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

Team members: Deep Patel Goutham Podili

GitRepo: https://github.com/GouthamC7/Database_Implementation/blob/main/DBImpl_part2.pdf

----

In the experiment, we are evaluating PostgreSQL with different memory options and query planner configuration in Postgres server on Google Cloud VM and local computer.

SYSTEM CONFIGURATION:

The memory level configuration of the system is mentioned below.

Machine type	RAM	OS Image	Postgres version
Google VMs	2 GB	Ubuntu-2004	Psql 12.6
Local	16GB	Windows10	Psql 13.1

SYSTEM RESEARCH:

In our research, we have encountered below postgres memory options and query planner options.

`shared_buffers(integer)`	● A database server also needs memory for quick access to data, whether it is READ or WRITE access. In PostgreSQL, it is referred to as shared buffers. ● Sets the amount of memory the database server uses for shared memory buffers. ● Default is typically 128 MB. ● Settings significantly higher than the minimum are usually needed for good performance
`work_mem(integer)`	● Specifies the amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files. ● Default is 4MB ● It is ideal to set lower *work_mem* for short queries

	that run very frequently and perform simple lookups and joins. ● However, if we have a lot of complex sorts, and have a lot of memory, then increasing the *work_mem* parameter allows PostgreSQL to do larger in-memory sorts which are faster than disk-based. ● Since *work_mem* on postgres server is used by multiple users if 50 MB memory use by 30 users then the total memory being used is 1.5 GB. Thus, it is important to consider the maximum number of connections allowed on the server. ● Another scenario would be if merge sorts of 5 tables, requires 5 times work_mem. ● The optimal work_mem can be count by below formula Work_mem = Total RAM * 0.25 / max_connections ● To experience *work_mem* performance we will be testing sort operations like ORDER BY and merge join
`temp_buffers(integer)`	● In PostgresSQL this memory area used to hold the temporary tables of each session, the memory will be cleared when the connection is closed. ● Sets the maximum number of temporary buffers used by each database session. ● The default value of temp_buffer is 8MB.
`enable_seqscan(boolean)`	• It is impossible to suppress sequential scans entirely, but turning this variable off discourages the planner from using one if there are other methods available. • Default is on
`enable_hashjoin(boolean)`	● Enables or disables the query planner's use of hash-join plan types. ● Default is on.

PERFORMANCE EXPERIMENTS:

1) **Testing the 10% rule of thumb**
    a) This test explores when it is good to use an unclustered index vs not using an index vs using a clustered index. We will also vary the selectivity of queries like 5%, 25%, and 50%.

b) We are using relations with size of 10,000 tuples(TENKTUP1) and 100,000 tuples(HUNDREDKTUP).
c) We are using Wisconsin benchmark queries 2, 4 and 6
    i) INSERT INTO TMP SELECT * FROM TENKTUP1 WHERE unique2 BETWEEN 792 AND 1791 **(no index)**
    ii) INSERT INTO TMP SELECT * FROM TENKTUP1 WHERE unique2 BETWEEN 792 AND 1791 **(clustered index)**
    iii) INSERT INTO TMP SELECT * FROM TENKTUP1 WHERE unique1 BETWEEN 792 AND 1791 **(non-clustered index)**
d) No parameters are changed in this test.
e) We are expecting that for queries which has selectivity less than ten percent attributes which have no index will perform better than attributes with index.

2) Testing work_mem

a) This test examines the relationship between the sort operations and work_mem. We will set work_mem to different sizes.
b) We are using relations with size of 10,000 tuples(TENKTUP1) and 100,000 tuples(HUNDREDKTUP).
c) The queries we are going to execute for this experiment are
    i) SELECT unique1, strin3 FROM TENKTUP1 ORDER BY string3.
    ii) INSERT INTO TMP SELECT * FROM ONEKTUP, TENKTUP1 WHERE (ONEKTUP.unique1 = TENKTUP1.unique1) AND (TENKTUP1.unique1 = TENKTUP2.unique1) AND (TENKTUP1.unique1 < 1000)
d) We will change the work_mem parameter in this test.
e) As the sorting uses work memory we are expecting to see an increase in the performance of the query when increasing the work_mem.

3) Testing shared_buffer parameter

a) This test illustrates what size of the shared buffer performs well on given queries. We will be testing with 25 MB, 128 MB, and 256 MB.
b) We consider table size of 10,000 tuples(TENKTUP1), 100,000 tuples(HUNDREDKTUP), and 1,000,000 tuples(ONEMILLIONTUPLES)
c) Queries:

    i) Join query
    SELECT onemilliontuples.unique1 , hundredktuples.unique1
    FROM onemilliontuples, hundredktuples
    WHERE onemilliontuples.string3 = hundredktuple.string3;
    ii) Aggregate query
    SELECT COUNT(DISTINCT unique1)
    FROM TENKTUP1
d) We will be testing on 25 MB, 128 MB, and 256 MB. Total RAM size of Postgres server is 2 GB
e) It is recommended to set 25% of RAM for buffer size.

When performing join and aggregate queries with different memory sizes, relatively low memory will provide better results.

4) **Testing temp_buffers parameter**

a) This test demonstrates what size of temp buffers is well suited for sort and hash table joins operations queries. We will be testing 4 MB, 8 MB, and 16 MB.
b) We consider table size of 10,000 tuples(TENKTUP1), 100,000 tuples(HUNDREDKTUP), and 1,000,000 tuples(ONEMILLIONTUPLES)
c) Queries:
   i) SELECT TENKTUP1.unique1, HUNDREDKTUP.unique1
      FROM TENKTUP, HUNDREDKTUP.
      WHERE TENKTUP1.string3 = HUNDREDKTUP..string3;
   ii) SELECT HUNDREDKTUP.unique1 , ONEMILLIONTUPLES.unique1
      FROM HUNDREDKTUP, ONEMILLIONTUPLES
      WHERE ONEMILLIONTUPLES.string3 = HUNDREDKTUP.string3;
d) The experiment will be held on 4 MB, 8 MB, and 16 MB temp_buffer size
e) Having a bigger temp_buffer size increase the performance of sort and hash table operations.

LESSON LEARNED:

1) Unable to set shared_buffers size on running psql using command

```
set shared_buffers="256";
Error: parameter "shared_buffers" cannot be changed without
restarting    the server
```

   Since shared buffers are the heart of the database system, it only possible to set up from config file and need to restart the server after changing the size.
2) We got to know about the different memory options and query planner configurations available for Postgres.
3) We are now familiar with variable types of join algorithms and what factors drive a system to choose a better join algorithm.
4) Optimizing the query plan is very important for getting faster results and improved performance.
5) We ran sample queries on ONEKTUP and TENKTUP, there is not much difference in the run times, so we decided to scale up the tables. We created two more tables with 1,00,000 and 10,00,000 tuples.
6) We got to know about Explain Analyze which when used with the queries prints information about the algorithms used and the time took to execute the query.

REFERENCES

https://www.postgresql.org/docs/9.6/runtime-config-resource.html

https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server

https://severalnines.com/database-blog/architecture-and-tuning-memory-postgresql-databases