Project-1 Report

This report presents an overview on the implementation of Run-Length Encoding (RLE) and Execution Models as a part of the Project-1 for the Database Systems (CS-422) course at EPFL. This project builds upon the previous Project-0. It uses the same algorithms for the scan, project, filter, aggregate, join and sort operators with some modifications. Hence, this report will be used to highlight the major changes in the algorithms. The implementation is divided into three tasks:

Task 1: Run Length Encoding

Sub-Task 1.A:

Scan was implemented to read RLE columns and return de-compressed tuples. Similar to project-0 scan, each entry of the scannable object was fetched and stored in a data buffer in the open() block, and later passed on to the next() block. Except in this case, there is an additional loop to decompress the RLE entries based upon their length parameter.

Sub-Task 1.B:

First, RLE-Decode was implemented to decompress RLE-Tuples to Tuples using the above logic in the next() block. The RLE-Reconstruct synchronizes two streams of RLE-Tuples to produce a stream of overlapping RLE-Tuples. This was done in the next() block using an infinite loop and conditional return statements. Finally, the project-0 algorithms for project, filter, aggregate and join were modified to operate on RLE-Tuples directly. The modifications include the use of input.value instead of input, keeping track of the startVID, and updating the length using simple arithmetic operations.

Task 2: Query Optimization Rules

Sub-Task 2.A:

Three rules were implemented to pull decode above the filter, aggregate and join operators in order to improve efficiency. The implementations were quite simple and very similar to each other. Copies of the operators were created using .copy and the traits and inputs were passed to the copied blocks in the desired order of operations. The rule for join was a little tricky since it involved two decodes, but the same logic still applies.

Sub-Task 2.B:

The last rule to push filter into the inputs of reconstruct was implemented. Depending on the fields the filter references, it was pushed towards either the left or right child of reconstruct. A copy of the filter with its new condition was simply passed as input to the reconstruct block.

Task 3: Execution Models

Sub-Task 3.A:

The selection vectors were enabled for operator-at-a-time execution. Most of the code is exactly the same except for minor modifications to append selection vectors to the result. For scan, a sequence of ‘true’ selection vector is appended to the scanned columns. Project and filter employ the same algorithm as before except they only operate on the tuples with the ‘true’ selection vectors, and the ‘false’ ones are left untouched. Additionally, instead of removing the tuples, filter simply sets their selection vector to ‘false’. Aggregate, join and sort remove the ‘false’ input entries, process them in the exact same way as before, and return the result with ‘true’ selection vectors appended.

Sub-Task 3.B:

Columnar-at-a-time execution was implemented along with map-based functions which operate on entire inputs instead of tuples. The logic is almost the same as operator-at-a-time except few modifications to accommodate the desired formats. For scan, project and filter, the entire input is scanned, projected or filtered at once using the new functions. Once again, these operators ignore the entries with 'false' selection vectors and append the new selection vectors instead of removing tuples. Finally, the aggregate, join and sort operators use the exact same code with some additional lines to convert the input (using transpose) and output (using toHomogeneousColumn) to the correct format.

Additional Notes:

Some improvements were made in the implementations of operators from Project-0. Number of loops and accesses were reduced, and more efficient data types were used. The implementation of join was improved by using hash join instead of nested join. There is still room for improvement in the join implementation since the right input is still not accessed tuple-at-a-time. Nevertheless, all the test cases pass with good accuracy and speed, crossing 1.00 speedup over the baseline performance.

To conclude, the project has been completed successfully, it has passed all the tests (locally and online) and it is efficient enough to meet the baseline performance.

Submitted by: Kushagra Shah (316002)

Dated: 31/03/2021