime(PAX_Store)	13s 702ms
Overall_BlockAtATime	44s 092ms
LateOperatorAtATime(Row_Store)	16s 75ms
LateOperatorAtATime(Column_Store)	15s 174ms
LateOperatorAtATime(PAX_Store)	14s 327ms
Overall_LateOperatorAtATime	46s 251ms
Overall	178s 326ms

System Properties:

- OS & Version: macOS Catalina & 10.15.3
- Processor: 3,1 GHz Dual-Core Intel Core i5
- Memory: 8 GB 2133 MHz LPDDR3
- Graphics: Intel Iris Plus Graphics 650 1536 MB

2 Evaluation

In this section, Implementation & Execution Time details are going to discussed. Implementation details, approaches and corresponding positive and negative effects are to be discussed in Subsection 2.1. Actual affects of these implementation to execution times, methodology in addition to methodology-data layout comparison are to be discussed in Subsection 2.2 in terms of Table 1.

2.1 Implementation

Volcano is the first model in our implementation. In terms of implementation, there are multiple alternative ways to implement the model. As already known, *Volcano* model consists of 3 main methods for each data model: **open()**, **next()**, **close()**. Naive way to implement would be put workload on the next function, to be more specific do all the necessary functionalities as the **next()** is called. Better approach, followed in the implementation as well, would be to get rid of all necessary calculations in **open()** and just pass on the computed results one by one in **next()**. Although space complexity might be an possible issue in some cases, this approach utilizes on execution time and space optimization would be very complex implementation-wise resulting in almost no difference.
OperatorAtATime contains the method **execute()** only, and therefore, has no specific initilization. Main objective is to return whole resulting table at once instead of returning resulting rows or columns one by one.
BlockAtATime consists of **open()**, **next()**, **close()** as *Volcano* model as well. However, instead of returning results one by one, *BlockAtATime* returns results by blocks of specified sizes. Therefore, it could be seen as a mixture of *OperatorAtATime* & *Volcano*. In other words, block size being 1 implies *BlockAtATime* = *Volcano* and block size being equal to table size indicates *BlockAtATime* = *OperatorAtATime*. Naive way to implement would be again as in *Volcano* putting the workload on **next()**. Fortunately, better approach of precomputing given results in **open()** works here as well.
LateOperatorAtATime is almost same as *OperatorAtATime*, however uses late materialization instead of early materialization. It only consists of **execute()** method as well. Instead of sending whole table between functionalities, only virtual ids are sent. These virtual ids are later on materialized by lazy evaluators. Therefore, intra-query communication happens faster, although requirement for materialization, even though it is late materialization, increases intra-query communication amount and causes negative affects on execution results.
Lastly, data-layout implementations were optimized as well. Since the incoming data has certain format either one of the provided data layouts element by element iteration in case of naive implementation. Hardest to implement among them is the PAX format, and depending of the incoming data to Scan, it could even get harder to implement even the PAX format. Therefore, simple matrix tricks were utilized, namely transpose operation. Depending on the incoming data layout, data were saved into a specific storage table. Even if the stored data may be in wrong format in some cases, a simple matrix transpose suffices to reorder data into requested format.

2.2 Execution Time Comparison

Discussion will be as the following: Starting from overall model comparison to specific inter model - data layout comparison and lastly, intra data layout comparison for each model.
Considering the overall time for each model-data layout comparison 180s 2ms could be perceived as a reasonable amount of execution times for the total amount of 540 tests, to be more specific 135 tests for each model. For the comparison of overall model-data layout, Table 1 depicts similar results for each of models, only differing by seconds among each other. As said before, for each specific model-data layout combination 10 tests were made, and their averages were taken. Table 1 implies order of overall execution model from best to worst as the following: *OperatorAtATime*, *BlockAtATime*, *LateOperatorAtATime*, & *Volcano*. As mentioned earlier, results rely heavily upon the implementation, and in addition to the types of requested queries. Although, provided queries were same for each execution model, a different subset of queries may have resulted in another ordering. Regardless of the given facts, there are some additional reasons that could enlighten the set of result times.
As known, in *BlockAtATime* and *Volcano* models **open()** and **close()** are utilized only once, but function call for **next()** is utilized multiple times. Especially in *Volcano* model, **next()** is called for every resulting row/column. Large amount of function calls is certainly one of the reasons that *OperatorAtATime* has the best overall result, and *BlockAtATime* outperforms *Volcano* even though the utilization of similar implementation. However, *LateOperatorAtATime* does not stand out as *OperatorAtATime*, rather demonstrates almost *Volcano* like results. This could be explained by another following reason that is already mentioned in the Subsection 2.1. Although, using virtual ids improves the intra-query communication by just sending virtual ids instead of whole sending tables, there are additional function calls for lazy evaluators to materialize resulting table. Another reason would be the need for materialization even if it is late. Therefore, all the implemented classes do need to perform additional function calls, except Projection, for lazy evaluators and materialize the provided virtual ids. As a consequence, in addition to implementation type and test queries, *LateOperatorAtATime* performs not as well as *OperatorAtATime*.
Lastly, impact of data-layouts on models need to be discussed as well. It is obviously shown by the results, PAX data format causes the best execution time overall regardless of the execution model. Row and column store show promising results as well, in fact with different implementation strategy or different queries, especially Scan, they may prove better than PAX format. However, it would be unreasonable to compare row-store and column-store, because of their interchanging execution time results. As mentioned in Subsection2.1, both data layouts, including PAX store, were optimized using simple matrix operations, namely transpose and as a result, iterating over the tables in Scan was achieved with reduced numbers.