CS 487/587 Database Implementation
Winter 2021
Database Benchmarking Project - Part 2

Likhitha Vanga
Jaya Bhargavi Vengala

Github link: https://github.com/LikhithaVanga/CS-587-DBImplem

Option 1: **Compare two systems (on the same hardware)**

System-1: PostgreSQL(SQL container GCP)

We used Postgresql**.**We chose this because postgresql is an object relational database and can handle features like table inheritance and function overloading, which can be important to certain applications. Postgres handles concurrency very well. Postgres is known for protecting data integrity at the transaction level and this makes it less vulnerable to data corruption.

Also, we wanted this as an opportunity to learn more about postgresql and gain expertise in this database.

System-2: Big Query(GCP)

BigQuery is Google's serverless data warehouse. BigQuery's "serverless" build, a fully on-cloud design that prioritizes scalability and quickness in queries, means that you can easily scale and perform ad hoc analyses much faster than you would on cloud-based server structures. Even better, its decentralized design means it can perform these queries and derive insights from datasets that stretch into petabyte scale.

It solves this problem by enabling super-fast SQL queries using the processing power of Google's infrastructure.

System Research:

	PostgreSQL	Big Query
Indexing	Indexes are a common way to enhance database performance. An index allows the database server to find and retrieve specific rows much faster than it could do without an index. But indexes also add overhead to the database system as a whole, so they should be used sensibly.	No indexing
Joins	PostgreSQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each. Join types are THE CROSS JOIN, THE INNER JOIN, THE LEFT OUTER JOIN, THE RIGHT OUTER JOIN and THE FULL OUTER JOIN	BigQuery supports ANSI SQL join types. JOIN operations are performed on two items based on join conditions and join type. Items in the JOIN operation can be BigQuery tables, subqueries, WITH statements, or ARRAYs.
Aggregates	PostgreSQL supports	Big Query supports

	aggregate functions. An aggregate function computes a single result from multiple input rows. For example, there are aggregates to compute the count , sum , **avg** (average), max (maximum) and min (minimum) over a set of rows.	aggregate functions like MIN, MAX, COUNT...
Column/Row Store	Row Based database	Column Based database
Foreign Keys	The first **advantage** of the **foreign key** constraint is that the database constantly maintains referential integrity. This means the database monitors the data inserted into the parent and child tables.	N/A
Secondary indices	All **indexes in PostgreSQL** are	N/A

	secondary indexes, meaning that each **index** is stored separately from the table's main data area (which is called the table's heap in **PostgreSQL** terminology). This means that in an ordinary **index** scan, each row retrieval requires fetching data from both the **index** and the heap.	
Partitioning	**Partitioning** refers to splitting a large table into smaller pieces. ... Currently, **PostgreSQL** supports range and list **partitioning** via table inheritance. Basically, you have to create each **partition** as a child table of the master table. Generally, if you want to split data into specific ranges, then use range **partitioning**.	A **partitioned** table is a special table that is divided into segments, called **partitions**, that make it easier to manage and query your data. By dividing a large table into smaller **partitions**, you can improve query performance, and you can control costs by reducing the number of bytes read by a query.
Clustering	**PostgreSQL does** not natively **support** any multi-master **clustering** solution, like MySQL or Oracle **do**. Nevertheless, there are	Clustering can improve the performance of certain types of queries such as queries that use filter clauses and queries that aggregate

	many commercial and community products that offer this implementation, along with others such as replication or load balancing for **PostgreSQL**	data. When data is written to a clustered table by a query job or a load job, BigQuery sorts the data using the values in the clustering columns.

Buffer pool size/structure

PostgreSQL:
For PostgreSQL, the buffer manager contains the buffer table, buffer descriptors, and the buffer pool. The buffer pool is structured as an array. Each slot holds 8KB of data, the same amount as a page in memory. Once data has been loaded into a slot in the buffer pool, PostgreSQL has chosen to use a clock algorithm for the replacement policy of buffer pages. The MySQL InnoDB manages the buffer pool as a list and the buffer pool has a default size of 8MB. MySQL uses a variation of the Least Recently Used algorithm in order to determine which pages will be evicted from the buffer pool.

Big Query:
In BigQuery the allocation of memory is done by arranging different clusters into partitions where the relevant data gets stored. To improve the performance it tries to keep the related data to the same cluster in order to save time from scanning clusters in the different zones. It has a good caching capacity and performs better when the exact query runs later. Total size of the resources depends on the number of partitions and the total number of clusters a partition has. Generally, we get an unbounded resource size using cloud and we can easily increase and decrease the number of resources.

Mechanism for measuring query execution time

PostgreSQL:

Gathering execution plans and time is fairly straightforward for PostgreSQL. PostgreSQL offers a command “EXECUTE ANALYZE” to prepend on any query that will display both the query plan chosen by the optimizer as well as the execution time for each node.

Also **\timing** can be used in the command line psql. The time that \timing returns also includes the network latency, if you're connecting to a remote server.

When you don't want that and don't need the query output too, better use “EXECUTE ANALYZE”, which outputs the query plan with the planner estimates plus the actual execution times.

Big Query:

For BigQuery, we only use the basic output schema which demonstrates the total execution time. When we need to compare performance of BigQuery and MySQL, we compare the total execution time for both systems.

Performance Experiment Design

Create 4 performance experiments that you will run(We are performing 6 performance experiments).

1.Compare the performance when scanning the tuple.

i. This test is to check the performance when scanning the whole table in three different systems(Google Cloud postgreSQL, Google BigQuery).

ii. Use the tables ONEKTUP, TENKTUP1.

iii. We will use queries **SELECT * FROM ONEKTUP, SELECT * FROM TENKTUP1**.

Iv. The parameter (size of the table) changes.

v. The performance order we expect is: for the table with small size, the performance of the three different systems may be very close. But for the table with large size, the performance(best to worst) might be: Google Cloud postgreSQL > Google BigQuery.

2.Compare queries with/without indices.

i. This test is to compare the the performance of queries with/without index in one system and among the three different systems (Google cloud postgreSQL)

ii. Use the table TENKTUP1

iii. We will divide this part into two sections. For the first section: we compare the queries by using a table in one system, that is in the same system, check the performance between using queries with clustered index and the queries without index (use Wisconsin Bench queries 4 and 2) , and between using queries with non-clustered indexes and the queries without index (use Wisconsin Bench queries 6 and 2) . And we also check the performance between using queries with clustered index and the queries with non-clustered indexes (use Wisconsin Bench queries 4 and 6) . Then we switch to other two systems and do the same comparisons as what we do in the first system. For the second section, we compare the performance among the three systems when we are using the same query for the same table.

iv. The parameters change when we are comparing queries in the same system (Option 2), we will need to change the parameters to change if

it is index based query or non-indexed query, or clustered indexed or non-clustered indexed query

v. The performance order we expect is: For the same system, queries with index > without index.

3.Compare different join algorithms across systems.

i. This test is to compare the performance of different join algorithms in the same system and among the three different systems (Google Cloud postgreSQL , Google BigQuery). Note that postgreSQL resolves all joins using a nested-loop join.

ii. Use the tables ONEKTUP, TENKTUP1 and TENKTUP2

iii. We will use the same two or three tables to join in one system and among three different systems (we compare the same join in different systems). Our plan is to do joins between one big table and one small table (use queries such as **SELECT * FROM ONEKTUP, TENKTUP1**
WHERE ONEKTUP.unique1 = TENKTUP1.unique1) , or both tables are big (use Wisconsin Bench queries 15- join between two big tables) or all the three tables (use Wisconsin Bench queries 11- join between three tables) in the same system, then compare the performance. Then do the same queries in other two systems to compare the performance

iv. Parameters change when we use different sizes of tables to do the join in the same system(for Option 2)

v. The performance order we expect is: Joins on more tables may take more execution time. Joins on the tables with similar big sizes may take more time. Join on one small sized table and one large sized table may

perform better. With the same query, the performance in the three different systems will also depend on the sizes of the tables and whether the predicates are indices. If the predicates are indices, postgreSQL should always perform better than BigQuery since BigQuery does not support indices.

4.Compare queries with different selectivities.

i. This test is to compare the performance of different selections in one system or among the three different systems (Google Cloud postgreSQL , Google BigQuery)

ii. Use the table TENKTUP1

iii. First we will use Wisconsin Bench queries 1 (1% selection) and 2 (10% selection), and the query such as **SELECT * FROM TENKTUP1 WHERE unique2 between 792 AND 2791** (20% selection), run those queries in each of the three systems, and check the performance of the queries in the same system, then compare the same query in different systems.

iv. The parameters change when we are comparing queries in the same system (Option 2), we will need to change the parameters by changing the selectivity.

v. The performance order we expect is: For the same system, the queries with smaller selectivity will take less time. For different systems with the same query, when the selectivity is small, the performance(best to worst) may be: Google BigQuery > Google Cloud postgreSQL, but if the selectivity is high, the performance(best to worst) may be: Google Cloud
postgreSQL > Google BigQuery

5.Compare aggregation operations on different systems.

i. This test is to compare the performance of aggregation operations among the three different systems (Google Cloud postgreSQL , Google BigQuery).

ii. Use the table TENKTUP1.

iii. We will use Wisconsin Bench queries 21 (minimum aggregate function with 100 partitions) and 22 (sum aggregate function with 100 partitions), and the query such as **SELECT tenPercent, count(*) FROM TENKTUP1 GROUP BY tenPercent**, run those queries in each of the three systems, and compare the same query in different systems.

iv. No parameters changed in this test.

v. The performance order we expect is: if the predicates are indices, postgreSQL will perform better than BigQuery since BigQuery doesn't support indices. If the predicates are non-index, then the performance will depend on the size of the table, if the size is small, postgreSQL will perform better than BigQuery, if the size is large, BigQuery will perform better. For the performance of local MySQL and Google Cloud postgreSQL , Google Cloud postgreSQL will be better.

6.Compare insertion, deletion and update operations in different systems.

i. This test is to compare the performance of insertion, deletion and update operations to the same table in the three different systems (Google Cloud postgreSQL , Google BigQuery).

ii. Use the table TENKTUP1.

iii. We will use Wisconsin Bench queries 26 (insert one tuple) and 27 (delete one tuple), and Wisconsin Bench queries 28 (update key attribute) and 32 (update indexed non-key attribute), run those queries in each of the three systems, and compare the same query in different systems.

iv. No parameters changed in this test.

v. The performance depends on the predicates whether they are indices or non-indices. If they are indices, the performance order will be: Google Cloud postgreSQL > Google BigQuery. If they are non-indices, the performance order will be: Google BigQuery > Google Cloud postgreSQL.

Lessons Learned

- We have learned about the properties of the systems that we selected. Knowing the pros and cons of each system will help us to design queries that will better suit the features of the systems. Optimizing the query plan is important for obtaining a better query that takes the advantage of all privileges that will help us to get the result faster and improve performance. We became familiar with how joins work and what factors drive a system to choose a better join plan for the algorithm.

- While doing some experiments we observed that when we scan tables with less tuples then BigQuery performs better or almost similar to MySQL. In some cases it also depends on the workload on the cloud which might delay the processing time.

- PostgreSQL supports indexes. The queries with the predicate conditions matched with the index will always perform better. Since BigQuery does not support index it might take longer time to process based on the selectivity factor.

- We learned that postgres provides us ways to turn on and off various index options that are available from which we can actually understand the usage of each of the indexes being present.

PostgreSQL and BigQuery both have their advantages and disadvantages. We will try to explore all the factors that we have considered and implement it to see the results and compare with our assumption. Performing the experiments will help us to get a deeper understanding of how a system behaves and what query algorithm it selects to perform better.