Final Presentation: Scheduler 2

Makoto, Mingkang, Aidan

May 2, 2024

Project Goals

75%

- Break down optimized query plans. ✅
- Dispatch fragments to enable correct execution. ✅
- Provide job status ✅

100%

- Able to abort/cancel a query ✅
- Facilitate both inter-query and intra-query parallelism. ✅
- Implement cost-based, dynamic scheduling. ✅

125%

- File-granularity Morsel-driven parallelism.
- NUMA-aware locality optimizations.
- Scheduler fault tolerance and scalability.
- Work Stealing.

Progress

- Expanded on integration testing
- Full integration of query graph parser and pipelined execution
- More intelligent scheduling policy (Task FIFO -> Stride)
- Developed profiling and benchmarking tools
- Testing on TPC-H queries
- Removed global scheduler lock in favor of lock-free data structures and finer-grained locking

Architectural Components: API and Internals

Item 1: Architectural Components – Internal

- Fundamental Internal Components and Interfaces
    - DAG Parser
        - Serialization/Deserialization ✅
        - Parsing DAG into pipelined stages ✅
            - Identify operators that are designated pipeline breakers (JOIN, LIMIT, …) ✅
            - Split plan and replace with operators with `PlaceholderExec` containing metadata with pointer to intermediate data ✅

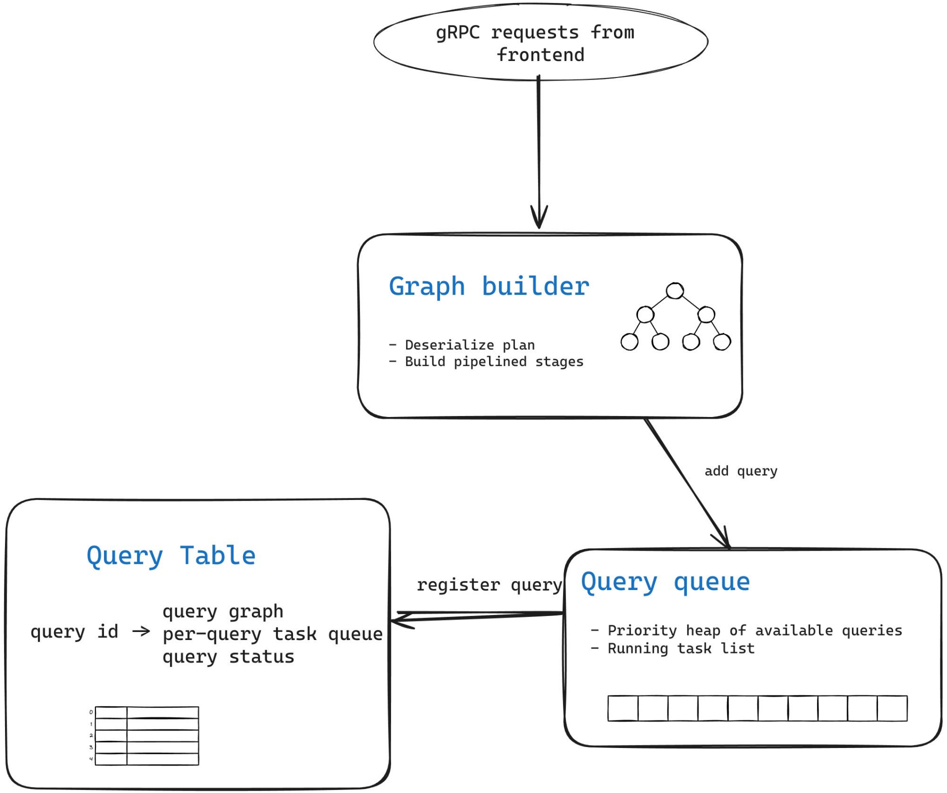
Item 1: Architectural Components – Internal

- Fundamental Internal Components and Interfaces
    - QueryID Table
        - Concurrent table and task structures, interfaces
    - Query Queue
        - Query-based stride scheduling
        - Per-query task queue (FIFO)
    - Pull-based task scheduling framework, EE-facing API

Item 2: Architectural Components – Executor-side

- Fundamental Internal Components and Interfaces
  - Interface for communicating intermediate results
    - Thread-safe HashMap<TaskKey, Vec<RecordBatch>
    - TaskKey = query ID + stage ID
    - Final results sent (Mock EE -> Scheduler -> Mock Frontend)
    - Blocking pull-based task assignment

Workflow – Query Setup



gRPC requests from frontend

Graph builder

- Deserialize plan
- Build pipelined stages

add query

Query queue

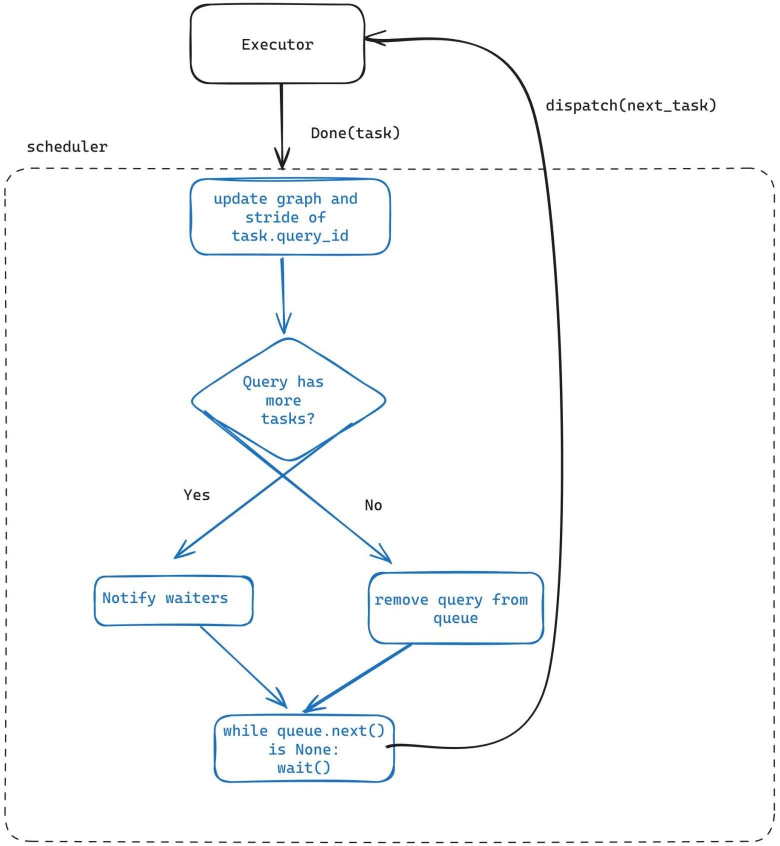
- Priority heap of available queries
- Running task list

register query

Query Table

query id → query graph
per-query task queue
query status

Workflow – Task Dispatch



Testing:
Overview and Architecture

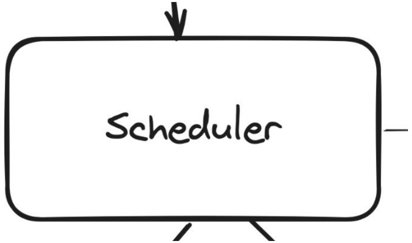
Testing Framework – Internal Components

- Component/Integration tests for core and auxiliary structures
    - Task queue, intermediate result table, DAG parser, query graph soundness
- Concurrency/Stress tests for concurrent structures
- Tests correct behavior of scheduling policy

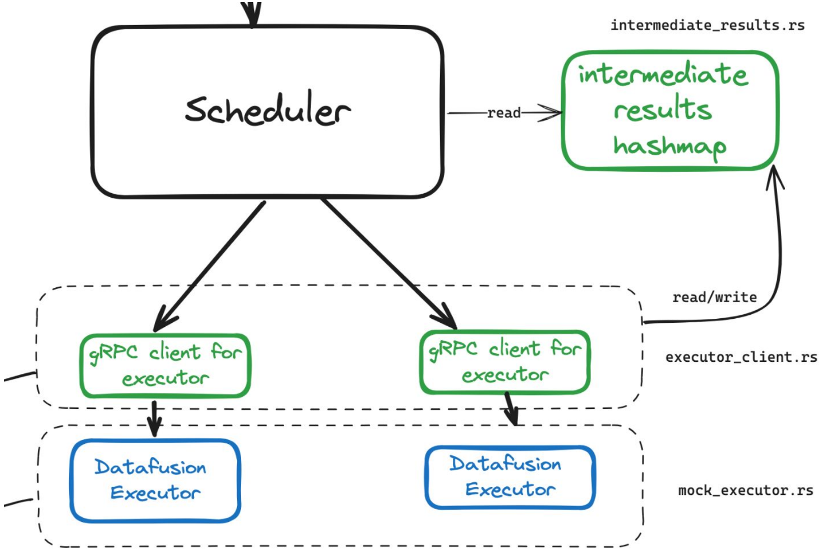
E2E Testing Framework

- Complete system with frontend, scheduler, mock executors, optimizer and catalog
- Supports **end-to-end query execution** and **result verification**
- Includes profiling and performance visualization tools
- **Highly modular** for future component integration

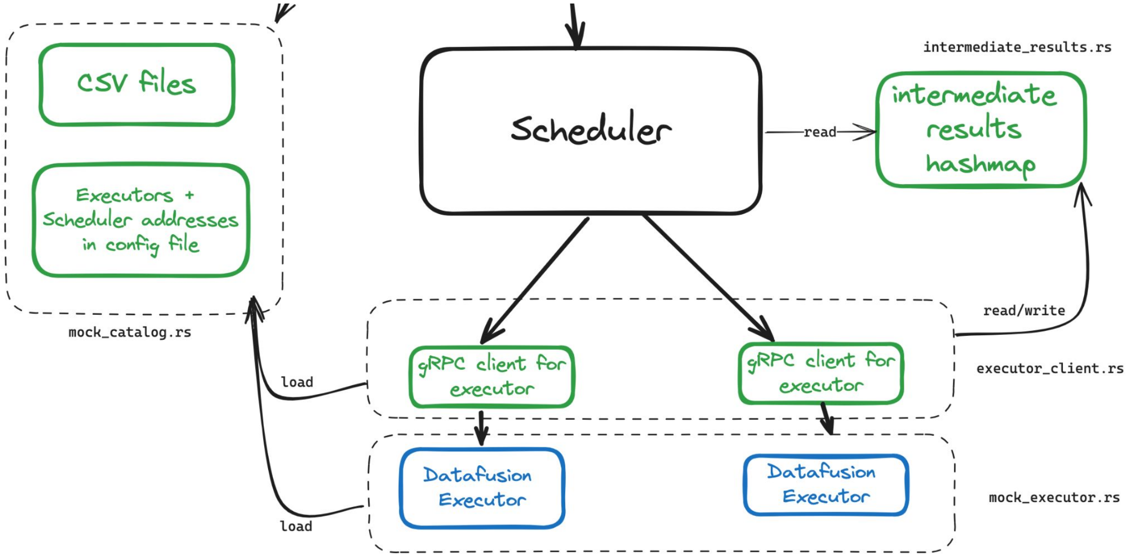
E2E Testing: Architecture



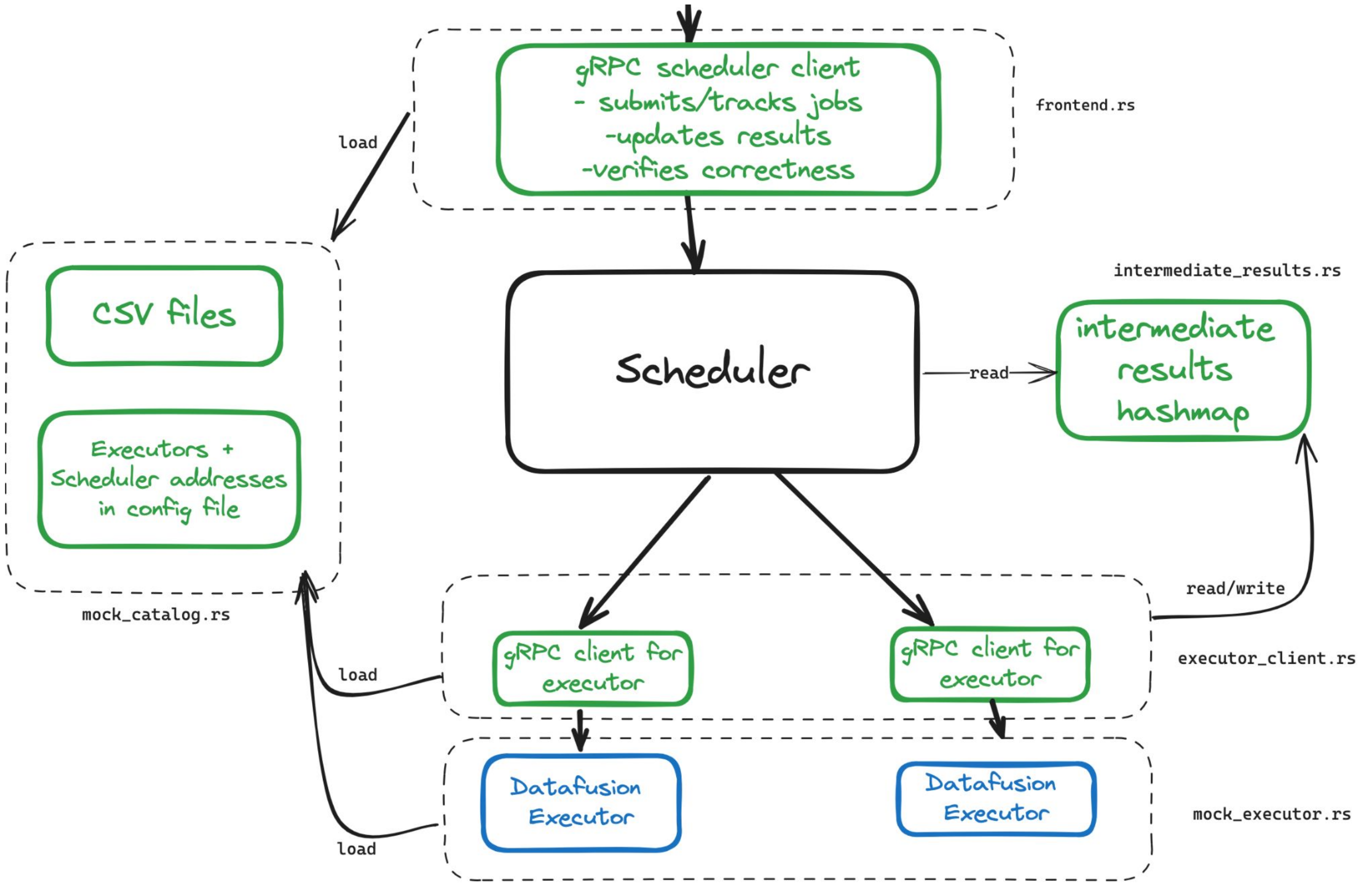
E2E Testing: Architecture

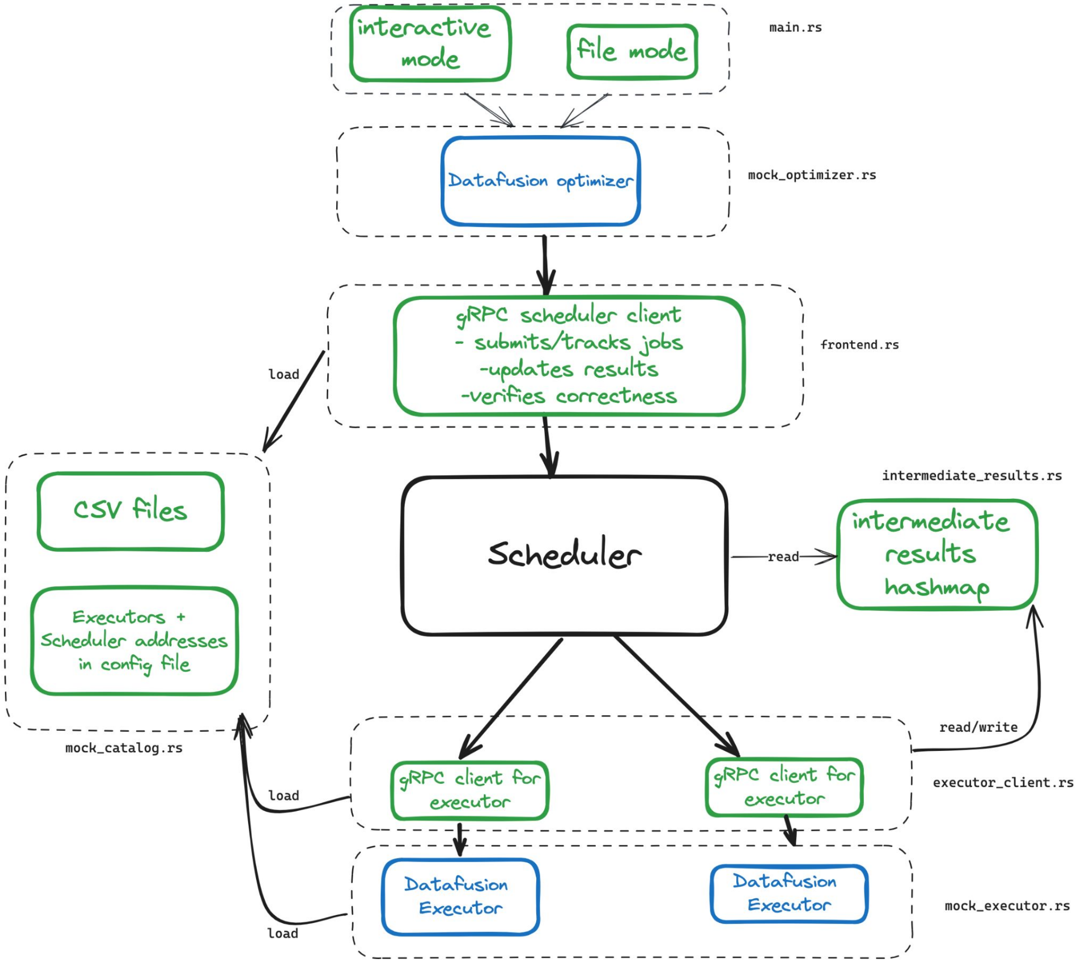


E2E Testing: Architecture



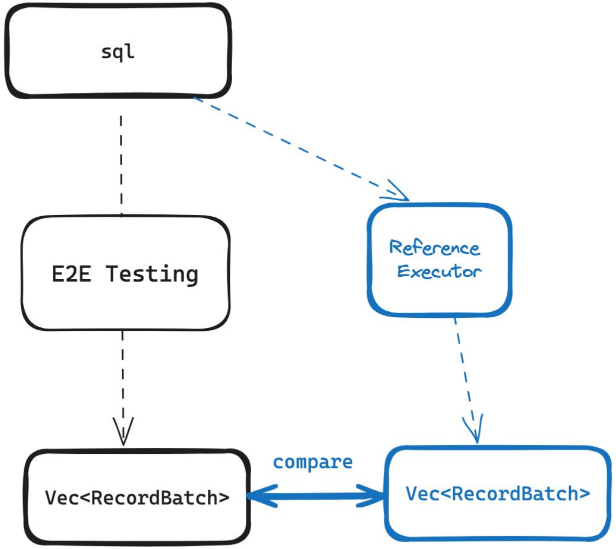
E2E Testing: Architecture




interactive mode
file mode
main.rs
Datafusion optimizer
mock_optimizer.rs
gRPC scheduler client
- submits/tracks jobs
-updates results
-verifies correctness
frontend.rs
load
CSV files
Executors + Scheduler addresses in config file
mock_catalog.rs
Scheduler
read
intermediate_results.rs
intermediate results hashmap
read/write
gRPC client for executor
gRPC client for executor
executor_client.rs
load
Datafusion Executor
Datafusion Executor
mock_executor.rs
load

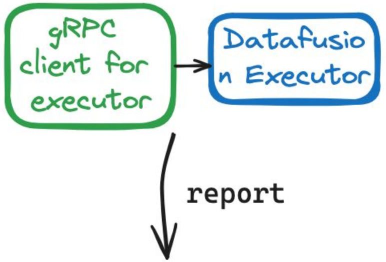
E2E Testing Features: Result Verification

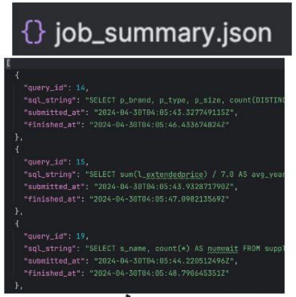
- Query results compared against reference Datafusion executor



E2E Testing Features: Profiling

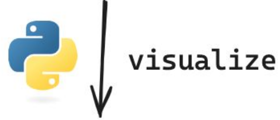
- Logs query submission/completion times
- Tracks executor client activity (busy/idle)
- Python tool for visualization

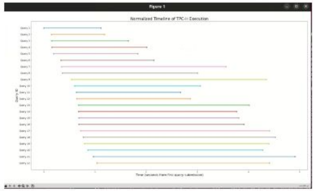




{} job_summary.json

```
[
  {
    "query_id": 14,
    "sql_string": "SELECT p_brand, p_type, p_size, count(DISTIN
    "submitted_at": "2024-04-30T04:05:43.327749115Z",
    "finished_at": "2024-04-30T04:05:46.433674824Z"
  },
  {
    "query_id": 15,
    "sql_string": "SELECT sum(l_extendedprice) / 7.0 AS avg_year
    "submitted_at": "2024-04-30T04:05:43.932871790Z",
    "finished_at": "2024-04-30T04:05:47.098213569Z"
  },
  {
    "query_id": 19,
    "sql_string": "SELECT s_name, count(*) AS numwait FROM supp
    "submitted_at": "2024-04-30T04:05:44.220512496Z",
    "finished_at": "2024-04-30T04:05:48.790645351Z"
  },
```

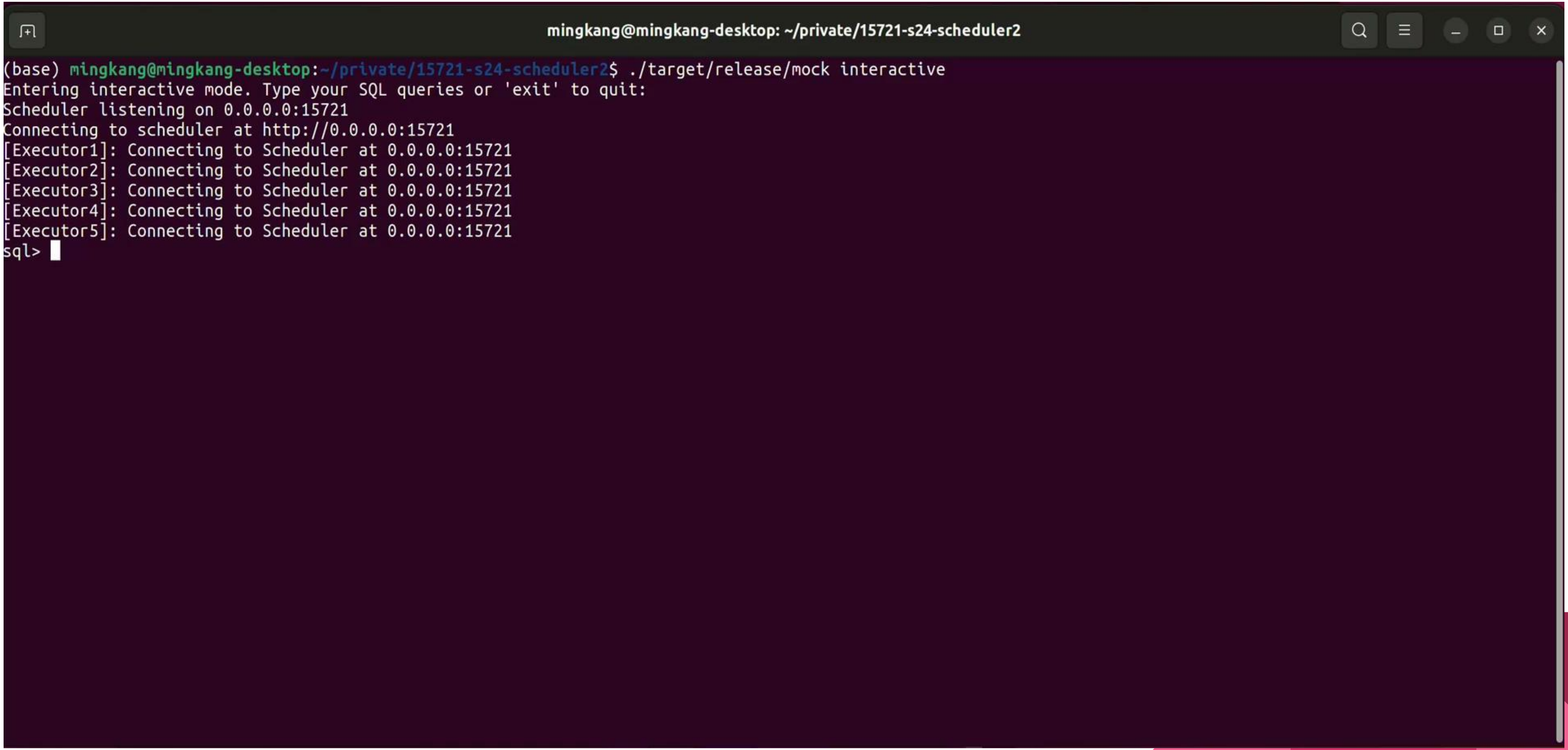




E2E Testing Features: Modularity

- Easily replaceable optimizer and executor functions
  - `optimize (p: &LogicalPlan) -> Result<Arc<dyn ExecutionPlan>>`
  - `execute (e: Arc<dyn ExecutionPlan>) -> Result<Vec<RecordBatch>>`
- Framework supports future integration

E2E Testing: Interactive Mode Demo

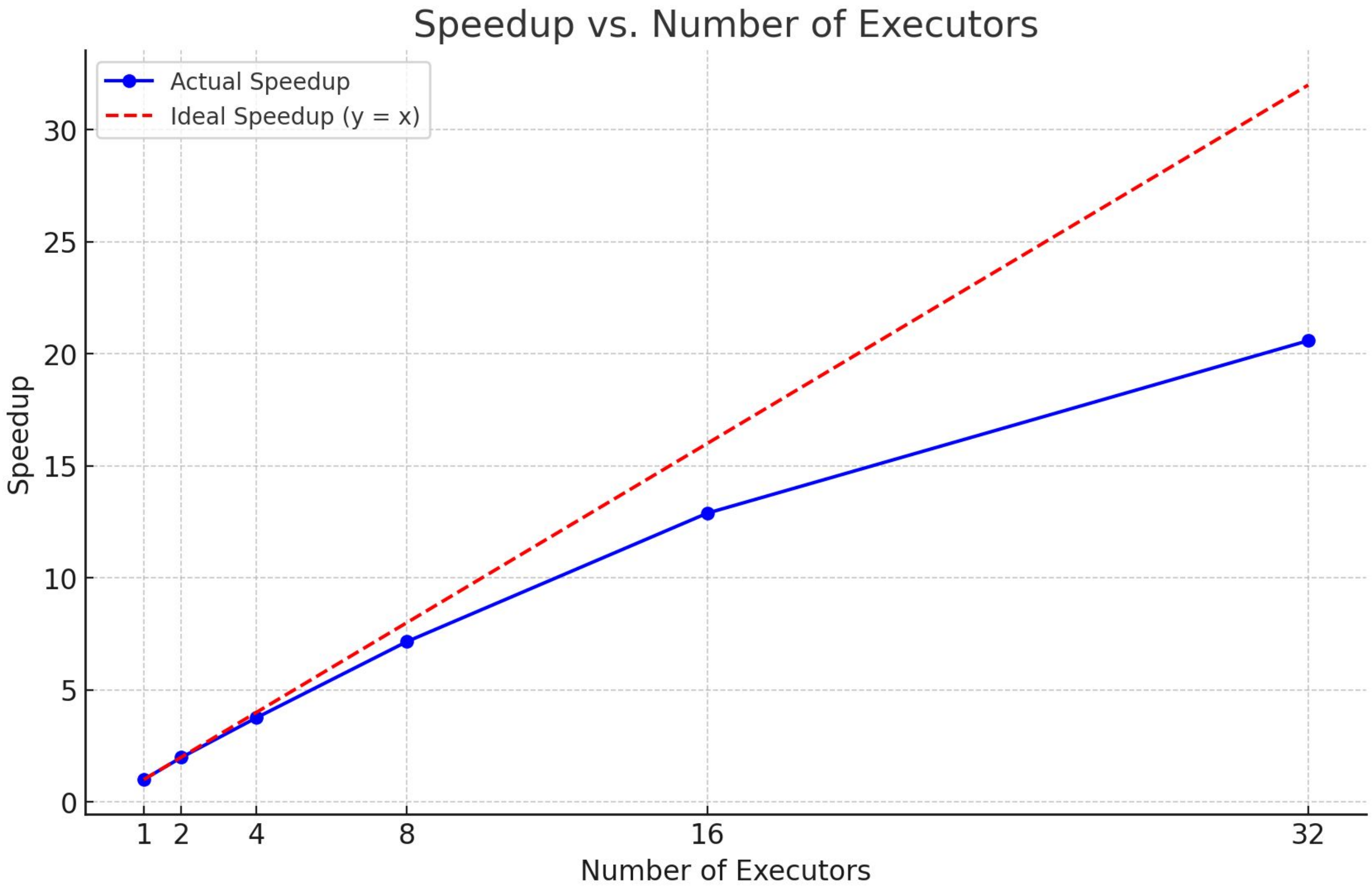


```
(base) mingkang@mingkang-desktop:~/private/15721-s24-scheduler2$ ./target/release/mock interactive
Entering interactive mode. Type your SQL queries or 'exit' to quit:
Scheduler listening on 0.0.0.0:15721
Connecting to scheduler at http://0.0.0.0:15721
[Executor1]: Connecting to Scheduler at 0.0.0.0:15721
[Executor2]: Connecting to Scheduler at 0.0.0.0:15721
[Executor3]: Connecting to Scheduler at 0.0.0.0:15721
[Executor4]: Connecting to Scheduler at 0.0.0.0:15721
[Executor5]: Connecting to Scheduler at 0.0.0.0:15721
sql> 
```

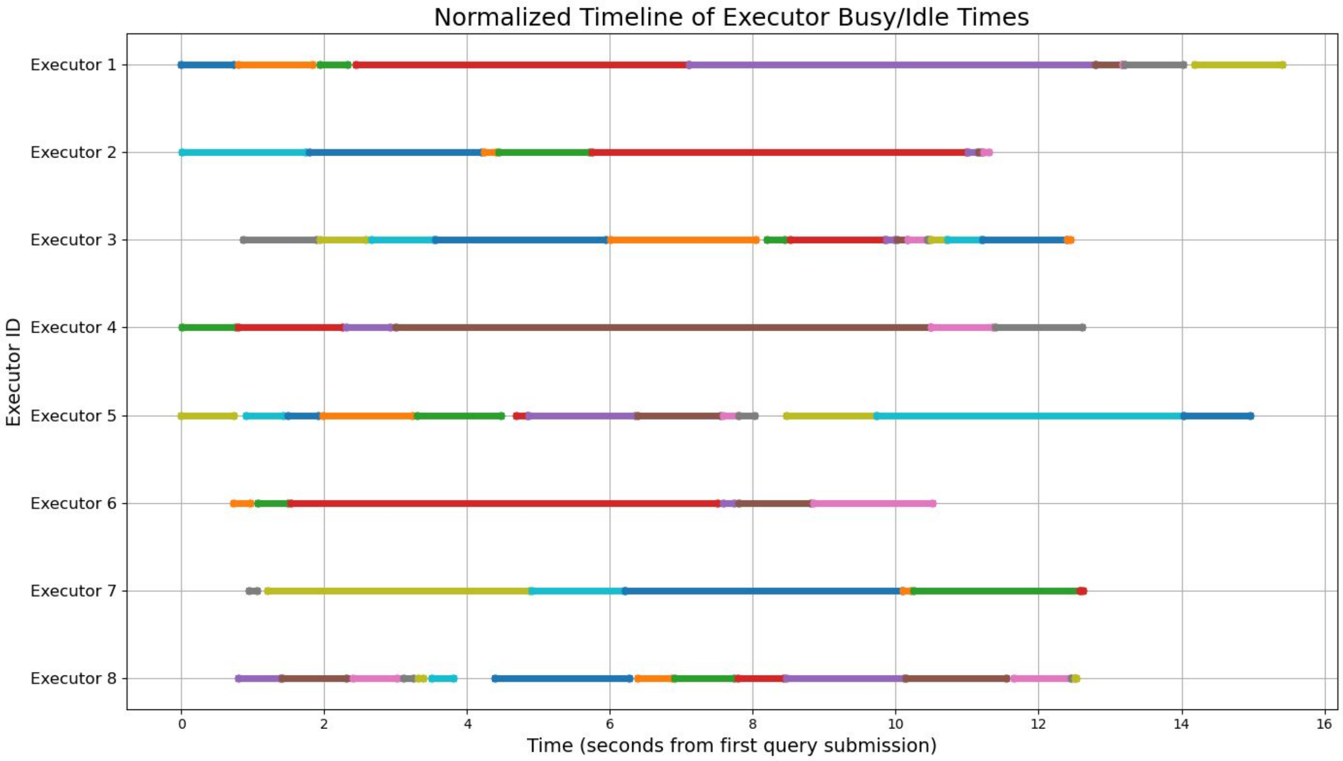
E2E Testing: Batch Submission/Profiling Demo

Benchmark Results

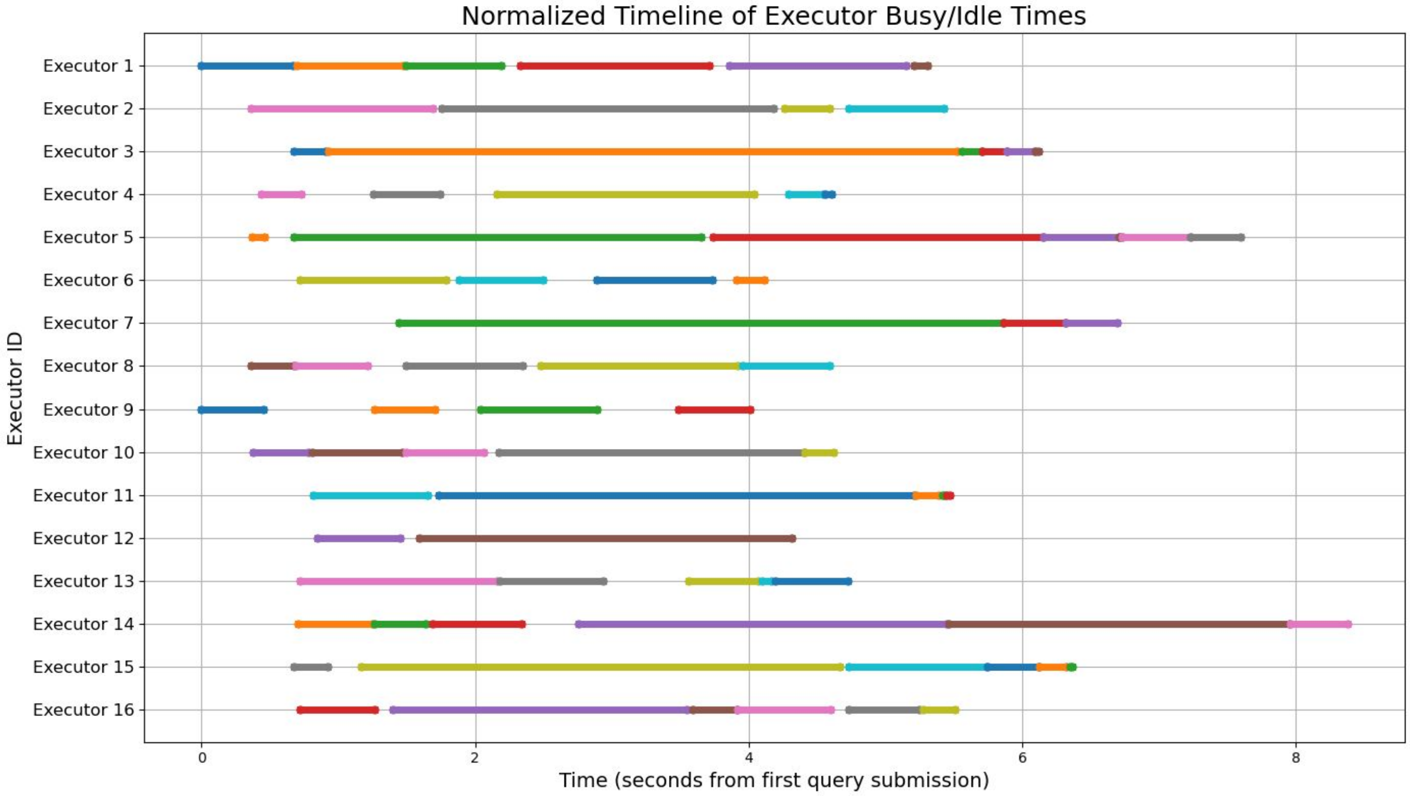
- Batch submitted all 22 TPC-H queries (scale factor: 2) on AWS EC2 with 32 vCPUs
- Tested with 1, 2, 4, 8, 16, 32 executors
- Complete data/graphs available here


Speedup vs. Number of Executors
Actual Speedup
Ideal Speedup (y = x)
Speedup
0
5
10
15
20
25
30
Number of Executors
1
2
4
8
16
32

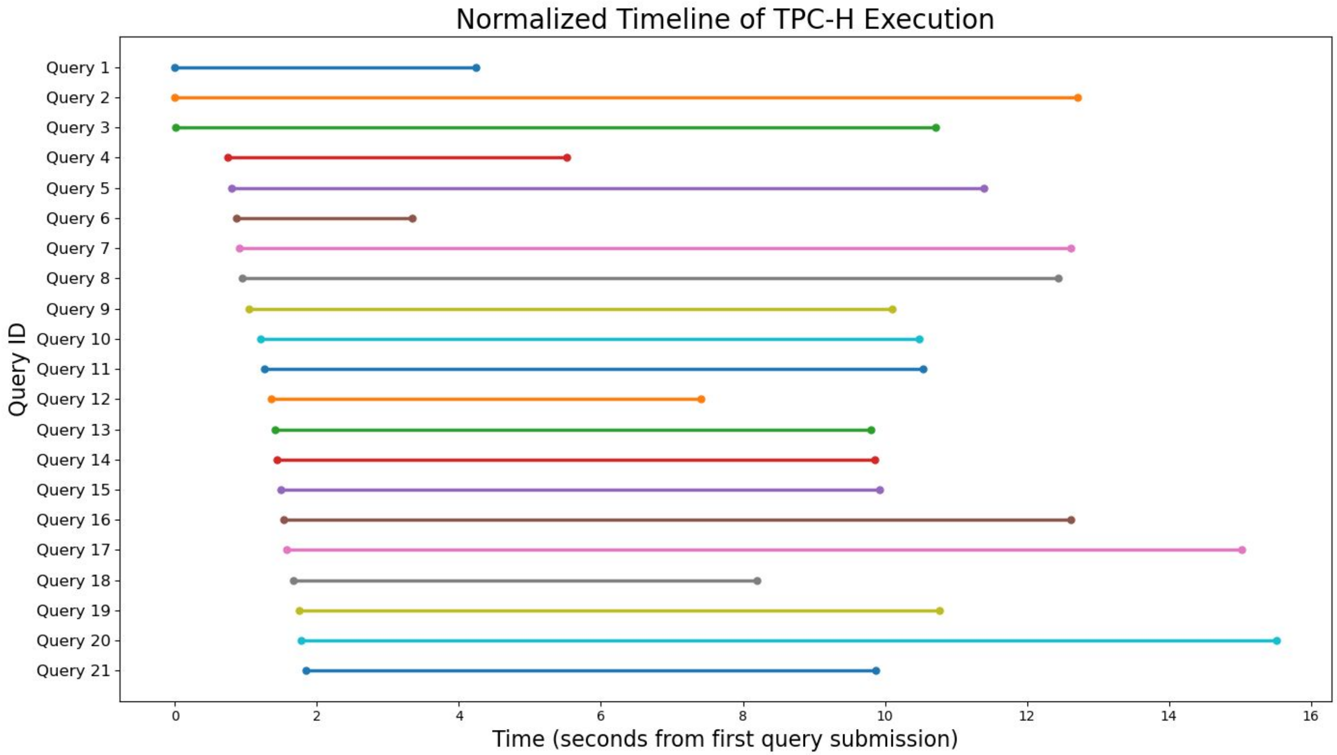
Busy/Idle Time Visualization (8 Executors)



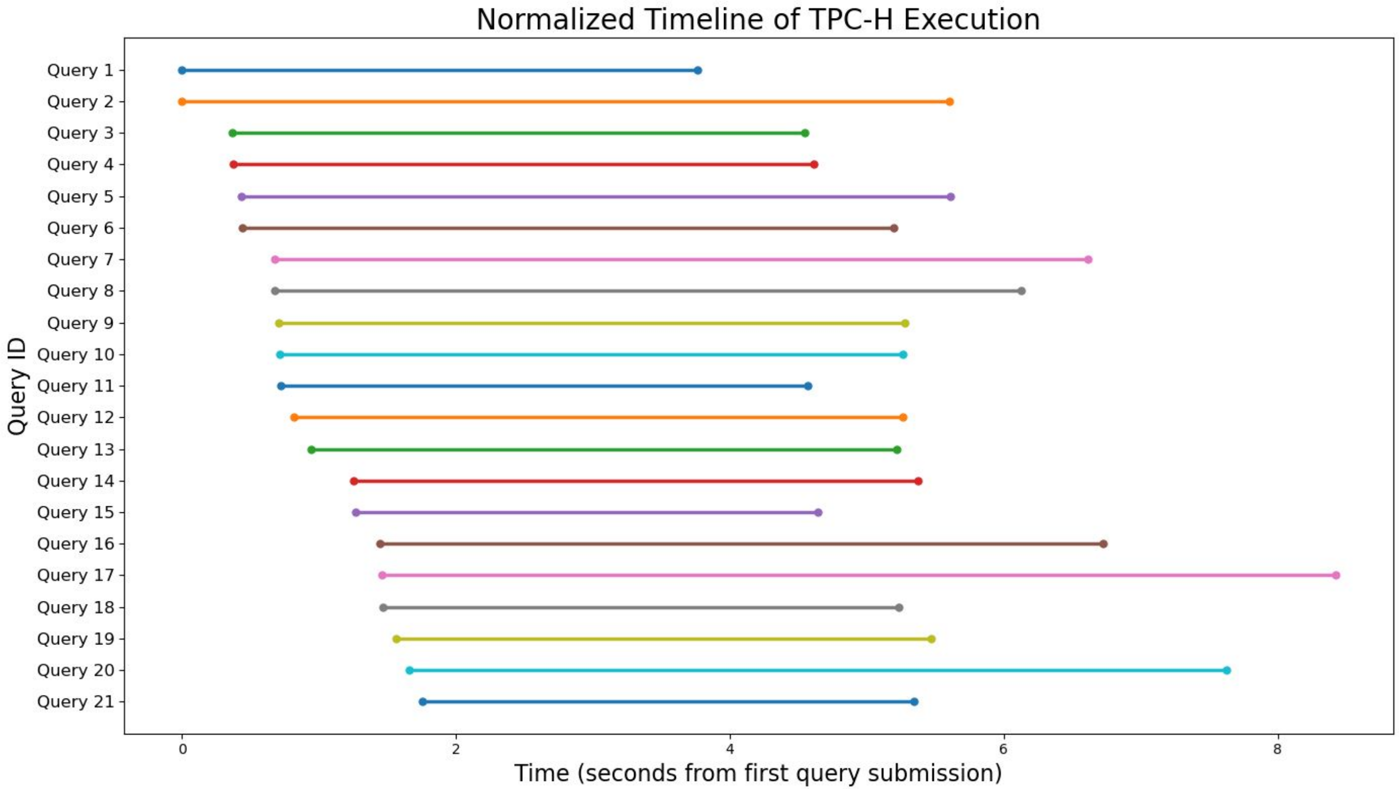
Busy/Idle Time Visualization (16 Executors)



Query Timeline (8 Executors)



Query Timeline (16 Executors)



Code Quality Assessment

Strong Areas:

- Core scheduling data structures and business logic
- E2E testing framework

Weak Areas:

- Error handling -> Should use more uniform approach
- Unittesting -> Suffered due to convenience of E2E framework
- General Robustness -> Presumes client familiarity

Future work

- Address less robust areas of the codebase
- More advanced scheduling policy
- Explore intra-query task ordering strategies
- NUMA-aware locality optimizations
- Morsel-based intra-operator parallelism
- Integration with other components