CS 587 Database Implementation

Database Benchmarking Project Part III

Janaki Raghuram Srungavarapu
Jetha Harsha Sai Emandi

System Overview

- The database system that we used for this project is PostgreSQL. We used PostgreSQL in our previous Introduction to Database Management class and thus have a fair knowledge of the system. It also has rich Documentation. It is the most advanced open source database management system. Also, the PostgreSQL has a very interactive community which can help us in guiding if we ever run into any difficulties during the project.
- One advantage is that it is robust in nature with high performance and multitasking. Scaling is another advantage in PostgreSQL. Since it is an open source database management system, there are a lot of active users who develop and propose various new modules to the community.
- Postgres gives immense flexibility to play around various memory parameters and Planner Cost Constants which is much needed for this project.

Tested parameters

1. Work_mem: This memory component indicates the measure of memory to be utilized by internal sort activities and hash tables creations. The default work_mem is postgres is set to 4 MB. Sort operations are used for ORDER BY, DISTINCT queries

2. Enable_seqscan: Sequential scan is one of the basic scan operations of the database for retrieving data on disk. Sequential scan is always conceivable. Regardless of the schema of the table or the presence of indices on the table, sequential scan always returns data. This parameter cannot be totally turned off. Setting this parameter off only encourages query optimizer to use other types of scans such as index and bitmap.

3. Enable_hashjoin: Hash join is mostly used on tables that are not already sorted on the join columns. The optimizer builds an in-memory hash table on the inner tables join column using work_mem . The optimizer then scans the outer table for matches to the hash table. The cost of performing a hash join highly depends on whether or not the hash table fits in the memory. Cost rises significantly if the hash table has to be written to disk.

4. Shared_buffers: This is the actual amount of memory allocated to postgresql page queue. The default memory that postgres sets is 128MB.This could be configured to higher or lower sizes based on the available RAM and other OS requirements.

Project objectives

- To learn how different memory parameters - Work_mem, Shared_buffers affect query execution time.
- To learn how sequential scan has an effect on the query execution runtime.
- To evaluate join algorithms (Nested loop and hash join) and their query execution times.
- To understand effect of other system parameters like OS, CPU on postgres query execution times .

Experiment - 1

Query:

```
SELECT unique2, two, string4  FROM fiftyktup ORDER by string4;
```

Parameter Tested
Work_mem varied from 4 MB to 50 MB
Relation size is kept constant 10 MB

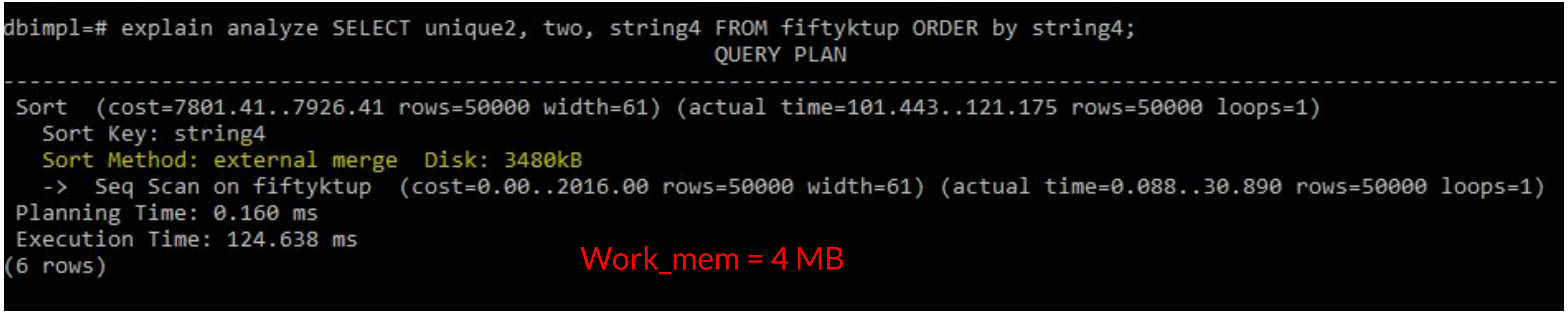
Performance Issue tested

This test explores the relation betweens the work_mem and the sort operation.

Expected Results:

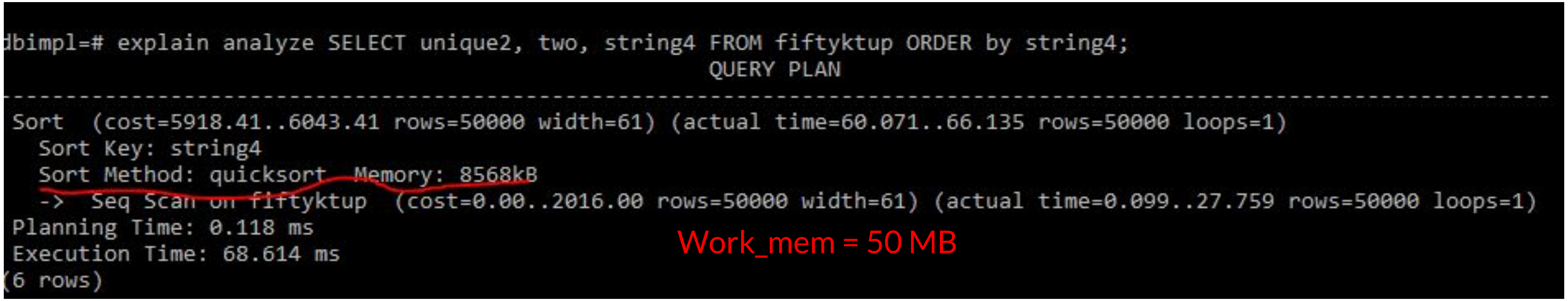
As the sorting in the work memory. We are expecting improved performance with increased work memory when dealing with larger relations that don't fit in work memory.

Execution - 1



```
dbimpl=# explain analyze SELECT unique2, two, string4 FROM fiftyktup ORDER by string4;
                                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Sort  (cost=7801.41..7926.41 rows=50000 width=61) (actual time=101.443..121.175 rows=50000 loops=1)
   Sort Key: string4
   Sort Method: external merge  Disk: 3480kB
   ->  Seq Scan on fiftyktup  (cost=0.00..2016.00 rows=50000 width=61) (actual time=0.088..30.890 rows=50000 loops=1)
 Planning Time: 0.160 ms
 Execution Time: 124.638 ms
(6 rows)
```

Work_mem = 4 MB



```
dbimpl=# explain analyze SELECT unique2, two, string4 FROM fiftyktup ORDER by string4;
                                                   QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Sort  (cost=5918.41..6043.41 rows=50000 width=61) (actual time=60.071..66.135 rows=50000 loops=1)
   Sort Key: string4
   Sort Method: quicksort  Memory: 8568kB
   ->  Seq Scan on fiftyktup  (cost=0.00..2016.00 rows=50000 width=61) (actual time=0.099..27.759 rows=50000 loops=1)
 Planning Time: 0.118 ms
 Execution Time: 68.614 ms
(6 rows)
```

Work_mem = 50 MB

Results and Inferences - 1

Exp No	Execution time in ms for wor_mem	
	4MB	50MB
1	127.5	68.4
2	149.9	74.4
3	129.6	78.5
4	124.1	76.1
5	120.8	81.9
Avg	130.38	75.86

The results of the experiment are as expected.

The size of the involved relation (fiftyktup) is 24 MB. DB uses work_mem for sorting the relationship.

In the initial experiments since the size of the relationship is larger than work_mem, the db had to use *external merge* to sort the relation.

After changing the work_mem to 50 MB, the relation fits in the work_mem. The DB now uses *quick sort* to sort the table. Thus bringing down the average execution time from 130 ms to 76 ms.

Experiment - 2

Query:

explain analyze select f1.twenty from fiftyktup1 f1 where f1.twenty in (select f2.twenty from onelakhktup2 f2 where f2.unique1 between 250 and 750000 and f1.two= f2.two);

Parameter Tested

The parameter that we toggled here was the enable_seqscan (ON/OFF). The default is ON.

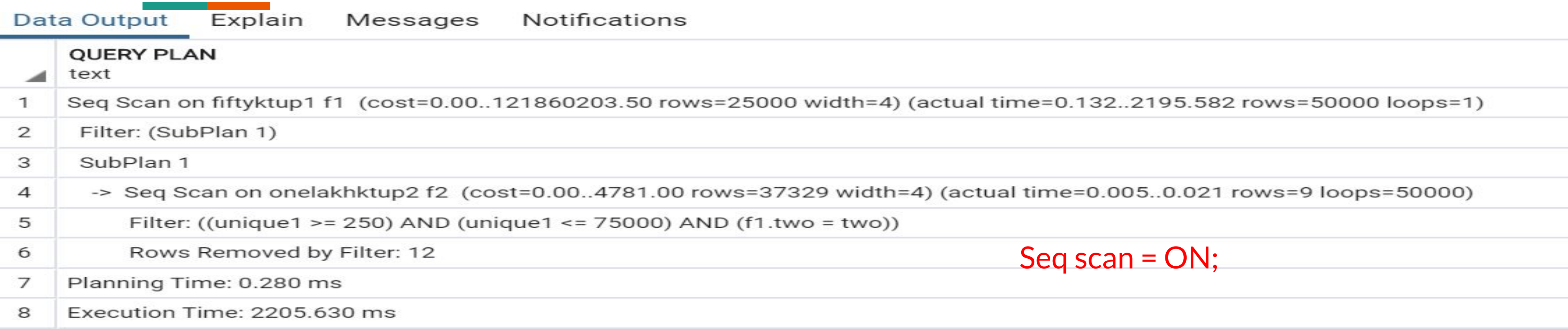
Performance Issue tested

This test explores the issue of how the parameter of sequential scan has an effect on the query execution runtime. Using a non indexed query for evaluation.

Expected Results:

According to us, if we run the query using the sequential scan, it may execute in less amount of time. But we think if we toggle it off, the query can take longer time than it does with sequential scan.

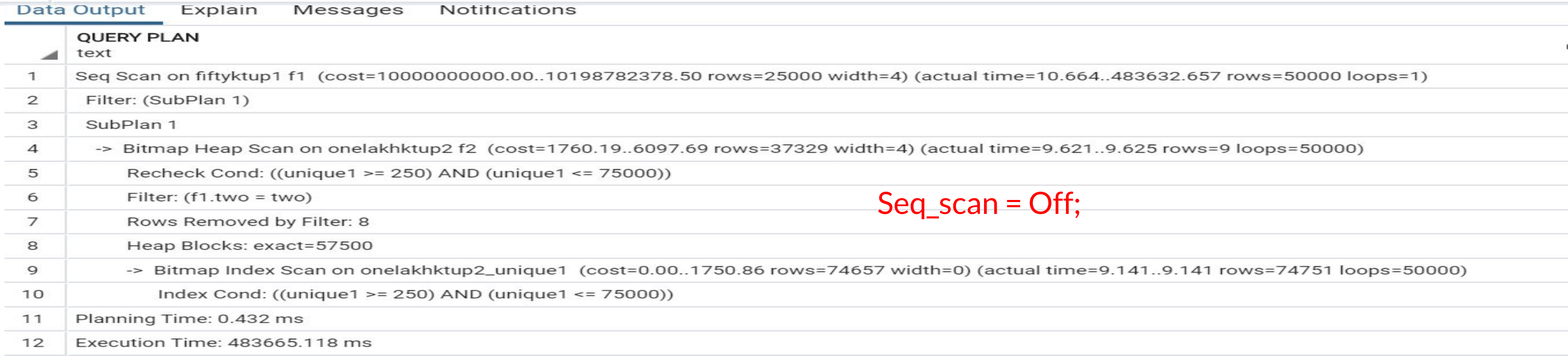
Execution - 2



Data Output   Explain   Messages   Notifications

```
   QUERY PLAN
   text
1  Seq Scan on fiftyktup1 f1  (cost=0.00..121860203.50 rows=25000 width=4) (actual time=0.132..2195.582 rows=50000 loops=1)
2    Filter: (SubPlan 1)
3    SubPlan 1
4      ->  Seq Scan on onelakhktup2 f2  (cost=0.00..4781.00 rows=37329 width=4) (actual time=0.005..0.021 rows=9 loops=50000)
5            Filter: ((unique1 >= 250) AND (unique1 <= 75000) AND (f1.two = two))
6            Rows Removed by Filter: 12
7  Planning Time: 0.280 ms
8  Execution Time: 2205.630 ms
```

Seq scan = ON;



Data Output   Explain   Messages   Notifications

```
    QUERY PLAN
    text
1   Seq Scan on fiftyktup1 f1  (cost=10000000000.00..10198782378.50 rows=25000 width=4) (actual time=10.664..483632.657 rows=50000 loops=1)
2     Filter: (SubPlan 1)
3     SubPlan 1
4       ->  Bitmap Heap Scan on onelakhktup2 f2  (cost=1760.19..6097.69 rows=37329 width=4) (actual time=9.621..9.625 rows=9 loops=50000)
5             Recheck Cond: ((unique1 >= 250) AND (unique1 <= 75000))
6             Filter: (f1.two = two)
7             Rows Removed by Filter: 8
8             Heap Blocks: exact=57500
9             ->  Bitmap Index Scan on onelakhktup2_unique1  (cost=0.00..1750.86 rows=74657 width=0) (actual time=9.141..9.141 rows=74751 loops=50000)
10                  Index Cond: ((unique1 >= 250) AND (unique1 <= 75000))
11  Planning Time: 0.432 ms
12  Execution Time: 483665.118 ms
```

Seq_scan = Off;

Results and Inferences - 2

Exp No	Execution time in ms for seq_scan	
	ON	OFF
1	2137.687	499044.045
2	2290.940	505469.346
3	2205.630	483665.118
4	1939.470	512001.137
5	2068.254	524982.145
Avg	1687.27	505032.358

The results of the experiment are as expected.

In the initial experiments if we run the query using sequential scan ON it runs in less amount of time. If we turn off the seqscan the query took longer time.

When it is off, Query optimiser prioritised the bitmap index scan which is less efficient than sequential scan when large part of relation read into memory. Thus the execution time increased.

It is interesting to notice that it is impossible to suppress sequential scans entirely, but turning this variable off discourages the planner from using one.

Experiment - 3

Query:

explain analyze select tenktup1.string4 from onelakhhktup1, tenktup1 where onelakhhktup1.unique2 = tenktup1.unique2 and onelakhhktup1.unique2 < 2000;

Parameter Tested

The parameter that we toggled here was the enable_hashjoin (ON/OFF). The default is ON.

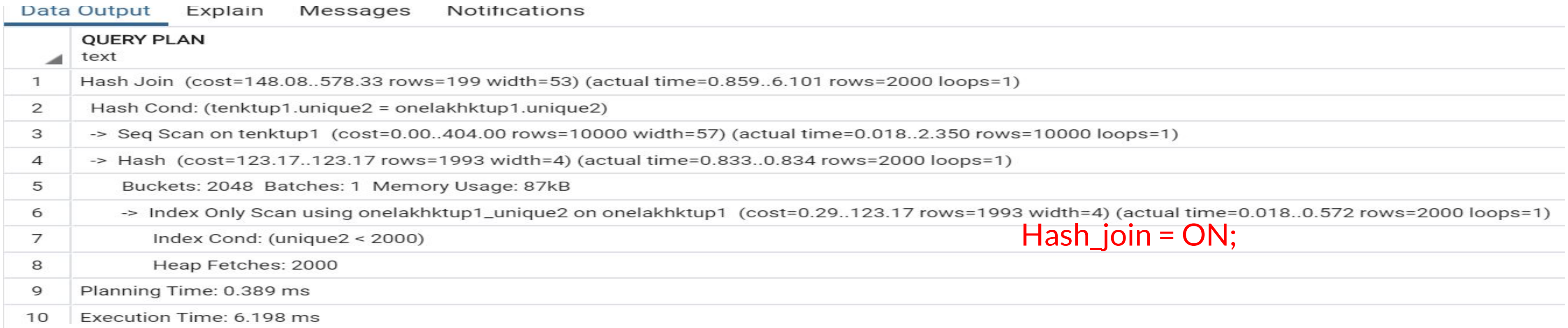
Performance Issue tested

This test explores the issue of how the execution time varies according to the join algorithms.

Expected Results:

As the number of tuples in one table is less which will fit in the memory, it will opt for hash join, which might provide the best execution time. Now, if we turn off hash join, the execution time increases. It uses merge join.

Execution - 3

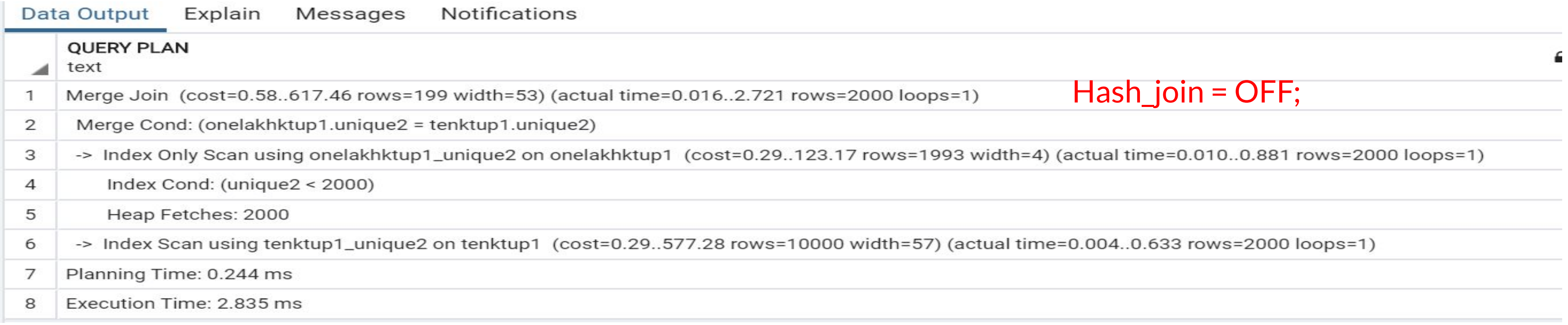


Data Output | Explain | Messages | Notifications

QUERY PLAN
text

```
Hash Join  (cost=148.08..578.33 rows=199 width=53) (actual time=0.859..6.101 rows=2000 loops=1)
  Hash Cond: (tenktup1.unique2 = onelakhktup1.unique2)
  ->  Seq Scan on tenktup1  (cost=0.00..404.00 rows=10000 width=57) (actual time=0.018..2.350 rows=10000 loops=1)
  ->  Hash  (cost=123.17..123.17 rows=1993 width=4) (actual time=0.833..0.834 rows=2000 loops=1)
        Buckets: 2048  Batches: 1  Memory Usage: 87kB
        ->  Index Only Scan using onelakhktup1_unique2 on onelakhktup1  (cost=0.29..123.17 rows=1993 width=4) (actual time=0.018..0.572 rows=2000 loops=1)
              Index Cond: (unique2 < 2000)
              Heap Fetches: 2000
Planning Time: 0.389 ms
Execution Time: 6.198 ms
```

Hash_join = ON;



Data Output | Explain | Messages | Notifications

QUERY PLAN
text

```
Merge Join  (cost=0.58..617.46 rows=199 width=53) (actual time=0.016..2.721 rows=2000 loops=1)
  Merge Cond: (onelakhktup1.unique2 = tenktup1.unique2)
  ->  Index Only Scan using onelakhktup1_unique2 on onelakhktup1  (cost=0.29..123.17 rows=1993 width=4) (actual time=0.010..0.881 rows=2000 loops=1)
        Index Cond: (unique2 < 2000)
        Heap Fetches: 2000
  ->  Index Scan using tenktup1_unique2 on tenktup1  (cost=0.29..577.28 rows=10000 width=57) (actual time=0.004..0.633 rows=2000 loops=1)
Planning Time: 0.244 ms
Execution Time: 2.835 ms
```

Hash_join = OFF;

Results and Inferences - 3

Exp No	Execution time in ms for hash_join	
	Before	After
1	7.298	2.248
2	5.695	3.340
3	6.265	4.890
4	6.036	2.918
5	6.198	2.835
Avg	6.298	3.246

The results of the experiment are not as expected.

In the initial experiments the hash_join is ON as the number of tuples in one table is less which will fit in memory, it will opt for hash join.

If we turn off hash join, the execution time decreases as there is a clustered index on unique2 attribute so it takes sort merge join as merging on clustered index takes less time. Thus it brings down the average execution time from 6.298 ms to 3.246ms.

Experiment - 4

Query:

SELECT fiftyktup.unique2, hundredktup.unique2

FROM fiftyktup, hundredktup WHERE fiftyktup.string4 = hundredktup.string4;;

Parameter Tested
Shared_Buffer varied from 128 MB to 10 MB

Relations size is kept constant 24 MB and 48 MB. Work_mem is constant

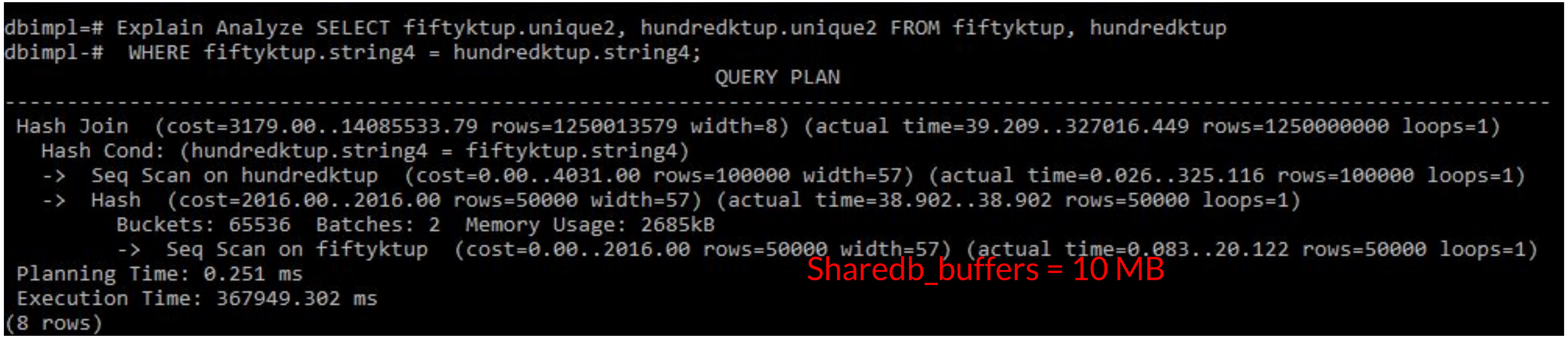
Performance Issue tested:

This test explores the relation betweens shared_buffers and the execution time of large queries whose relations do not fit in memory

Expected Results:

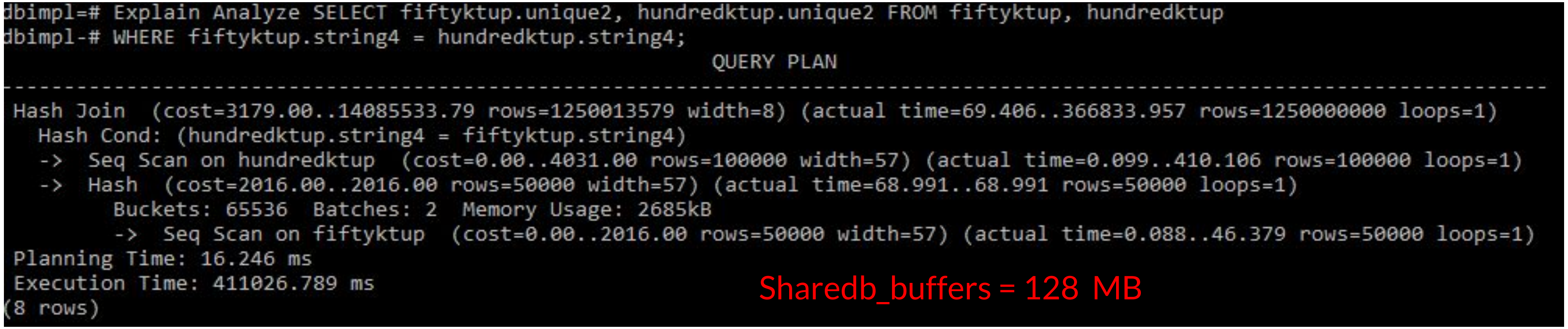
Theoretically. The join algorithms should give better execution times with increased buffer sizes.

Execution - 4



```
dbimpl=# Explain Analyze SELECT fiftyktup.unique2, hundredktup.unique2 FROM fiftyktup, hundredktup
dbimpl-#  WHERE fiftyktup.string4 = hundredktup.string4;
                                                         QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=3179.00..14085533.79 rows=1250013579 width=8) (actual time=39.209..327016.449 rows=1250000000 loops=1)
   Hash Cond: (hundredktup.string4 = fiftyktup.string4)
   ->  Seq Scan on hundredktup  (cost=0.00..4031.00 rows=100000 width=57) (actual time=0.026..325.116 rows=100000 loops=1)
   ->  Hash  (cost=2016.00..2016.00 rows=50000 width=57) (actual time=38.902..38.902 rows=50000 loops=1)
         Buckets: 65536  Batches: 2  Memory Usage: 2685kB
         ->  Seq Scan on fiftyktup  (cost=0.00..2016.00 rows=50000 width=57) (actual time=0.083..20.122 rows=50000 loops=1)
 Planning Time: 0.251 ms
 Execution Time: 367949.302 ms
(8 rows)
```

Sharedb_buffers = 10 MB



```
dbimpl=# Explain Analyze SELECT fiftyktup.unique2, hundredktup.unique2 FROM fiftyktup, hundredktup
dbimpl-# WHERE fiftyktup.string4 = hundredktup.string4;
                                                         QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=3179.00..14085533.79 rows=1250013579 width=8) (actual time=69.406..366833.957 rows=1250000000 loops=1)
   Hash Cond: (hundredktup.string4 = fiftyktup.string4)
   ->  Seq Scan on hundredktup  (cost=0.00..4031.00 rows=100000 width=57) (actual time=0.099..410.106 rows=100000 loops=1)
   ->  Hash  (cost=2016.00..2016.00 rows=50000 width=57) (actual time=68.991..68.991 rows=50000 loops=1)
         Buckets: 65536  Batches: 2  Memory Usage: 2685kB
         ->  Seq Scan on fiftyktup  (cost=0.00..2016.00 rows=50000 width=57) (actual time=0.088..46.379 rows=50000 loops=1)
 Planning Time: 16.246 ms
 Execution Time: 411026.789 ms
(8 rows)
```

Sharedb_buffers = 128 MB

Results and Inferences - 4

Exp No	Execution time in ms for shared_buffers	
	10 MB	128 MB
1	309712	479473
2	351670	477868
3	338567	495160
4	384105	495896
5	371832	411026
Avg	351177	471884

The results of the experiment **are not as expected.**

Surprisingly the average execution time decreased as the shared_buffers size reduced. The experiment is repeated multiple times but the result wouldn't change.

When there isn't enough shared buffer and if the kernel buffer or OS cache is available, the relation could be cached there. Thus resulting in better execution time.

More research into OS cache tells that "on Windows, large values for shared_buffers aren't as effective, and you may find better results keeping it relatively low and using the OS cache more instead"

Summary

- Increasing work_mem increases the sort operation speeds by avoiding read and writes to Disc
- Changing Shared_buffers on windows system doesn’t have clear effect on the execution time of queries involving large joins.
- Disabling the hashjoin decreases the execution time as there is a clustered index on the join predicate. Merge join is efficient for a clustered index query as the relations are already sorted.
- Disabling seqscan increases the execution time when a large part of relations has to be read in to the memory.

Challenges

- Keeping the **database environment Isolated** was difficult. Load from different applications running on the system slightly affects the execution time.
- **Changing shared_buffers** was difficult. It took a while find the postgresql.conf file. The postgress system services needed a restart to bring in the change. The experiment required multiple restarts of postgres services.
- For better results, we tried **flushing out relations from shared_buffers** for every experiment. We were partially successful. This needs deeper understanding of how shared_buffers work.
- Explain and Analyse for operations like ‘Alter table’ is not available.

Lessons learnt

- Few **relations could be cached in kernel buffers or OS cache**. This interference between these two buffers is an interesting topic and needs further research.
- Shared_buffers guarantees primary memory space by reserving it for database. However, **OS cache or kernel buffers is relatively faster** than the database buffers (shared_buffers).
- **The default work_mem(4MB) is too low on postgres**. It could be increased if the database in general performs more sorting or hashing operations. Care should be taken while increasing it, since a work_mem of 4 MB is allocated for every sort operation.
- Sequential cannot be completely suppressed by setting it off.

References

- https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server>
- https://www.postgresql.org/docs/9.1/runtime-config-resource.html
- <http://fibrevillage.com/database/134-useful-sqls-to-check-contents-of-postgresql-shared-buffer>

Thank you.