# Data Layouts and Execution Models

Robin G.F. Mamie
*CS-422: Database Systems (Spring 2020)*
*École Polytechnique Fédérale de Lausanne, Switzerland*
2020–03–30

## I. Introduction

The task of this project is to implement three different data models on three different storage layouts. The models use early materialisation, except for one – namely ”operator at a time” – which also uses late materialisation. The timing of the different provided tests will be explored in this report.

Times were taken on a computer having 64GB of RAM and an 8 cores/16 threads processor overclocked at 5 GHz, which executes the tests about 2.7 times faster than on the continuous integration used to grade the project. All means and standard deviations were computed out of 20 different executions.

## II. Tests

All 540 provided tests successfully pass in 11.8s locally, or 8.0s without the warmup tests – meaning ca. 31s, respectively 21s on GitLab's continuous integration environment.

### A. *”Micro” tests*

All tests of this category run in well under 10ms for each model. There is no distinction between models, nor store types.

### B. *”TPCH” tests*

Each query runs well under 300ms on the described machine. The second slowest query is executed in less than 200ms, and the remaining queries in under 150ms. We can observe some interesting distinctions.

The volcano model on the row layout is particularly slow for query number 2 (see second graph). This is perfectly logical: this query operates on a significant amount of data dispersed over a lot of lines. Here, the number of calls to `next` is very big throughout the query, like the number of disk fetches. Column and PAX stores do not have this problem since they directly read the data either via columns or pages. Other models do not encounter such a number of function calls and disk operations and are very well designed to handle such a case.

The vectorized volcano model is a step in the right direction. Since disk fetches happen more closely to each other – we work in blocks of tuples –, we greatly reduce the amount of memory page faults, and possibly cache misses, we had with the normal volcano model.

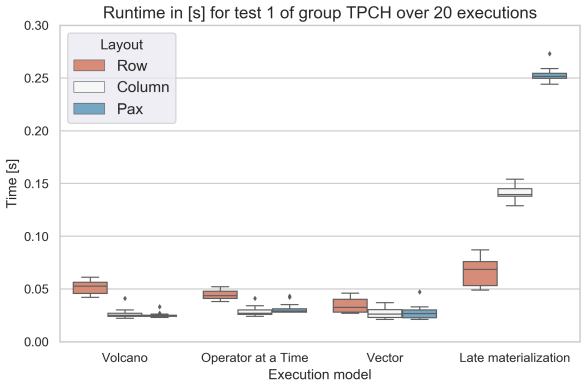
Finally, the operator at a time model takes this paradigm to its extreme. Every operator is executed at the same time. This is at a great cost memory-wise, since everything is materialised at once. However, since everything fits into memory, we do not see a significant difference with regards to the vectorised volcano model.

Most of the queries (model- and layout-wise) see similar times across the board. However, the late materialization model is consistently slower than its counterparts. This can be explained by the amount of overhead created by this method that in our situation just adds time compared to simpler models. Indeed, late materialization is well-suited for a query such as query number 2, but e.g. not for query number 1 which requires that almost all fields be materialized during the execution of the query. Since all datasets easily fit in memory, it makes sense that early materialisation is always faster than late materialisation – as empirically shown by our experiments. The abysmal difference between the three layouts in the late materialization can be explained by the fact that a VID represents a row. It seems logical that such a representation is better handled access-wise using a row layout. Column and PAX stores suffer from the fact that rows are not necessarily close in memory, which induces more time lost during data fetching.

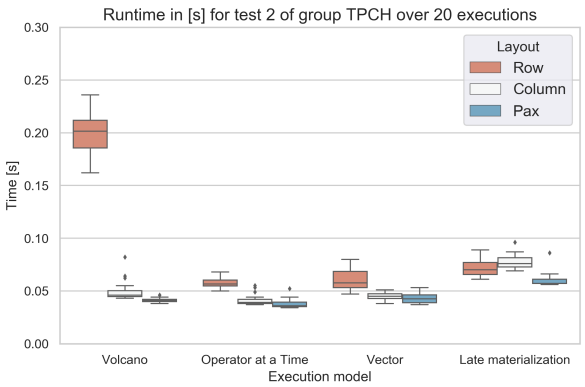
In summary, our three models operate on a spectrum from ”tuple at a time” to ”complete materialisation”, with a second dimension working on early and late materialisation. The former influences the number of tuples materialised at a certain point in the execution, while the latter is beneficial when memory is scarce for the implemented queries.

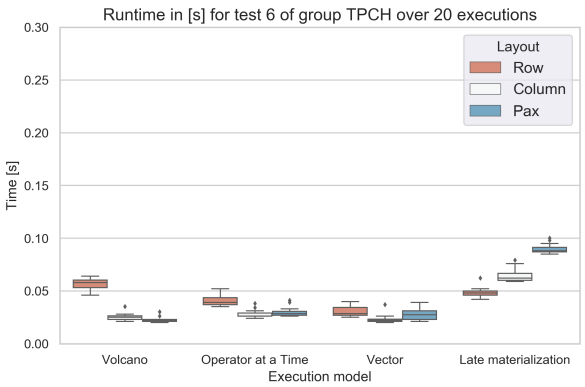
## Graphs

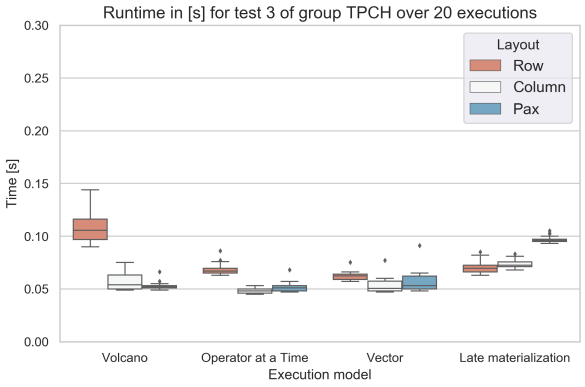
For completeness, here are all the graphs summarising the execution times for the TPCH queries [1].

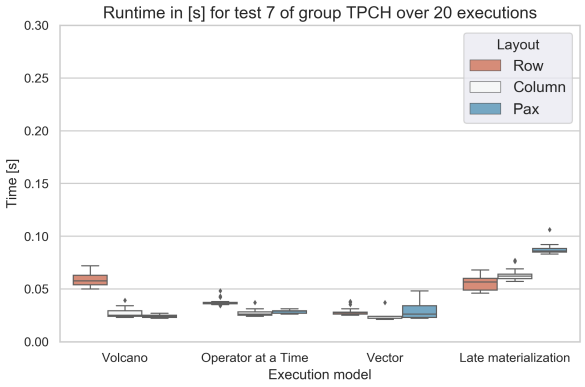


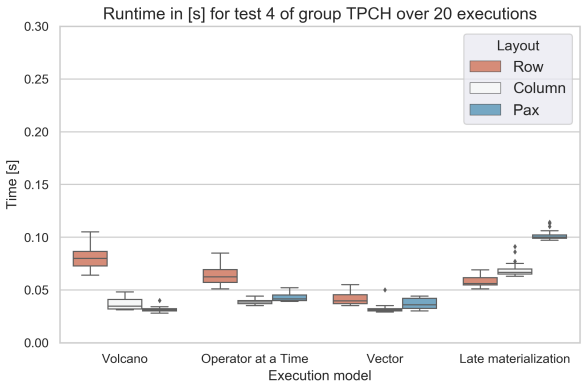

[1]The number of a query is not the one on the filename, it is simply its ordinal during the execution of the tests.

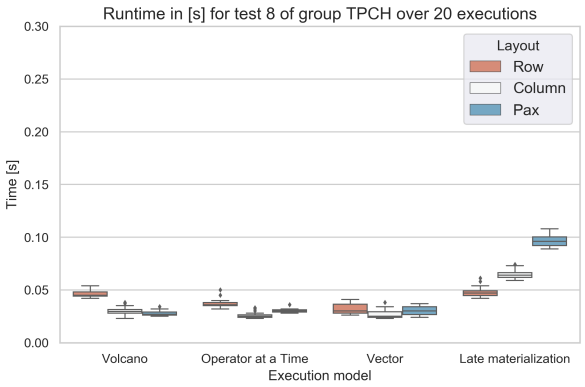

Runtime in [s] for test 2 of group TPCH over 20 executions
Time [s]
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Layout
Row
Column
Pax
Volcano
Operator at a Time
Vector
Late materialization
Execution model

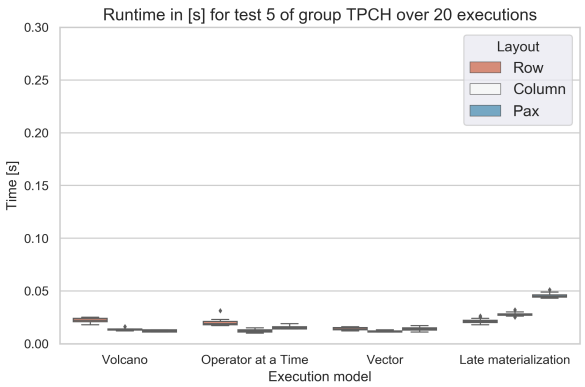

Runtime in [s] for test 6 of group TPCH over 20 executions
Time [s]
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Layout
Row
Column
Pax
Volcano
Operator at a Time
Vector
Late materialization
Execution model

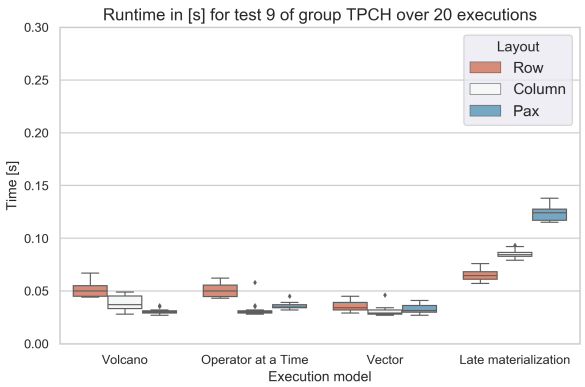

Runtime in [s] for test 3 of group TPCH over 20 executions
Time [s]
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Layout
Row
Column
Pax
Volcano
Operator at a Time
Vector
Late materialization
Execution model

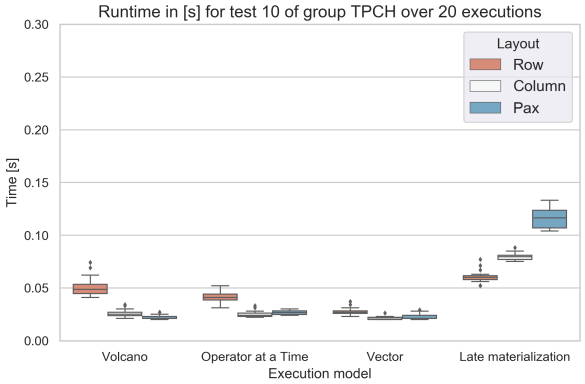

Runtime in [s] for test 7 of group TPCH over 20 executions
Time [s]
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Layout
Row
Column
Pax
Volcano
Operator at a Time
Vector
Late materialization
Execution model


Runtime in [s] for test 4 of group TPCH over 20 executions
Time [s]
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Layout
Row
Column
Pax
Volcano
Operator at a Time
Vector
Late materialization
Execution model


Runtime in [s] for test 8 of group TPCH over 20 executions
Time [s]
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Layout
Row
Column
Pax
Volcano
Operator at a Time
Vector
Late materialization
Execution model


Runtime in [s] for test 5 of group TPCH over 20 executions
Time [s]
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Layout
Row
Column
Pax
Volcano
Operator at a Time
Vector
Late materialization
Execution model


Runtime in [s] for test 9 of group TPCH over 20 executions
Time [s]
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Layout
Row
Column
Pax
Volcano
Operator at a Time
Vector
Late materialization
Execution model


Runtime in [s] for test 10 of group TPCH over 20 executions
Layout
Row
Column
Pax
Time [s]
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Volcano
Operator at a Time
Vector
Late materialization
Execution model