LLVM Query Runtime

VALKyrie

Arindam
Kaushik

Ladan
Vinayak

What we did so far

Scan operator

```
valkyrie> SELECT * FROM <table_name>;
```

Select operator

```
valkyrie> SELECT * FROM <table_name> WHERE <condition>;
```

This week

1. Projection Operator
2. Evaluation

Demo

Projection

Two approaches to implementing Projection,

1. Create a new tuple in the projection step and pass it to the parent operators
2. Maintain the same tuple format, map old tuple format to new expressions and evaluate these when necessary

We chose method 2 as we don’t need to allocate memory and break the pipeline

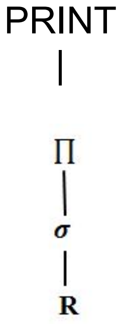
e.g. If PRINT is parent of PROJECT(A+B)

it gets evaluated as PRINT(A+B)

Projection

Intuitively:

```
for(each tuple in R){
     //Selection predicate
     if(predicate.getValue()){
          //Printing projection expressions
          for(each projected expression){
               print expression.getValue()
          }
     }
}
```



Evaluation

Limitations:

- Cannot run TPC-H Queries because Joins and Aggregates are not implemented
- Cannot evaluate some queries on 100MG dataset

Experimental setup:

Experiments were performed on Intel® Core™ i5-3337U CPU

- @ 1.80GHz × 4
- 12 GB of RAM

Queries

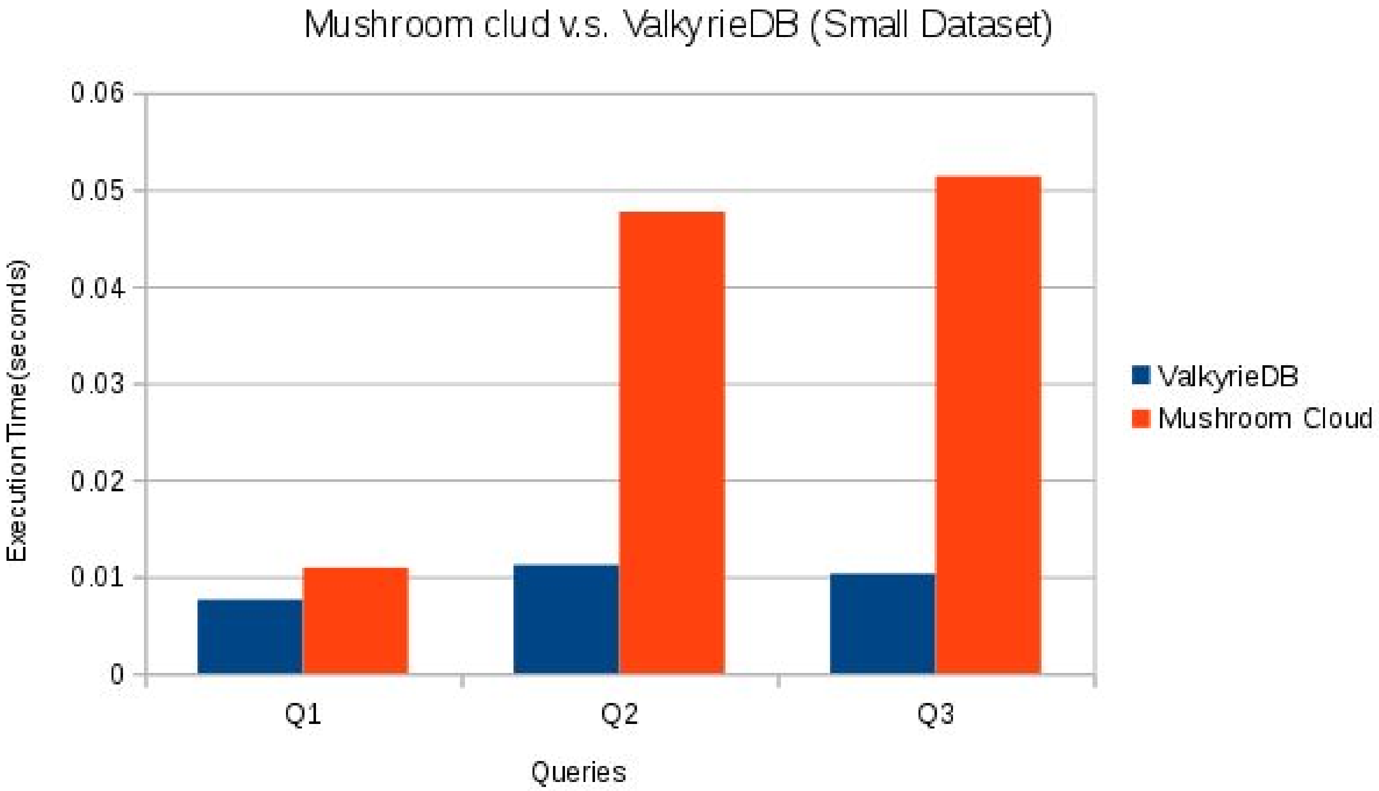
```
Q1  SELECT * FROM orders
Q2  SELECT * FROM lineitem WHERE orderkey > 300000
Q3  SELECT * FROM lineitem WHERE orderkey = 20000
```

Execution Time:

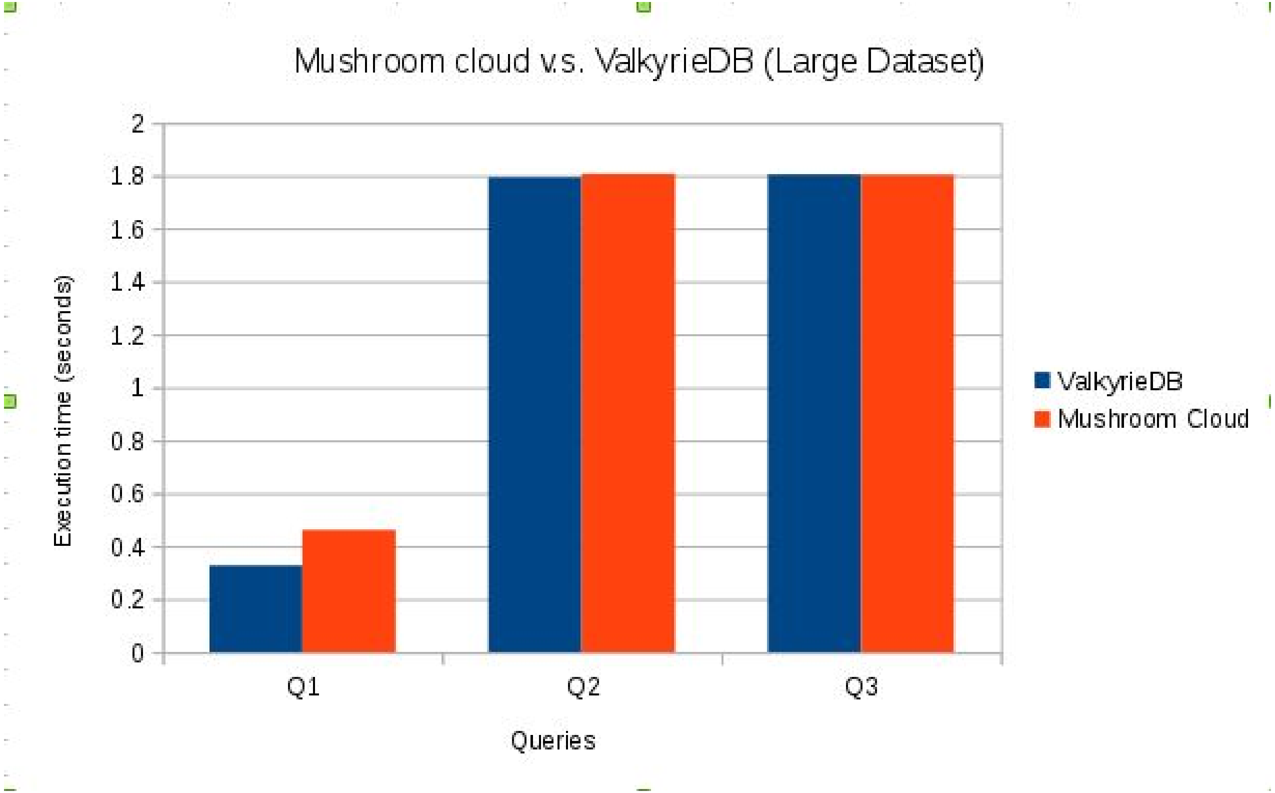
Mushroom Cloud: query plan parsing + generation + execution time

Valkyrie DB: generate the LLVM code + executing it

Execution time: **Mushroom cloud** vs **Valkyrie sf = 0.01**



Execution time: **Mushroom cloud** vs **Valkyrie sf = 0.1**



Discussion

- Comment out the call to printf function
    - Q1: 0.3278s with printf 0.6679s
    - Q2: 1.794s with printf 2.79061s
    - Q3: 1.8039s with printf 1.78264s
- Q3 has high selectivity, there are just two tuples in the result.

Next Steps

1. Join Operator
2. Transferring result data between pipelines