Project Part II: Benchmark Design

David Djernaes
Grady Ku

Why we chose Option 1:

System 1 will be Google Cloud SQL-hosted MySQL server, and system 2 will be Google Cloud SQL-hosted PostgreSQL server. We chose Option 1 because we were interested in seeing the performance differences between the two SQL variants and if the relative strengths and weaknesses of either system manifests itself with the Wisconsin Benchmark tests.

System Research:

1) PostgreSQL (hosted by Google Cloud SQL)

   a. *types of indices:*
   - B-tree
   - Hash
   - GiST (infrastructure that can be used to implement multiple indexes)
   - SP-GiST (like GiST except it implements indexes on non-balanced disk-based data structures)
   - GIN (inverted indexes that can handle values that contain more than one key)
   - BRIN (Block Range Index that operates on physical block ranges)

   b. *types of join algorithms:*
   - Nested Loop join
   - Merge join
   - Hash join

   c. *buffer pool size & structure:*

   The default size for buffer pool buffers are 8 kilobytes and the database allocates a default number of 1000 shared buffers in the buffer pool. The buffer pool is implemented as a simple array of buffers, and each individual buffer can be accessed via array indices. The buffer pool indices are known as "buffer_ias".

   d. *mechanisms for measuring query execution time:*

   EXPLAIN ANALYSE <query> will provide detailed information about a query's execution time. We will be using a more specific variant of EXPLAIN ANALYSE by manually setting the TIMING parameter to 'False' so that some of the unnecessary overhead will be reduced.

2) MySQL server (hosted by Google Cloud SQL)

   a. *types of indices:*
   - Hash
   - B-tree
   - Column index (index on a single column)

- Multiple-Column/Composite index
- Descending index (utilizes the DESC operator)
- Spatial (index used on spatial data types and objects)
- FULLTEXT (for efficient full-text searches)

b. *types of join algorithms:*

- Nested-Loop join
- Block Nested-Loop join
- Hash join

c. *buffer pool size & structure:*

The size of the buffer pool buffers can be configured individually. The size of the buffer pool itself must be equal to or be a multiple of (innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances). The buffer pool is implemented as a linked list of buffer pages, with the "hotter" pages at the head of the list and the "colder" pages at the tail of the list. A new page read in from disk is inserted into the middle of the list.

d. *mechanisms for measuring query execution time:*

Google Cloud SQL has a built-in *Stackdriver* tool with a "Trace" command that can be entered into the Cloud shell. The command will list a history of past requests and we will click on the query requests to see their execution times.

3) Query types explored

Four general categories of query types will be used in the project.

a. *The storage organization of the relation*

Various queries in the study will experiment with projections on the indexed attribute and those that are not.

b. *The selectivity outputs of the join predicates*

Varying degrees of output size will be measured between the two database systems to see if there is a notable performance edge with either system.

c. *The output mode of the query*

Whether the output is delivered to the user via the screen or output to disk will be compared.

d. *Comparisons of the hardware and software performance*

Throughout the study, the underlying theme for all the query experiments is to measure the performance via time comparisons to see if one relational database system is more efficient than the other.

Project Part II: Benchmark Design

Performance Experiment Design:

1) Compare join algorithm performance between PostgreSQL and MySQL

   a. *performance issue we are wanting to test:*

   The purpose of this test is to see how the simple join algorithms perform on the two systems. Because we are testing the performance of the join algorithms themselves, we will keep the queries relatively simple.

   b. *data sets used:*

   For the data sets, we will be using a HUNDREDKTUP relation (100K tuples) for both PostgreSQL and MySQL.

   c. *queries used:*

   For this test, we will be using the Wisconsin Benchmark queries #10 and #11. However, for query #10, the other relation will be HUNDREDKTUP instead of BPRIME.

   d. *metric used:*

   For the metric, we will be measuring query execution time using EXPLAIN (ANALYSE TRUE, TIMING FALSE) (PostgreSQL) and Stackdriver "Trace" (MySQL). Our testing methodology will involve 5 runs of the same query with the results averaged, after removing the high and low values from each testing set.

   e. *expected results:*

   For general join queries, PostgreSQL is known to perform slightly better than MySQL and therefore we expect PostgreSQL to exhibit better performance times on our queries. Additionally, we expect query #10 to have overall faster query execution times than query #11 because query #10 will be working with the two larger relations (meaning the applied selection operator should remove more tuples) and the query itself is simpler.

2) Compare query execution times of each system on three different relation sizes

   a. *performance issue we are wanting to test:*

   The purpose of this test is to see how the performance of both systems are affected when the tuple count is first increased by a factor of 10 and then increased again by another factor of 10 (so the total would be a factor of 100).

   b. *data sets used:*

   For this test, we will be using the ONEKTUP relation, the TENKTUP relation, and the HUNDREDKTUP relation.

Project Part II: Benchmark Design

c. *queries used:*

For this test, since we are interested in knowing how relation size impacts the query execution times, we will try two different query sets: one that uses an index, and one that does not. So then for each of the three relations, the following two queries will be tested on both systems:

First set of queries (uses the clustered index):

```
SELECT *
FROM ONEKTUP
WHERE unique2 BETWEEN 450 AND 500

SELECT *
FROM TENKTUP
WHERE unique2 BETWEEN 4500 AND 5000

SELECT *
FROM HUNDREDKTUP
WHERE unique2 BETWEEN 45,000 AND 50,000
```

Second set of queries (no index):

```
SELECT *
FROM ONEKTUP
WHERE tenPercent = 5

SELECT *
FROM TENKTUP
WHERE tenPercent = 50

SELECT *
FROM HUNDREDKTUP
WHERE tenPercent = 500
```

d. *metric used:*

For the metric, we will be measuring query execution time using EXPLAIN (ANALYSE TRUE, TIMING FALSE) (PostgreSQL) and Stackdriver “Trace” (MySQL). Our testing methodology will involve 5 runs of the same query with the results averaged, after removing the high and low values from each testing set.

e. *expected results:*

PostgreSQL is known to have slightly slower execution times than MySQL on all read-heavy workloads so we expect MySQL to produce faster query execution times across the three different relations. Additionally, we expect the indexed query to have better execution times than the non-indexed query for our ONEKTUP relation but we expect the opposite to be true for the TENKTUP and HUNDREDKTUP relations.

3) Compare performance between PostgreSQL and MySQL database systems when queries output to the screen versus written to disk, via a temporary table. Two selectivity levels will be tested.

a. *performance issue we are wanting to test:*

The purpose of this test is to see if there is a performance difference between the two database systems in writing to disk--virtual disk in the case of Google Cloud Services--and keeping the output in memory versus output to the screen.

b. *data sets used:*

The data sets used for this test will be two HUNDREDKTUP tables, with the output being 10% of the relation. Then repeat the test for 20% of the relation.

c. *queries used:*

There will be two variants used for this query: first, with the query written to screen; second, with the query written to disk. Both PostgreSQL and MySQL will be measured for this test.

First set of queries:

```
INSERT INTO temp
SELECT *
FROM HUNDREDKTUP1 H1, HUNDREDKTUP2 H2
WHERE H1.tenPercent = H2.tenPercent

SELECT *
FROM HUNDREDKTUP1 H1, HUNDREDKTUP2 H2
WHERE H1.tenPercent = H2..tenPercent
```

Second set of queries:

```
INSERT INTO temp
SELECT *
FROM HUNDREDKTUP1 H1, HUNDREDKTUP2 H2
WHERE H1.twentyPercent = H2.twentyPercent
```

```
SELECT *
FROM HUNDREDKTUP1 H1, HUNDREDKTUP2 H2
WHERE H1.twentyPercent = H2.twentyPercent
```

d. *metric used:*

For the metric, we will be measuring the preceding query execution times using EXPLAIN (ANALYSE TRUE, TIMING FALSE) (PostgreSQL) and Stackdriver "Trace" (MySQL). Our testing methodology will involve 5 runs of the same query with the results averaged, after removing the high and low values from each testing set.

e. *expected results*

It is expected that MySQL will perform better for the 10% relation output since its general perception for being more optimized for read-heavy workloads. However, with PostgreSQL, the ability to handle concurrency better, so it should handle a larger write load with the 20% output with greater efficiency. It will be interesting to note if there is a significant difference between the two systems. If there is not a significant difference, the output selectivity may be adjusted until differences appear.

4) Increasing complexity.

a. *performance issue we are wanting to test:*

This test is going to measure the performance of the two relational database systems with how they manage queries of different complexity levels. They will first be measured with a simple query and then measured with how they manage a more complex join query.

b. *data sets used:*

The data sets used will be TENKTUP and two HUNDREDKTUP tables.

c. *queries used:*

To measure this experiment, the following queries will be used:

```
SELECT *
FROM HUNDREDKTUP H1, TENKTUP T
WHERE H1.tenPercent = T.tenPercent
```

Second query

```
SELECT * COUNT( *)
FROM HUNDREDKTUP1 H1, HUNDREDKTUP2 H2, TENKTUP T
```

```
WHERE H1.tenPercent  =  H2.tenPercent AND
        H1.evenOnePercent = H2.evenOnePercent AND
        T.uniqe1 < 100 AND H1.unique1 < 100 AND H2.unique1 < 100;
GROUP BY fiftyPercent;
```

d. *metric used:*

For the metric, we will be measuring query execution time using EXPLAIN (ANALYSE TRUE, TIMING FALSE) (PostgreSQL) and Stackdriver "Trace" (MySQL)

e. *expected results:*

It is expected that MySQL will perform better with the less complicated query and PostgreSQL will manage the more complicated query with better efficiency. These are the general performance assumptions for the two systems and this test will see if the differences are visible with this experiment.

Lessons Learned:

There was some difficulty trying to navigate the MySQL documentation looking for exactly what we wanted during the research process (i.e. some concepts were not as cleanly cut and organized as we would've liked them to be). Information on the buffer pool size & structures for both systems were difficult to track down. The prevailing opinion that MySQL works best for online transactions and PostgreSQL is best for append only processes like data warehousing may yield results that do not differ significantly when being tested against Wisconsin Benchmark query variants, but we hope to find some conclusive results in the course of this project.