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

Due: Tues February 23, midnight (end of the day on the 23rd), D2L

Team: Swetha Venkatesan, Sai Deepika Gooty

For the design benchmark, we have chosen to evaluate PostgreSQL with different parameter values and optimizer options.

SYSTEM CONFIGURATION

1. **Local Machine**
   Configuration - 16GB RAM, Windows 10 operating system and Postgres with Psql version 13.1

2. **Google Cloud Platform VM**
   Configuration - 2GB RAM, Ubuntu operating system and Postgres with Psql version 12.6

SYSTEM RESEARCH

1. **Postgres Query Planner Configs**

   1.1 enable_hashjoin (boolean)
   1.2 enable_nestloop (boolean)
   1.3 enable_mergejoin (boolean)

2. **Postgres Memory Options**

   2.1 temp_buffers (integer)
   2.2 shared_buffers (integer)
   2.3 work_mem (integer)

PERFORMANCE EXPERIMENTS

1. Testing the 10% rule of thumb

- The relations used for this experiment are 1,000 tuples (ONEKTUP), 10,000 tuples (TENKTUP1) and 100,000 tuples (HUNDREDKTUP).
- We are taking into consideration the 2, 4 and 6 queries from Wisconsin Benchmark

Query 2

INSERT INTO TMP SELECT * FROM TENKTUP1 WHERE unique2
BETWEEN 792 AND 1791
Query 2 is run without an index on unique2

Query 4

INSERT INTO TMP SELECT * FROM TENKTUP1 WHERE unique2
BETWEEN 792 AND 1791
Query 4 is run with a clustered index on unique2

Query 6

INSERT INTO TMP SELECT * FROM TENKTUP1 WHERE unique1
BETWEEN 792 AND 1791
Query 6 is run with an unclustered index on unique1

- We will also compare the query selection for 5, 25 and 50 percent.
- Datasets and system parameters remain constant while the queries vary.

Expected Result

The performance of query 2 with lesser query selectivity is expected to show better results when compared to the queries with greater query selectivity.

2. **Testing Parameter - shared_buffers (integer), enable_hashjoin (boolean), enable_nestloop (boolean), enable_mergejoin (boolean)**

- This experiment is to demonstrate the performance of the shared buffer based on its size for different queries.

- The relations used for this experiment are 10,000 tuples (TENKTUP1), 100,000 tuples (HUNDREDKTUP), and 1,000,000 tuples (MILLIONTUP)

- We will be testing with different shared buffer sizes such as 50MB, 100MB, and 150MB with 2GB Postgres RAM

Queries

Hash Join Query

```
SELECT h.unique1, h.string1, m.string1
FROM hundredktup as h
JOIN milliontup as m
ON h.unique1 = m.unique1
WHERE h.string like 'AAAAI%'
```

Nested Loops Join Query

What the "Nested Loops" operator basically does is: For each record from the outer input – find matching rows from the inner input.

```
SELECT t.twenty, h.unique1, h.string1
FROM tenktup1 as t
JOIN hundredktup as h
ON t.unique1 = h.unique1
WHERE t.twenty BETWEEN 1 AND 19;
```

Merge Join

The "Merge" algorithm is the most efficient way to join between two very large sets of data which are both sorted on the join key.

```
SELECT t1.unique3, t1.four, t2.string1
FROM tenktup1 as t1
JOIN tenktup2 as t2
ON t1.unique3 = t2.unique3
```

Aggregate

```
SELECT COUNT(*) AS number_of_rows
FROM tenktup1 as t1
INNER JOIN tenktup2 as t2
ON t1.unique1 = t2.unique1
```

Expected Result

Based on the Nested Loops / merge join logic, We are expecting the PostgresSQL to opt nested loops / merge join join over hash join. Better results are observed with low memory on join and aggregate queries.

3. **Testing Sizeup**

- This experiment is to demonstrate the performance when aggregate function is applied to different relations In this test we see how the same aggregate function takes different execution time for different relations

- The relations used for this experiment are 1,000 tuples (ONEKTUP), 10,000 tuples (TENKTUP1), 100,000 tuples (HUNDREDKTUP), and 1,000,000 tuples (MILLIONTUP)

Queries

Query 1 - sum of twenty

```
SELECT sum(twenty) as sum_onektup
FROM onektup

SELECT sum(twenty) as sum_tenktup
FROM tenktup1

SELECT sum(twenty) as sum_hundredktup
FROM hundredktup

SELECT sum(twenty) as sum_millionktup
FROM milliontup
```

Query 2 - average of ten

```
SELECT sum(ten) as sum_onektup
FROM onektup

SELECT sum(ten) as sum_tenktup
FROM tenktup1
SELECT sum(ten) as sum_hundredktup
```

```
FROM hundredktup

SELECT sum(ten) as sum_millionktup
FROM milliontup
```

- Parameter are not changed for this experiment

Expected Result
In this testing, as the size of the relation increases the query execution time should increase, even though the query is aggregate which returns only one tuple in every case.

4. **<u>Testing work_mem</u>**

- This experiment is to demonstrate the collaboration of the work_mem containing different sizes with different sort operations

- The relations used for this experiment are 10,000 tuples (TENKTUP1), 100,000 tuples (HUNDREDKTUP)

  Queries

  Query 1

```
SELECT  DISTINCT string1
FROM tenktup1
ORDER by ten
```

  Query 2

```
SELECT  t2.string1
FROM tenktup1 as t1
JOIN tenktup2 as t2
On t1.unique1 = t2.unique1
ORDER by t1.twenty
```

- work_mem parameter is changed for this experiment.

Expected Result
Query performs increases with an increase in the work_mem

5. **Testing Parameter - temp_buffers**

- This experiment is to demonstrate the suitable temp buffer size for different table joins such as sort and hash

- The relations used for this experiment are 10,000 tuples (TENKTUP1), 100,000 tuples (HUNDREDKTUP), and 1,000,000 tuples (MILLIONTUP)

  Queries

  Query 1

  ```
  SELECT t.unique1, h.unique1
  FROM tenktup1 as t, hundredktup as h
  WHERE t.string1 = t.string1;
  ```

  Query 2

  ```
  SELECT h.unique1 ,m.unique1, m.twenty
  FROM hundredktup as h, milliontup as m
  WHERE h.string3 = m.string3
  AND m.twenty  BETWEEN  2 AND 18
  ```

  Query 3

  ```
  SELECT t2.twenty, t1.unique3, t1.string1, t1.twenty
  FROM tenktup1 as t1
  JOIN tenktup2 as t2
  ON t1.unique3 = t2.unique3
  WHERE t1.twenty BETWEEN 5 AND 12
  ORDER BY t2.twenty
  ```

- temp_buffer sizes varies in this experiment with 4 MB, 8 MB, and 16 MB

Expected Result
Higher temp_buffer size should perform better.

LESSONS LEARNT

We have learned about the different properties of the Postgres System that we have selected for testing. Knowing the advantages and disadvantages of the system will help us to design queries that will better suit the features of the Postgres. 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. Also, we have become familiar with how different types of joins work and what factors drive a system to choose a better join plan for the algorithm.

We learned that postgres provides different methods to turn on and off various index options that are available FROM which we can clearly understand the usage of each of the indexes.

We were unable to set shared_buffers size while running psql using command *'set shared_buffers="256"'.* We got error message parameter "shared_buffers" cannot be changed without restarting the server. This is because the shared buffers is the center of the database system, it is only possible to set up FROM a config file and need to restart the server after changing the size.

REFERENCE

1. https://www.postgresql.org/docs/9.6/runtime-config-resource.html
2. https://eitanblumin.com/2018/10/28/the-loop-hash-and-merge-join-types/