Efficient SQL Query Compilation with JIT Compilers

Saumya Verma, Sudip Chatterjee, Suprio Ray
Faculty of Computer Science, University of New Brunswick
{nivan, sverma3, sudip.chatterjee, sray}@unb.ca
Mark Stoodley, Calisto Zuzarte, Ian Finlay
IBM Canada
{mstoodle, calisto, finlay}@ca.ibm.com

ECLIPSE OMR JITBUILDER

Eclipse OMR JitBuilder, developed using Eclipse OMR project, simplifies the task for a runtime system to incorporate a JIT compiler. It is easy to implement and with just a few hundred lines of code, it can target more than one platform.

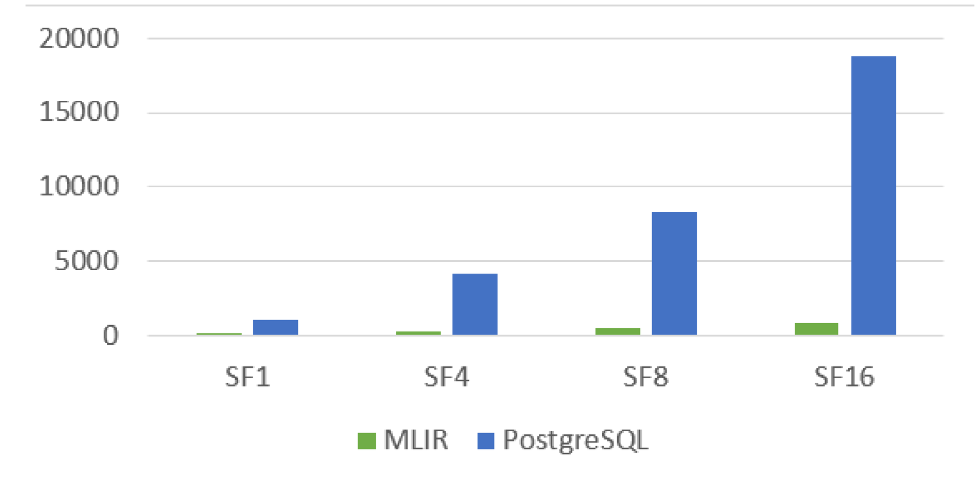
MULTI-LEVEL INTERMEDIATE REPRESENTATION (MLIR)

MLIR is a modular and extensible compiler infrastructure. It provides a common intermediate representation that can optimize code generation across different hardware targets. It supports custom optimizations, transformations and analyses.

PROBLEM STATEMENT

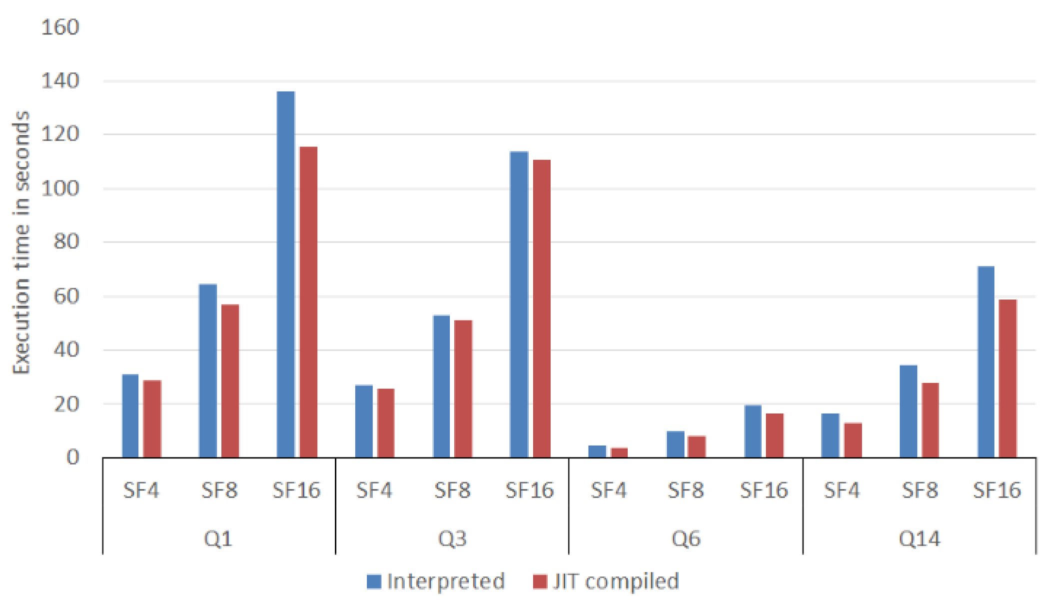
- Evaluation of SQL Expressions and tuple materialization consumes a significant portion of query execution time.
- With relational database systems following an interpretation based tuple-at-a-time model of volcano style query execution, this process becomes more inefficient and slow.
- Recent research emphasizes the need to compile the query plans, but the integration of these techniques within a database system requires re-architecting the query engine.

RESULTS



Code generation, compilation, and execution of TPCH Q6 using MLIR Vs PostgreSQL

Code generation (C++), compilation and execution using CasaDB



Execution time for interpreted vs. JIT compiled

CONTRIBUTIONS

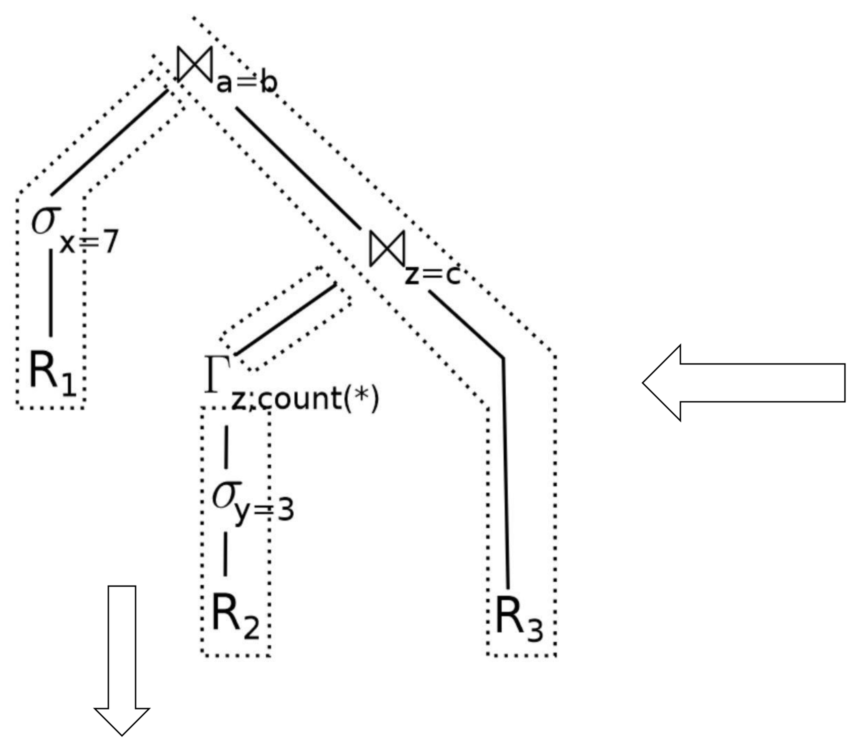
- Development of CasaDB based on in-memory database query compilation
- Conference Paper Published on query compilation
- Presented multiple posters
- Presented *Efficient Compilation Of SQL Queries* at CASCON x Evoke 2022
- Received best video award at CASCON x EVOKE 2022

GOALS

- Light-weight integration of Eclipse OMR JitBuilder into PostgresSQL-12.5 to JIT compiler operations
- A fast and multi-target backend generation technique using JIT compilers (Eclipse OMR and MLIR) supporting code generation for single node and distributed nodes.

APPROACHES

- Data-centric, run-time code generation model
- Use of pipelines and Intermediate Representation




```
select *
from R1,R3,
        (select R2.z,count(*)
        from R2
        where
        R2.y=3
        group by
        R2.z) R2
where
R1.x=7 and
R1.a=R3.b and
R2.z=R3.c
```

```
std::vector<MaterializedTupleRef_V7_0_1_2_3_4_5_6_7_8> V7;
arrow::Result<std::vector<MaterializedTupleRef_V7_0_1_2_3_4_5_6
_7_8>> res_V7
   =
cdb_arrow::arrow_read_MaterializedTupleRef_V7_0_1_2_3_4_5_6_7_8
("orders");
if(res_V7.ok()){V7 = res_V7.ValueOrDie();}
else {std::cerr << res_V7.status();}
std::cerr << "RETURNED from : orders" << std::endl;
std::cout << "RECORDS:V7 = " << V7.size() << std::endl;
tuplesFromAscii<MaterializedTupleRef_V7_0_1_2_3_4_5_6_7_8>("ord
ers");
auto end_2 = walltime();
auto runtime_2 = end_2 - start_2;
std::cout  << "pipeline 2: " << runtime_2 << " s" << std::endl;
std::cout  << "timestamp 2 end " << std::setprecision(15) <<
end_2 << std::endl;
auto start_4 = walltime();
std::cout  << "timestamp 4 start " << std::setprecision(15) <<
start_4 << std::endl;
std::vector<MaterializedTupleRef_V8_0_1_2_3_4_5_6_7> V8;
```

SQL query to Query Execution Plan (QEP) using pipeline

CAS–Atlantic

*Contributors : Stephen A. MacKay, DeVerne Jones, Nithin Ivan, Shubh Sharma and Debajyoti Datta