Testing Accuracy of Query Optimizer

Dmitrii Sherstobitov

Friday, Feb/17/2023

YugabyteDB Friday Tech Talks

YFTT

Talks by Engineers for Engineers

yugabyteDB

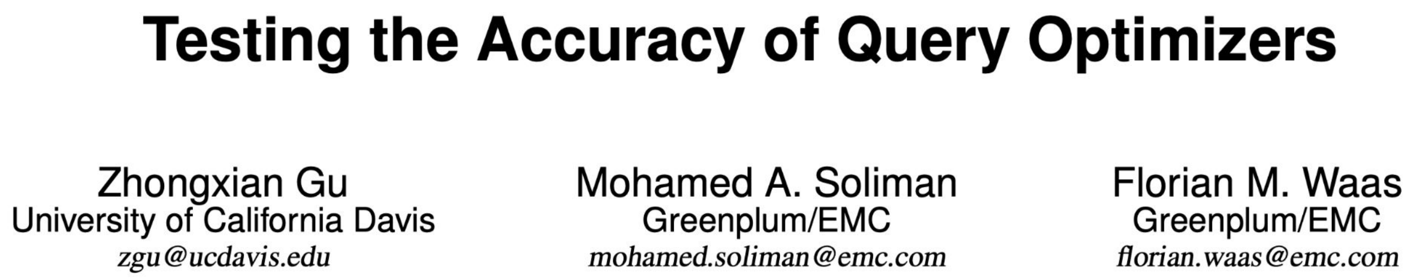
Testing query optimization and execution

Common testing approaches to cover QO and QE

- Unit and integration tests
- Benchmarks and microbenchmarks:
  - TPC*
  - join-order-benchmark
  - ClickBench
- Random query generation (sqlsmith, SQLancer)

Searching for alternative method

1. Adding new cases should be as simple as possible
   a. Possibility to use third-party QE frameworks
   b. Cover client cases
2. Should have clear failure criteria
3. Readable reporting



Testing the Accuracy of Query Optimizers

Zhongxian Gu
University of California Davis
zgu@ucdavis.edu

Mohamed A. Soliman
Greenplum/EMC
mohamed.soliman@emc.com

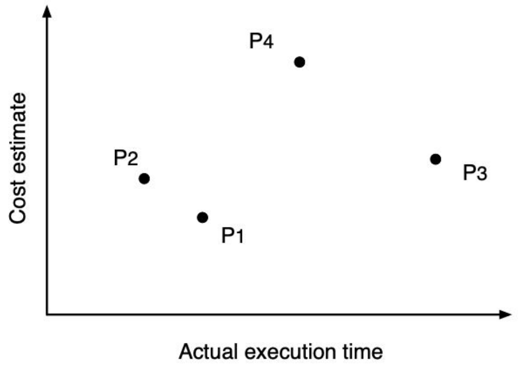
Florian M. Waas
Greenplum/EMC
florian.waas@emc.com

What TAQO page is about (very briefly)

Evaluate original query

Collect information from original execution plan:

- Estimated cost
- Actual execution time

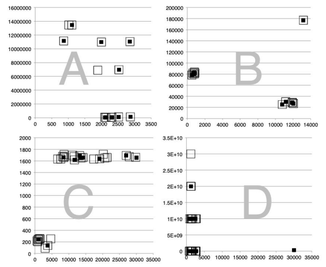


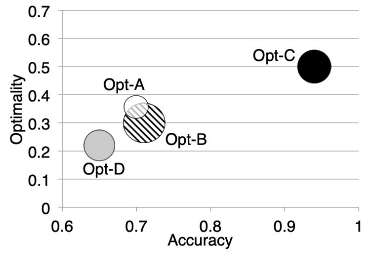
Evaluate possible optimizations

Get set of possible optimizations and evaluate each

Calculate score and compare vendors

$$s = \sum_{i<j} w_i w_j d_{ij} \cdot \mathrm{sgn}(e_j - e_i)$$





Automation

1. pg_hint usage
   a. Predictable plans with pg_hint_plan full hinting
   b. YFTT about pg_hint

```
explain (costs off, timing off)
/*+ Leading( ( c (b a) ) ) NestLoop(a b c) IndexScan( a ) */
select * from table_a a join table_b b using(id)
                        join table_c c using(id);
```

2. Finding all tables in query and generate possible query hints

Automation challenges: Generating Leading hints

For 3 tables `'a', 'b', 'c'` there will be following permutations generated:

```
[('a', 'b', 'c'), ('a', 'c', 'b'), ('b', 'a', 'c'), ('b', 'c', 'a'), ('c', 'a', 'b'), ('c', 'b', 'a')].
```

Each permutation will be transformed into a Leading hint: `Leading ((a b) c) Leading ((a c) b).`

Try to generate all possible combinations of NestedLoop, Merge, Hash joins

```
Leading ((a b) c) Merge(a b) Merge(a b c)

Leading ((a b) c) Merge(a b) Hash(a b c)

...
```

Apply all possible combinations of scans based on the tables used and its indexes.

Automation challenges: Reduce number of combinations

Huge amount of possible optimizations in case of 3+ tables with indexes

1. Using pairwise technique
   a. Pairwise testing, also known as all-pairs testing is **the method of finding defects by using a combinational method of two test cases**. It relies on the observation that most defects are caused by interaction of at most two factors.
   b. There are 3 joins that should appear in the execution plan, one of `['Nested','Hash','Merge']` for each 2 tables. Here is the list of combinations that generated by using pairwise approach:

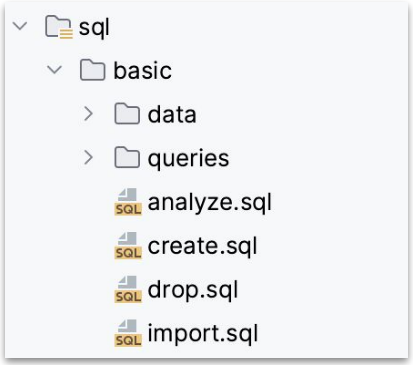
```
['Nested', 'Nested', 'Nested'], ['Nested', 'Merge', 'Hash'], ['Nested', 'Hash', 'Merge']
['Hash', 'Hash', 'Nested'] , ['Hash', 'Merge', 'Merge'], ['Hash', 'Nested', 'Hash'],
['Merge', 'Hash', 'Hash'], ['Merge', 'Merge', 'Nested'], ['Merge', 'Nested', 'Merge']
```

   Note that here each 3 tables (2 joins) will be tried to be joined by each join type, but for example `[‘Merge’,'Merge','Merge']`combination is not here.

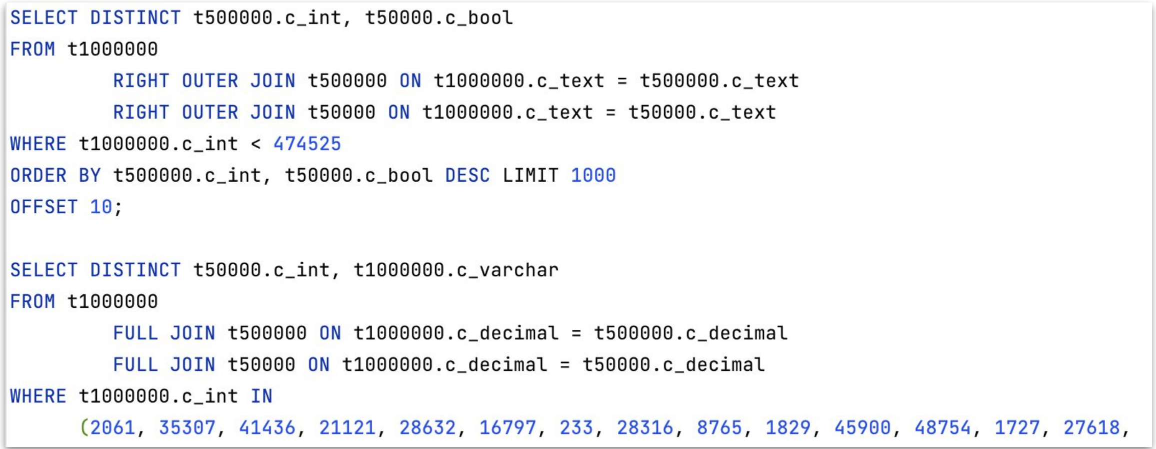
2. Special hints to allow/reject joins or table orders and limit execution time at start
3. Limit query timeout with currently the best execution time

More details about framework

- Input data is set of SQL files
- Each query evaluated few times and collected average execution time
- System Under Test can be any cluster with any data distribution
- TAQO algorithm execution time depends on testing queries set
- Report in HTML and XLS formats



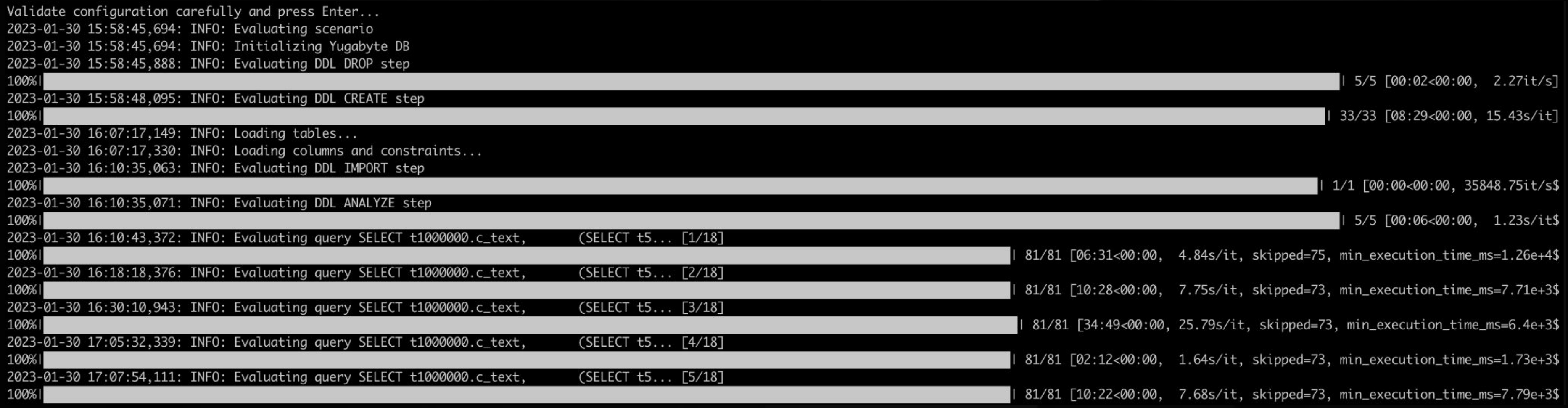
```
sql
  basic
    data
    queries
    analyze.sql
    create.sql
    drop.sql
    import.sql
```



```
SELECT DISTINCT t500000.c_int, t50000.c_bool
FROM t1000000
         RIGHT OUTER JOIN t500000 ON t1000000.c_text = t500000.c_text
         RIGHT OUTER JOIN t50000 ON t1000000.c_text = t50000.c_text
WHERE t1000000.c_int < 474525
ORDER BY t500000.c_int, t50000.c_bool DESC LIMIT 1000
OFFSET 10;

SELECT DISTINCT t50000.c_int, t1000000.c_varchar
FROM t1000000
         FULL JOIN t500000 ON t1000000.c_decimal = t500000.c_decimal
         FULL JOIN t50000 ON t1000000.c_decimal = t50000.c_decimal
WHERE t1000000.c_int IN
      (2061, 35307, 41436, 21121, 28632, 16797, 233, 28316, 8765, 1829, 45900, 48754, 1727, 27618,
```

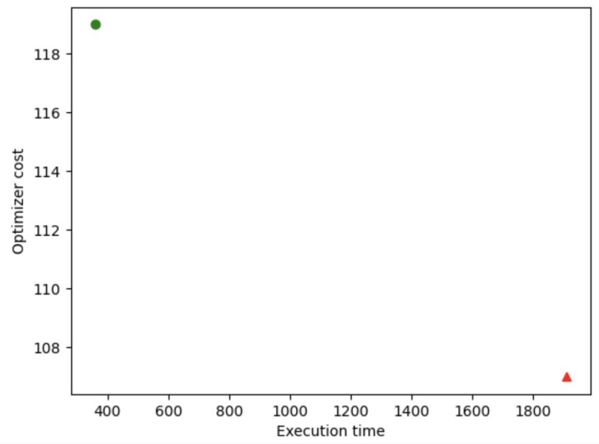
Instead of Demo...



```
Validate configuration carefully and press Enter...
2023-01-30 15:58:45,694: INFO: Evaluating scenario
2023-01-30 15:58:45,694: INFO: Initializing Yugabyte DB
2023-01-30 15:58:45,888: INFO: Evaluating DDL DROP step
100%|████████████████████████████████████████████████████| 5/5 [00:02<00:00,  2.27it/s]
2023-01-30 15:58:48,095: INFO: Evaluating DDL CREATE step
100%|███████████████████████████████████████████████████| 33/33 [08:29<00:00, 15.43s/it]
2023-01-30 16:07:17,149: INFO: Loading tables...
2023-01-30 16:07:17,330: INFO: Loading columns and constraints...
2023-01-30 16:10:35,063: INFO: Evaluating DDL IMPORT step
100%|██████████████████████████████████████████████████| 1/1 [00:00<00:00, 35848.75it/s$
2023-01-30 16:10:35,071: INFO: Evaluating DDL ANALYZE step
100%|████████████████████████████████████████████████████| 5/5 [00:06<00:00,  1.23s/it$
2023-01-30 16:10:43,372: INFO: Evaluating query SELECT t1000000.c_text,      (SELECT t5... [1/18]
100%|██████████████████████████████████████| 81/81 [06:31<00:00,  4.84s/it, skipped=75, min_execution_time_ms=1.26e+4$
2023-01-30 16:18:18,376: INFO: Evaluating query SELECT t1000000.c_text,      (SELECT t5... [2/18]
100%|██████████████████████████████████████| 81/81 [10:28<00:00,  7.75s/it, skipped=73, min_execution_time_ms=7.71e+3$
2023-01-30 16:30:10,943: INFO: Evaluating query SELECT t1000000.c_text,      (SELECT t5... [3/18]
100%|██████████████████████████████████████| 81/81 [34:49<00:00, 25.79s/it, skipped=73, min_execution_time_ms=6.4e+3$
2023-01-30 17:05:32,339: INFO: Evaluating query SELECT t1000000.c_text,      (SELECT t5... [4/18]
100%|██████████████████████████████████████| 81/81 [02:12<00:00,  1.64s/it, skipped=73, min_execution_time_ms=1.73e+3$
2023-01-30 17:07:54,111: INFO: Evaluating query SELECT t1000000.c_text,      (SELECT t5... [5/18]
100%|██████████████████████████████████████| 81/81 [10:22<00:00,  7.68s/it, skipped=73, min_execution_time_ms=7.79e+3$
```

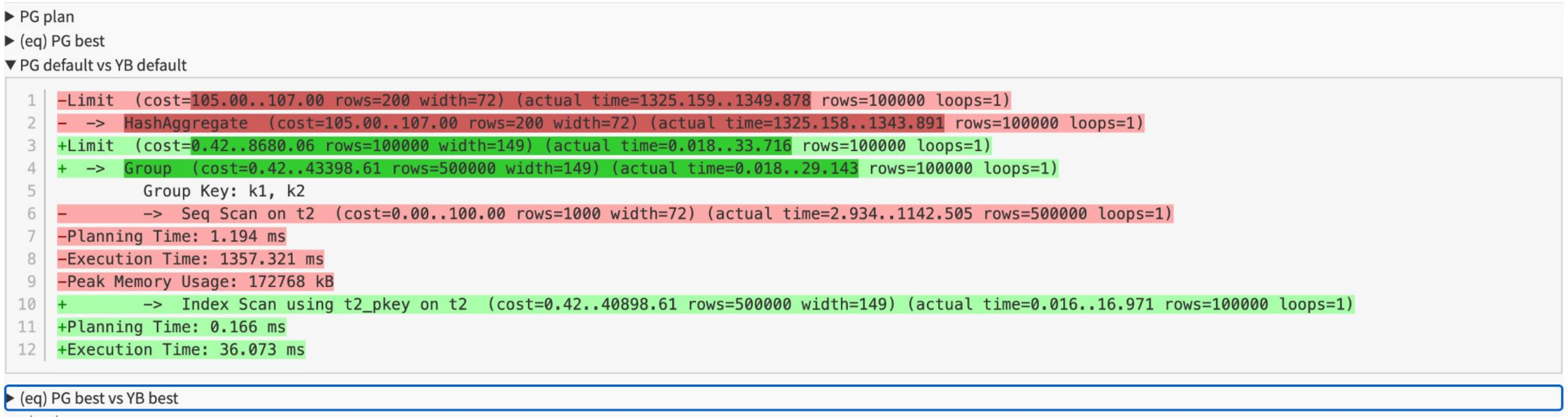
Reporting: Basics

```sql
SELECT t2.k1,
       t2.k2,
       t2.v1,
       t2.v2
FROM   t2
GROUP BY t2.k1, t2.k2, t2.v1, t2.v2 limit 100000
```



```
-Limit  (cost=105.00..107.00 rows=200 width=72) (actual time=1325.159..1349.878 rows=100000 loops=1)
-  ->  HashAggregate  (cost=105.00..107.00 rows=200 width=72) (actual time=1325.158..1343.891 rows=100000 loops=1)
+Limit  (cost=0.00..119.00 rows=200 width=72) (actual time=2.655..248.113 rows=100000 loops=1)
+  ->  Group  (cost=0.00..119.00 rows=200 width=72) (actual time=2.655..241.981 rows=100000 loops=1)
         Group Key: k1, k2
-        ->  Seq Scan on t2  (cost=0.00..100.00 rows=1000 width=72) (actual time=2.934..1142.505 rows=500000 loops=1)
-Planning Time: 1.194 ms
-Execution Time: 1357.321 ms
-Peak Memory Usage: 172768 kB
+        ->  Index Scan using t2_pkey on t2  (cost=0.00..114.00 rows=1000 width=72) (actual time=2.653..222.052 rows=100000 loops=1)
+Planning Time: 0.503 ms
+Execution Time: 254.172 ms
+Peak Memory Usage: 8512 kB
```

Reporting: More information



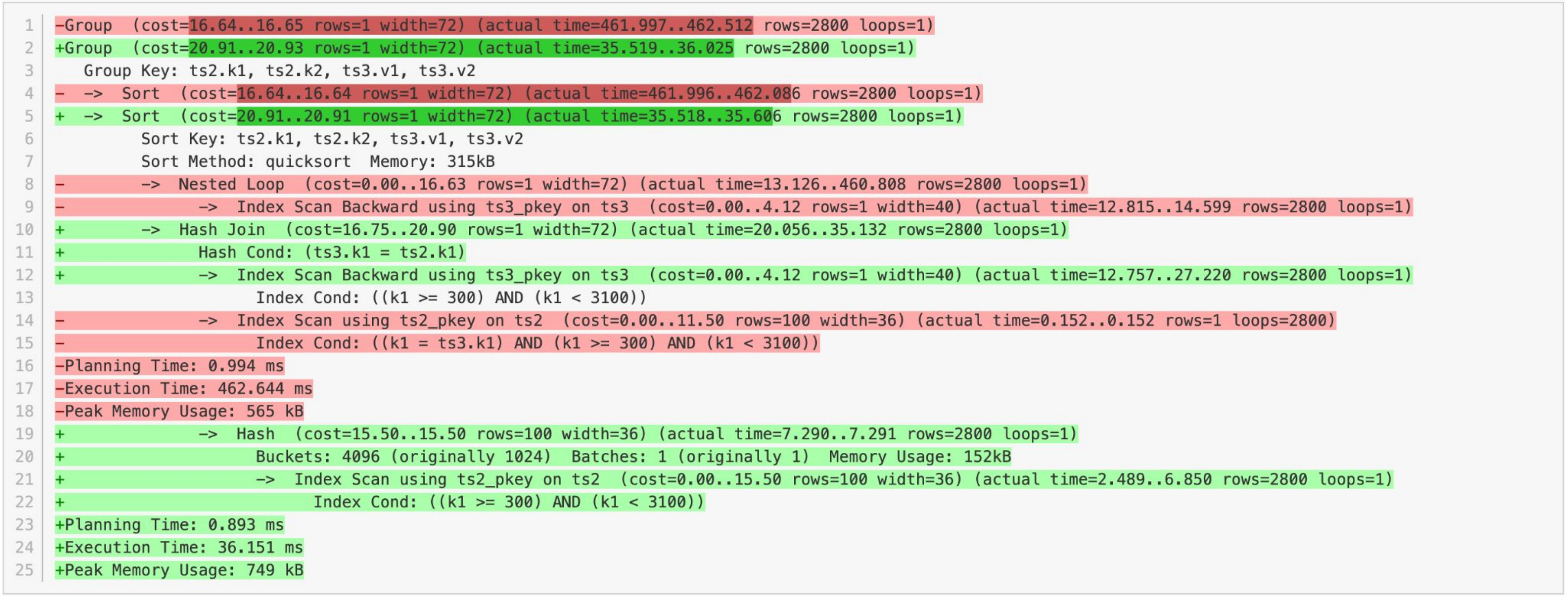
▶ PG plan
▶ (eq) PG best
▼ PG default vs YB default

```
-Limit  (cost=105.00..107.00 rows=200 width=72) (actual time=1325.159..1349.878 rows=100000 loops=1)
-  ->  HashAggregate  (cost=105.00..107.00 rows=200 width=72) (actual time=1325.158..1343.891 rows=100000 loops=1)
+Limit  (cost=0.42..8680.06 rows=100000 width=149) (actual time=0.018..33.716 rows=100000 loops=1)
+  ->  Group  (cost=0.42..43398.61 rows=500000 width=149) (actual time=0.018..29.143 rows=100000 loops=1)
         Group Key: k1, k2
-          ->  Seq Scan on t2  (cost=0.00..100.00 rows=1000 width=72) (actual time=2.934..1142.505 rows=500000 loops=1)
-Planning Time: 1.194 ms
-Execution Time: 1357.321 ms
-Peak Memory Usage: 172768 kB
+          ->  Index Scan using t2_pkey on t2  (cost=0.42..40898.61 rows=500000 width=149) (actual time=0.016..16.971 rows=100000 loops=1)
+Planning Time: 0.166 ms
+Execution Time: 36.073 ms
```

▶ (eq) PG best vs YB best

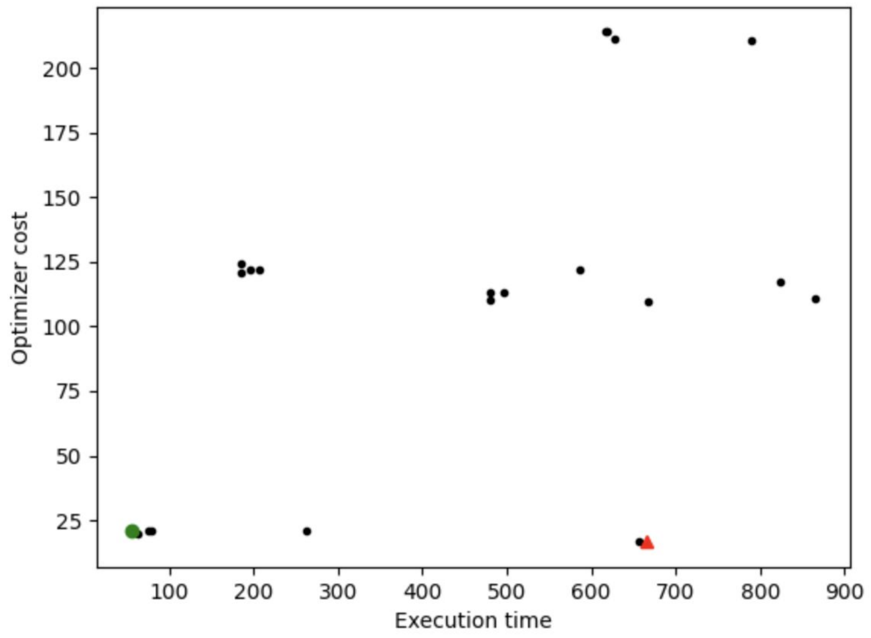
Metric	YB	YB Best	PG	PG Best
Cardinality	100000	100000	100000	100000
Estimated cost	107.0	119.0	8680.06	(eq) 8680.06
Execution time	1910.88	357.59	220.62	(eq) 220.62

More complex case: Plan difference

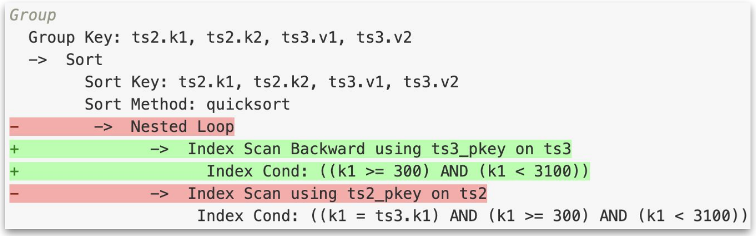


```
-Group  (cost=16.64..16.65 rows=1 width=72) (actual time=461.997..462.512 rows=2800 loops=1)
+Group  (cost=20.91..20.93 rows=1 width=72) (actual time=35.519..36.025 rows=2800 loops=1)
   Group Key: ts2.k1, ts2.k2, ts3.v1, ts3.v2
-  ->  Sort  (cost=16.64..16.64 rows=1 width=72) (actual time=461.996..462.086 rows=2800 loops=1)
+  ->  Sort  (cost=20.91..20.91 rows=1 width=72) (actual time=35.518..35.606 rows=2800 loops=1)
         Sort Key: ts2.k1, ts2.k2, ts3.v1, ts3.v2
         Sort Method: quicksort  Memory: 315kB
-        ->  Nested Loop  (cost=0.00..16.63 rows=1 width=72) (actual time=13.126..460.808 rows=2800 loops=1)
-              ->  Index Scan Backward using ts3_pkey on ts3  (cost=0.00..4.12 rows=1 width=40) (actual time=12.815..14.599 rows=2800 loops=1)
+        ->  Hash Join  (cost=16.75..20.90 rows=1 width=72) (actual time=20.056..35.132 rows=2800 loops=1)
+              Hash Cond: (ts3.k1 = ts2.k1)
+              ->  Index Scan Backward using ts3_pkey on ts3  (cost=0.00..4.12 rows=1 width=40) (actual time=12.757..27.220 rows=2800 loops=1)
                    Index Cond: ((k1 >= 300) AND (k1 < 3100))
-              ->  Index Scan using ts2_pkey on ts2  (cost=0.00..11.50 rows=100 width=36) (actual time=0.152..0.152 rows=1 loops=2800)
-                    Index Cond: ((k1 = ts3.k1) AND (k1 >= 300) AND (k1 < 3100))
-Planning Time: 0.994 ms
-Execution Time: 462.644 ms
-Peak Memory Usage: 565 kB
+              ->  Hash  (cost=15.50..15.50 rows=100 width=36) (actual time=7.290..7.291 rows=2800 loops=1)
+                    Buckets: 4096 (originally 1024)  Batches: 1 (originally 1)  Memory Usage: 152kB
+                    ->  Index Scan using ts2_pkey on ts2  (cost=0.00..15.50 rows=100 width=36) (actual time=2.489..6.850 rows=2800 loops=1)
+                          Index Cond: ((k1 >= 300) AND (k1 < 3100))
+Planning Time: 0.893 ms
+Execution Time: 36.151 ms
+Peak Memory Usage: 749 kB
```

More complex case: Optimizer accuracy



Metric	YB	YB Best	PG	PG Best
Cardinality	2800	2800	2800	2800
Estimated cost	16.65	20.93	147.5	(eq) 147.5
Execution time	665.8	54.77	3.51	(eq) 3.51

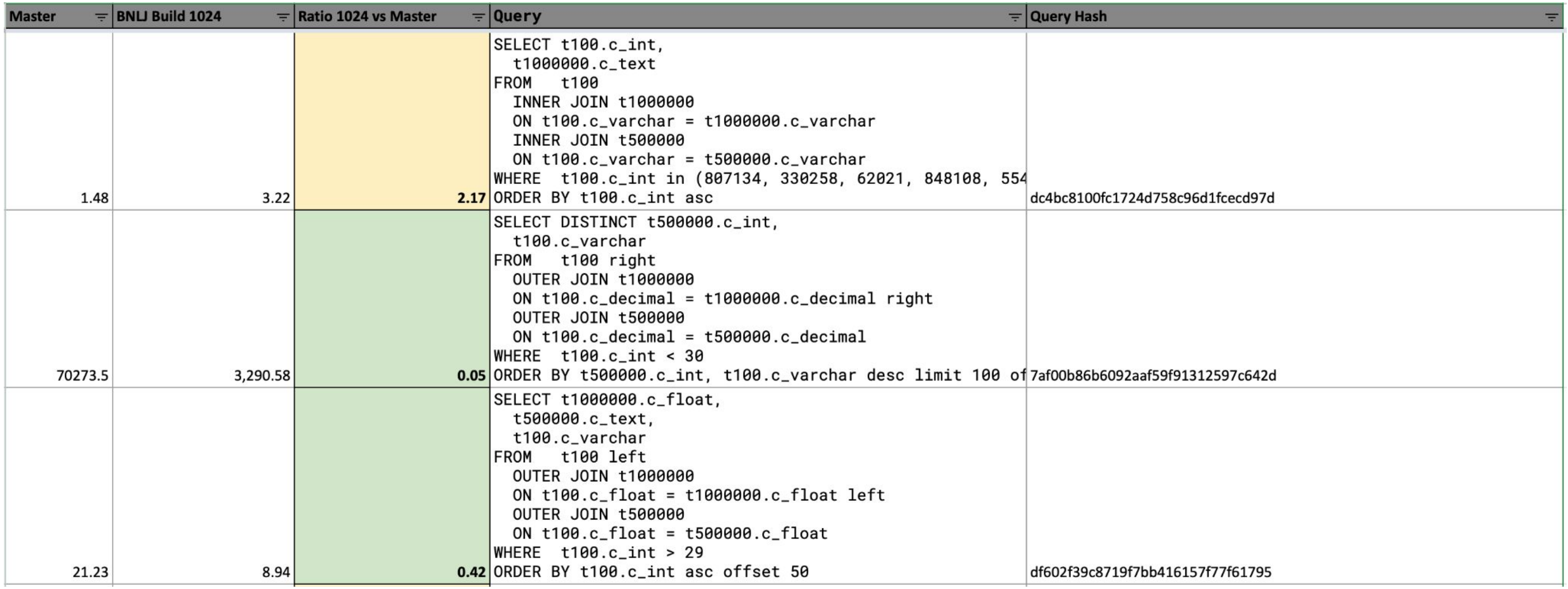


```
Group
  Group Key: ts2.k1, ts2.k2, ts3.v1, ts3.v2
  ->  Sort
        Sort Key: ts2.k1, ts2.k2, ts3.v1, ts3.v2
        Sort Method: quicksort
-           ->  Nested Loop
+                 ->  Index Scan Backward using ts3_pkey on ts3
+                       Index Cond: ((k1 >= 300) AND (k1 < 3100))
-                 ->  Index Scan using ts2_pkey on ts2
                      Index Cond: ((k1 = ts3.k1) AND (k1 >= 300) AND (k1 < 3100))
```

Report: Summary tables

YB	YB Best	PG	PG Best	Ratio YB vs PG	Best YB vs PG	Query
1181.22	**1181.22**	1140.87	**1140.87**	**1.04**	**1.04**	SELECT |/ t, 'foo ' || t, now() FROM generate_series(1, 1000000) as t
1263.52	**1263.52**	83.53	**83.53**	**15.13**	**15.13**	SELECT * FROM t1 WHERE v1 > %(20000) ORDER BY v1 asc limit 100000
5984.56	2865.16	397.54	**397.54**	**15.05**	**7.21**	SELECT * FROM t1 WHERE v1 > %(20000) ORDER BY v1 asc limit 1000000
1352.35	**1352.35**	237.44	**237.44**	**5.7**	**5.7**	SELECT * FROM t2 WHERE v1 > %(20000) ORDER BY v1 asc limit 100000

Report: Regression and new feature testing example



Master	BNLJ Build 1024	Ratio 1024 vs Master	Query	Query Hash
1.48	3.22	2.17	`SELECT t100.c_int, t1000000.c_text FROM t100 INNER JOIN t1000000 ON t100.c_varchar = t1000000.c_varchar INNER JOIN t500000 ON t100.c_varchar = t500000.c_varchar WHERE t100.c_int in (807134, 330258, 62021, 848108, 554 ORDER BY t100.c_int asc`	dc4bc8100fc1724d758c96d1fcecd97d
70273.5	3,290.58	0.05	`SELECT DISTINCT t500000.c_int, t100.c_varchar FROM t100 right OUTER JOIN t1000000 ON t100.c_decimal = t1000000.c_decimal right OUTER JOIN t500000 ON t100.c_decimal = t500000.c_decimal WHERE t100.c_int < 30 ORDER BY t500000.c_int, t100.c_varchar desc limit 100 of`	7af00b86b6092aaf59f91312597c642d
21.23	8.94	0.42	`SELECT t1000000.c_float, t500000.c_text, t100.c_varchar FROM t100 left OUTER JOIN t1000000 ON t100.c_float = t1000000.c_float left OUTER JOIN t500000 ON t100.c_float = t500000.c_float WHERE t100.c_int > 29 ORDER BY t100.c_int asc offset 50`	df602f39c8719f7bb416157f77f61795

Goals achieved

1. Added common query execution frameworks by just adding SQL files
2. Introduced failure criteria based on few automated checks
3. Not hardcoded to specific environments or configurations
4. Report that can provide a lot of information about QO state
   a. Using in regression reporting to validate all recent QO changes
   b. Using in all stages of development - research, development and QA

Roadmap

1. Extend coverage by covering all possible execution plan nodes
    a. Add more benchmark and workloads
    b. Add custom testing suites (DML coverage e.g.)
    c. Implement random query generation
2. Add more sophisticated automated checks
3. Collect more data from query execution process (flamegraphs, resource usage, etc.)

Thank You

Join us on Slack: yugabyte.com/slack (#yftt channel)

Star us on Github: github.com/yugabyte/yugabyte-db

YugabyteDB Friday Tech Talks

YFTT

Talks by Engineers for Engineers

yugabyteDB