CS-587 DATABASE IMPLEMENTATION

DATABASE BENCHMARKING

Sukanya Kothapally
Sowmya Sri Indukuri

System - PostgreSQL

- The reason for us to choose postgresql is we are familiar with it and have user friendly interface.
- It can also run parallel queries and offers advanced cost-based query optimization.
- PostgreSQL allows you to define your own data types, index types,etc.
- PostgreSQL has many systems and query parameters, which can be adjusted according to the data being fetched and this is very much useful in this project.

System - PostgreSQL

- If you don’t like any part of the system, you can always develop a custom plugin to enhance it to meet your requirements e.g., adding a new optimizer.
- PostgreSQL can handle a lot of data.

Data Size

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250 - 1600 depending on column types
Maximum Indexes per Table	Unlimited

Goals

- Learn how to design a benchmark and make it scalable.
- Deep understanding of how postgresql chooses which join algorithms according to the query to reduce the cost.
- Getting hand-on experience in tuning different parameters for queries and how they behave with different table sizes/queries/selectivity/indices.

Parameters Chosen

1. Work_mem
2. Enable_Hashjoin
3. Enable_Seqscan
4. Enable_Mergejoin

Experiment 1- work_mem

Test: Evaluates the performance of the query planner when given different values for work_mem.

Results expected:

According the behaviour of work_mem when there is large relation and when the work_mem is given higher values we expect in less disk-swapping and therefore, we might get better performance.

Experiment 1 - work_mem

Query 1:

explain analyze select distinct u.stringu4 from HUNDREDKTUP u, HUNDREDKTUP v

where u.two = v.two and u.unique3 < 1000 and v.unique3 >1

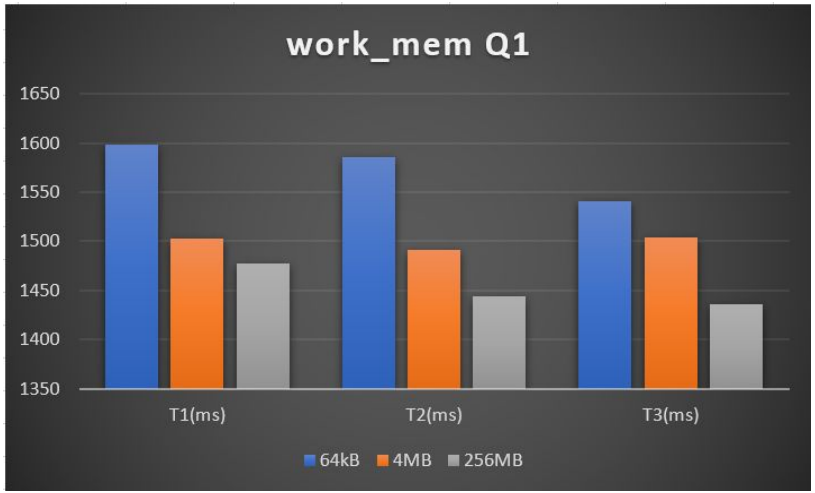
Query 2:

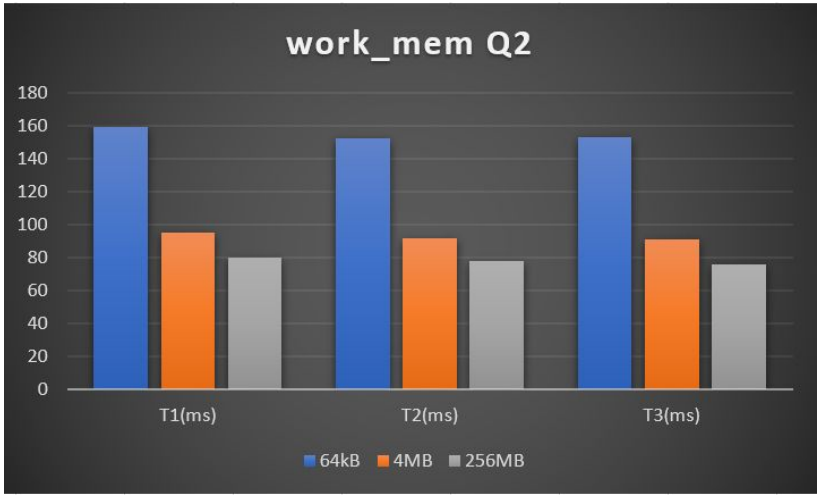
explain analyze select two,four,ten from HUNDREDKTUP order by string4;

Datasets: *100KTUP*

Experiment 1 - work_mem

Results for 100KTUP





Experiment 2- Seq scan

Test:

Here we toggle with Enable_seqscan parameter and evaluate the performance.

Results Expected:

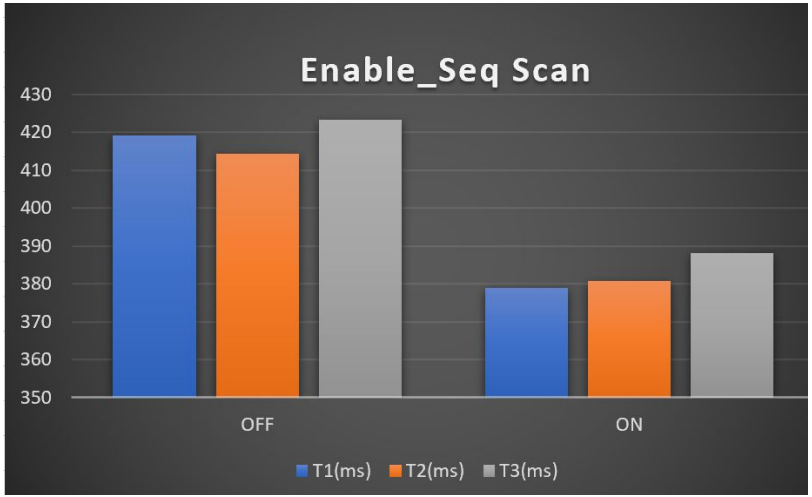
When the sequential scan is toggled, according to the behaviour it should sequentially iterate through the table a row at a time and then returns the rows requested in the query. If it is not toggled it would randomly iterate through the table which takes more execution time.

Experiment 2

Query:

Explain analyze select u.stringu4
from "THOUSANDKTUP" u where u.unique2 in
(select v.unique2 from "HUNDREDKTUP" v
where v.unique2 < 10000) ;

Datasets: *1Million, 100K*



Results:
Results are as expected. When the table sizes are different and there is a nested query, query planner chooses sequential scanner irrespective of the nested loop.

Conclusions

- Higher work_mem value may make complex queries faster if it allows us to fit all the temp data for the query into memory. However, one should use it carefully because sometimes increasing the value too much may cause out of memory errors on your database server.
- In general postgres prefers hash join over others and has less execution time when compared to other other join algorithms.
- Sequential scan is not suppressed completely even when it is off, it runs in the background.
- If the join method chosen by the optimizer is not efficient, then the configuration parameters can be switch-off to force the query optimizer to choose a different kind of join methods which performs better.

Lessons Learned

- Understood how postgres query optimizer will identify which plan to choose by looking into the cost of each plan.
- As work_mem can result in less disk-swapping, increase in work_mem is of no use if we have small relations.
- Even after tuning the parameters, the optimizers do not work the way we expect it to work. So, instead of toggling the parameters,we tried to rewrite the same query in a slightly tweaked way which forces the optimizer to choose a better plan.
- Clearing Shared_buffers is very important as the previous run pages remains in the buffer pool and gives us misleading results during the second run of the same experiment.

References

- https://www.compose.com/articles/what-postgresql-has-over-other-open-source-sql-databases/
- https://severalnines.com/database-blog/overview-various-scan-methods-postgresql
- https://www.cybertec-postgresql.com/en/join-strategies-and-performance-in-postgresql/
- https://severalnines.com/database-blog/overview-join-methods-postgresql

APPENDIX

Experiment 3-Merge Join

Test:
Here we toggle with enable_mergejoin parameter and evaluate the performance.

Results Expected:

This join algorithm is only used if both relations are sorted and the join clause operator is “=”. when we set enable_mergejoin parameter to on and find the time the query planner uses the merge join algorithm. When it is set to off it should use another algorithm (maybe nested loop join or hash join).

Experiment 3- Results

Query:

Set Enable_mergejoin=OFF;
Explain analyze Select u.string4
from "THOUSANDKTUP" u, "THOUSANDKTUP" v
where u.two = v.two and u.two < 1000;

DataSet: *100K*

Results:

When the merge join is not toggled the query planner have chosen nested loop join which increased the time compared to that of when the merge join is toggled.

Experiment 4 - Hash join

Test

Here we toggle with enable_hashjoin parameter and evaluate the performance.

Results Expected:

Hash Join works best when we have a small relation that can fit into the memory and another relation which is extremely big and that doesn't fit into the memory. When the enable_hashjoin is on we could expect significantly better performance than when it is set off. When hash join is disabled, postgres is more likely to use nested loop join or merge join operator and hence taking more time.

Experiment 4 - Results

Query:

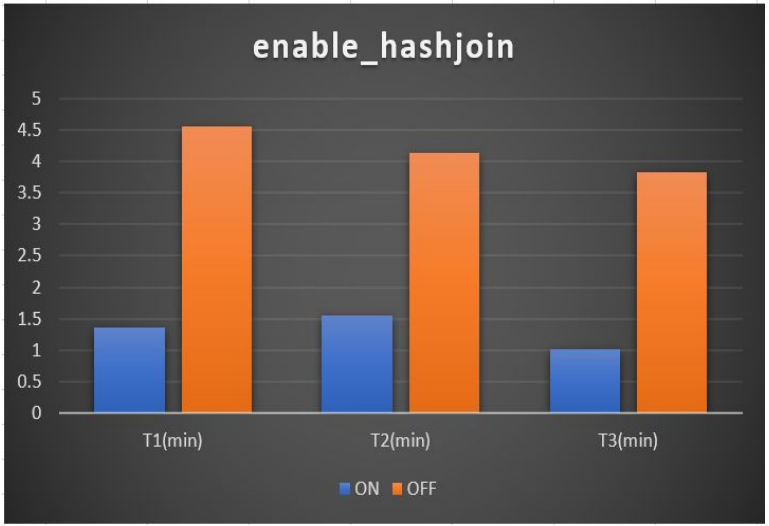
*Explain analyze select **
from TENKTUP u, HUNDREDKTUP v
where u.two = v. two;

Datasets:

One relation with *10k tuples* and another with *1 Lakh* tuples.

Results:

Results are as expected. After running the queries it was found that when hash join is on query planner opted for Hash join and when it's off query planner chose merge join.



Experiment 4 - Query Planner

Hash Join -> ON

	QUERY PLAN text
1	Hash Join (cost=529.00..5629810.00 rows=5000000...
2	Hash Cond: (v.two = u.two)
3	-> Seq Scan on "HUNDREDKTUP" v (cost=0.00..403...
4	-> Hash (cost=404.00..404.00 rows=10000 width=2...
5	Buckets: 16384 Batches: 1 Memory Usage: 2502...
6	-> Seq Scan on "TENKTUP" u (cost=0.00..404.00 ...
7	Planning Time: 47.316 ms
8	JIT:
9	Functions: 10
10	Options: Inlining true, Optimization true, Expressions ...
11	Timing: Generation 1.831 ms, Inlining 296.997 ms, O...
12	Execution Time: 90091.515 ms

Hash Join -> OFF

	QUERY PLAN text
1	Merge Join (cost=22591.11..7524126.08 rows=500...
2	Merge Cond: (u.two = v.two)
3	-> Index Scan using idx80 on "TENKTUP" u (cost=0...
4	-> Materialize (cost=22590.82..23090.82 rows=10...
5	-> Sort (cost=22590.82..22840.82 rows=10000...
6	Sort Key: v.two
7	Sort Method: external merge Disk: 21672kB
8	-> Seq Scan on "HUNDREDKTUP" v (cost=0....
9	Planning Time: 0.202 ms
10	JIT:
11	Functions: 7
12	Options: Inlining true, Optimization true, Expression...
13	Timing: Generation 1.252 ms, Inlining 2.926 ms, Opt...
14	Execution Time: 234457.038 ms

THANK YOU!!!

Questions?