Santiago Tobon & Nhan Le
Dr. Kristin Tufte
CS487/587

DB implementations Project Part 2

Option 1: Comparing Postgres and BigQuery

System Research: Postgres

1. *Types of Indices:*

Postgres has multiple different index types. These index types are B-tree, Hash, GiST, SP-GiST, GIN, and BRIN. B-trees are self-balancing trees that maintain sorted data and allow searches, insertions, deletions, and sequential access in logarithmic time. Hash indexes only handle simple equality comparisons (=). GIN indexes are generalized inverted indexes that are best for when you have multiple values stored in a single column. BRIN indexes are block range indexes that allow the use of an index on really big tables and are often used on a column that has a linear sort order. GiST indexes are generalized search trees that allow a building of general tree structure. SP-GiST indexes are space-partitioned GIST indexes that support partitioned search trees that are great for a wide range of non-balanced data structures.

2. *Types of Join Algorithms:*

Postgres supports three kinds of joins, nested Loop join, hash join, and merge join. Nested loop join is where in each record of outer relation is matched with each record of the inner relation. Hash join where a hash table is built using the inner relation records and the outer relation is hashed based on the join clause key. Merge join is an algorithm where each record of outer relation is matched with each record of inner relation until there is a possible join clause match.

3. *Buffer pool size/structure:*

The buffer pool is a simple array that stores data file pages like tables and indexes. The buffer pool slot size is 8KB. So each slot can store an entire page.

4. *Mechanisms for measuring query execution time:*

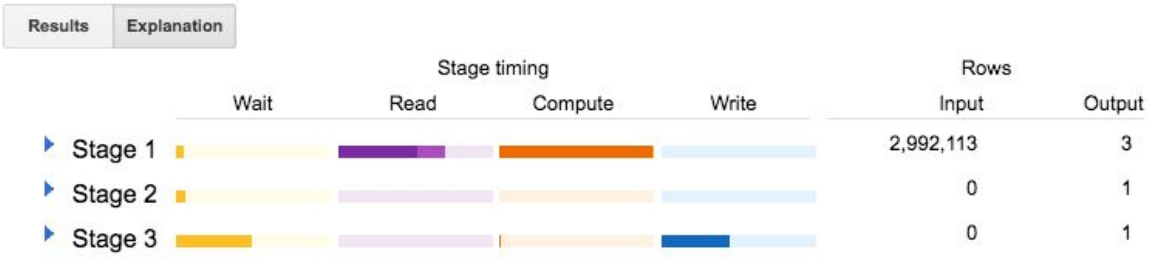
In Postgres's command line client there is a \timing feature that measures query time. It prints out the time in milliseconds after executing a query.

System Research: BigQuery

1. Types of Indices:

Since BigQuery doesn’t support the indices, each time the BigQuery executes the query, it scans the full column. The BigQuery performance and query costs are based on the amount

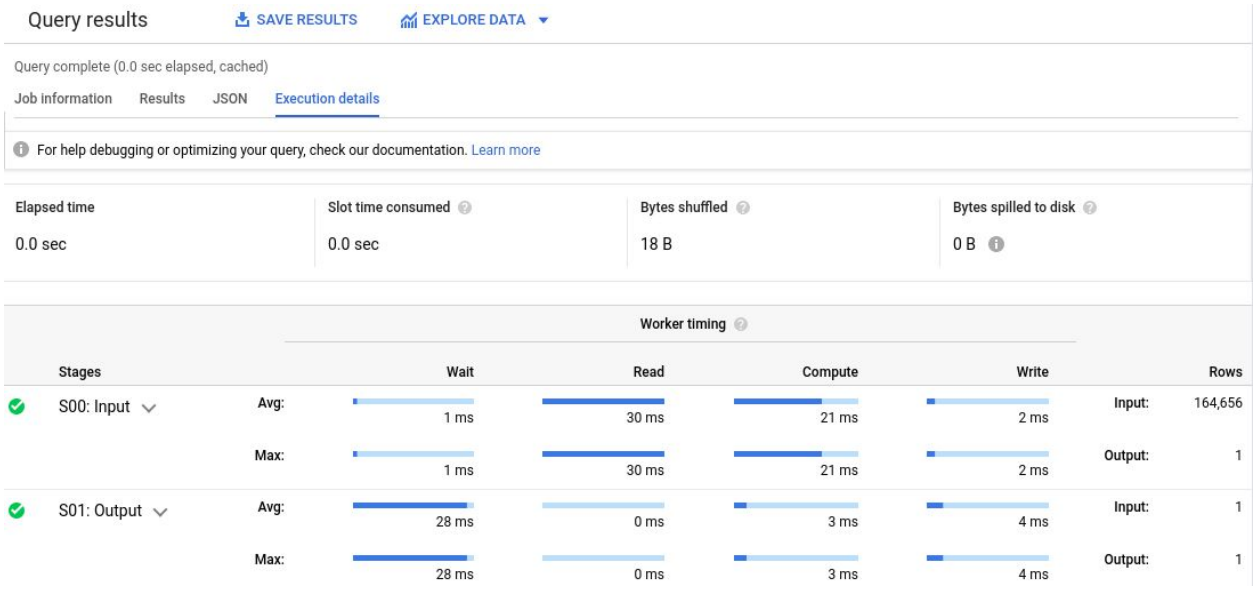
of data scan during the query. So, design the queries in the way that they reference the only column relevant to the query. In order to understand the performance characteristics after a query executes, we take a look at the query plan explanation.



When BigQuery executes a query, it converts the declarative SQL statements into a graph of execution like a picture above, broken down into a series of query stages. Stage model the units of work that many potential workers may execute in parallel. In addition to the query plan, query jobs also expose a timeline of execution which provides the units of work completion, pending, and active within the query workers.

2. Measuring the execution time:

   In the Cloud Console, we can see details of the query plan for a completed query by clicking the Execution Detail.



   In this picture, the color indicators show the relative timings for all stages. The stage S01 is waiting for the Stage S00 to complete. This is because it is dependent on Stage S00's input and it could not start until S00 completes to write the output into intermediate shuffle.

3. Type of join algorithm:

BigQuery supports ANSI SQL join types. JOIN operations are performed on two items based on join conditions and join type. Items on JOIN operation can be BigQuery tables, subqueries, WITH statements or ARRAY.

Broadcast joins: is used to join a large table to a small table. The BigQuery creates a broadcast join where the small table is sent to each slot processing the large table. The best practice is to place a largest table first, followed by the smallest and then by decreasing size.

Hash joins: is used to join two large tables. BigQuery uses a hash and shuffle operations to shuffle the left and right tables so that the matching keys end up in the same slot to perform a local join. This is an expensive operation because the data needs to be removed.

Self Joins: a table is joined with itself. This can be an expensive operation for large tables. Also, it may be required to get data in more than one pass.

Skewed joins: Data skew can occur when data is partitioned into unequally sized partitions. To avoid this problem,we need to pre-filter data from the table as early as possible.

4. Buffer pool size/structure:

In terms of BigQuery memory, BigQuery doesn't use the Buffer Pool, but instead it uses a cache. BigQuery writes all query results to a table. The table is identified by the user (is called the destination table), or the temporary, cached results table. Temporary, cached results tables are maintained per-user or per-project. Since there are no costs for temporary tables, but when we write query results to permanent tables, we are charged for storing the data. By the way, all query results, including both interactive and batch queries, are cached in a temporary memory table for approximately 24 hours with some exceptions.

Performance Experiment Design:

1. Compare join algorithm performance
    a. Experiment specifications
        i. This experiment compares the join algorithm performance on BigQuery and Postgres.
        ii. We are going to use a 10,000 tuple relation and 100,000 tuple relation.
        iii. We are going to test with the following two queries in both Postgres and BigQuery:
            1. INSERT INTO TMP
               SELECT * FROM HUNDREDKTUP1, BPRIME

WHERE (HUNDREDKTUP1.unique2 = BPRIME.unique2)

2. INSERT INTO TMP
   SELECT * FROM TENKTUP, HUNDREDKTUP1
   WHERE (TENKTUP.unique2 = HUNDREDKTUP1.unique2)
   AND (HUNDREDKTUP1.unique2 = HUNDREDKTUP2.unique2)
   AND (HUNDREDKTUP1.unique2 < 10000)

We will compare the execution time of both queries with both systems.

iv. n/a

v. Results expected: We expect that the two queries will be faster on the Postgres tables than on the BigQuery tables.

2. Test 10% selection via a clustered index
   a. Experiment specification
      i. The issue is that BigQuery scans full columns; otherwise, Postgres can also use a clustered index to get the result. This experiment is good to compare the performance of using the clustered index on Postgres and BigQuery.
      ii. Use a 10,000 tuple relation (scale up version TENKTUP1)
      iii. This query will run a clustered index on two systems and compare the performance.

```
INSERT INTO TMP
SELECT * FROM TENKTUP1
WHERE unique2 BETWEEN 792 AND 7910
```

      iv. No parameter has been set.
      v. Results expected: We expect that a query will be faster on BigQuery than on Postgres.

3. Update key attribute
   a. Experiment specification
      i. The experiment on two systems will compare the performance of updating the piece of data. Since, the BigQuery doesn't support the indices then it

may be an issue on performance. It's a good time to test the performance whether it runs slower or faster on BigQuery than on Postgres.

ii. Use a 100,000 tuple relation (HUNDREDKTUP1)

iii. Query:

```
UPDATE HUNDREDKTUP1
SET unique2 = 100001 WHERE unique2 = 97001
```

iv. n/a

v. Result expected: We would expect that it will run faster on Postgres than on BigQuery.

4. Compare aggregate functions COUNT & SUM performance
    a. Experiment specification
        i. This experiment compares the COUNT & SUM functions performance on BigQuery and Postgres.
        ii. We are going to use a 10,000 tuple relation.
        iii. Query:
            1. INSERT INTO TMP
               SELECT SUM (TENKTUP1.unique3) FROM TENKTUP1
               GROUP BY TENKTUP1.onePercent
            2. INSERT INTO TMP
               SELECT COUNT (TENKTUP1.unique3) FROM TENKTUP1
               GROUP BY TENKTUP1.onePercent
        iv. n/a
        v. Results expected: We expect postgres to perform faster than BigQuery for both COUNT & SUM.

Lessons Learned:

Postgres:

What I have learned about Postgres:

1. Postgres supports 6 different types of indexes, B-tree, Hash, GiST, SP-GiST, GIN, and BRIN.
2. Postgres supports 3 types of join Algorithms, nested Loop join, hash join, and merge join.

3. There isn't a ton of documentation for timing queries in postgres.

Issues: When working with postgres on gcp, I ran into trouble inserting the data into the table. I realized that the csv file couldn't have headers when inserting it. Also didn't have the benchmark data scaled properly during part 1 of the project. Had to get help from the TA to fix our data generator.

BigQuery:

What I have learned from BigQuery:

1. BigQuery doesn't support the indices. Yet, it has supported the clustered column.
2. When the data is loaded to BigQuery, charges are based on the amount of data stored in the tables per second.
3. Use BigQuery data for analytics to satisfy business use cases like predictive analysis, real time inventory management…
4. Use the caching instead of the buffer pool.
5. Learn some different algorithms like skew join, hash join, self join and broadcast join to improve the performance.

Issues: n/a

Github: https://github.com/santitobon9/487DB-Project

Citation:

Query Optimization - BigQuery for data warehouse practitioners | Solutions | Google Cloud
JOIN- How to perform joins and data denormalization with nested and repeated fields in BigQuery | Google Cloud Blog
Cached - Using cached query results | BigQuery | Google Cloud
Measure execution time - Query plan and timeline | BigQuery | Google Cloud

https://www.postgresqltutorial.com/postgresql-indexes/postgresql-index-types/
https://severalnines.com/database-blog/overview-join-methods-postgresql
https://www.interdb.jp/pg/pgsql08.html#:~:text=The%20buffer%20pool%20is%20a,can%20store%20an%20entire%20page.
https://stackoverflow.com/questions/9063402/get-execution-time-of-postgresql-query/9064100