Database Management System Implementation Project - Part 2

Team Members

- Shweta Korulkar
- Divya Sravani Vukkusila

System Selection:

For this project we decided to evaluate a single system (e.g. postgres) with different parameter values and optimizer options (option 2) . Due to our familiarity with PostGres local installation, interactions and diagnosis, we chose this option. With local Instance, we would have all the flexibility over tuning or controlling all the aspects of the DB compared to a cloud hosted or a remotely hosted PostGres. And Postgres provides better language support for Python and flexibility for user defined data types for any complex computation to be foreseen.

System Research:

For this part we have selected 6 parameters:

1. enable_hashagg(boolean)
2. enable_indexscan(boolean)
3. enable_hashjoin(boolean)
4. enable_nestloop(boolean)
5. enable_seqscan(boolean)
6. enable_mergejoin(boolean)
7. work_mem(integer)

❖ From runtime config query,

1. **enable_hashagg(boolean)** : Enables or disables the query planner's use of hashed aggregation plan types. The default is on.

2. **enable_indexscan（boolean）:** Enables or disables the query planner's use of index-scan plan types. The default is on

3. **enable_seqscan (boolean):** Enables or disables the query planner's use of sequential scan plan types. 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. The default is on.

4. **enable_mergejoin （boolean） :** Enables or disables the query planner's use of merge-join plan types. The default is on.

5. **enable_hashjoin (boolean)**: Enables or disables the query planner's use of hash-join plan types. The default is on.

6. **enable_nestloop (boolean) :** Enables or disables the query planner's use of nested-loop join plans. It is impossible to suppress nested-loop joins entirely, but turning this variable off discourages the planner from using one if there are other methods available. The default is `on`.

❖ From runtime_config_resource,

7. **work_mem (integer)**: Specifies the amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files.

Performance Experiment Design

The performance experiments we are going to run are:

1. Test the performance with no index and with indexes with 10% rule of thumb :

a. What performance issue you testing?

1) In this, we are going to explore when it is good to use an unclustered index vs. not using an index vs. using a clustered index
2) Execution time before and after using the parameter value. I.e execution time require for normal sql query without optimization and execution time of sql query with optimization using parameter.

b. What data sets will you include in the test? Will you use a single size data set or will you use multiple data sets of different sizes?

1) We are using multiple size data sets with 10,000, 100,000, 100,000 tuples in each Table.
2) We are going to use queries other than wisconsin benchmark. Will run query on same dataset. Few queries will be using clustered index and few with non-clustered index and few with no index.

c. What queries will you run?

We will run selectivity query which will contain range selection 1% or 10% on clustered index, non-clustered and no index

1) SET enable_indexscan = off; // with no index
EXPLAIN ANALYZE SELECT * FROM tenktup1
WHERE unique2 > 50000
AND unique2 < 60000
AND stringu1 LIKE 'AAAADA%';

2) SET enable_indexscan = on; // with clustered index
EXPLAIN ANALYZE SELECT * FROM tenktup1
WHERE unique2 > 50000
AND unique2 < 60000
AND stringu1 LIKE 'AAAADA%';

3) SET enable_indexscan = on; // with unclustered index

```
SET enable_seqscan = off;
 EXPLAIN ANALYZE SELECT * FROM tenktup1
 WHERE unique1 > 50000
 AND unique1 < 60000
 AND stringu1 LIKE 'AAAADA%';
```

4) SET enable_indexscan = on;
```
// with no index though index exists since more than 10% selectivity
EXPLAIN ANALYZE SELECT * FROM tenktup1
WHERE unique2 > 10
```

d. If you are doing Option 2, what parameters will you use for this test? How will the parameters be set/varied?

We will use following parameter:
enable_indexscan(boolean),enable_seqscan(boolean)
We can set these parameters as follows,
SET enable_indexscan = on;
SET enable_seqscan = off;

e. What results do you expect to get?

The expected result would be that the execution time for queries with no index takes more execution time, then followed by queries with unclustered index, then followed by queries with clustered index.

2. Test the aggregation algorithm performance:

a. What performance issue you testing?

1) In this, we are going to explore how aggregation performs when enabling and disabling parameters like enable_hashagg(boolean) and work_mem(integer).
2) Execution time before and after using the parameter value. I.e execution time require for normal sql query without optimization and execution time of sql query with optimization using parameter.

b. What data sets will you include in the test? Will you use a single size data set or will you use multiple data sets of different sizes?

1) We are using multiple size data sets with 10,000, 100,000, 100,000 tuples in each Table.
2) We are going to use queries other than wisconsin benchmark. Will run query on same dataset. Few queries will be run enabling parameters and some with parameters disabled.

c. What queries will you run?

We will run selectivity query which will contain range selection 1% or 10% on clustered index, non-clustered and no index

1) Count aggregate with 10 partitions

```
SET  work_mem = '20MB'; // with clustered index and explicit work_mem setting
SET enable_hashagg = off; // disable hashing aggregation
SET enable_indexscan = on;
EXPLAIN ANALYZE SELECT COUNT(unique2) FROM tenktup1
WHERE unique2 > 90000
GROUP BY tenPercent
HAVING tenPercent > 5;
```

2) Count aggregate with 10 partitions

```
SET  work_mem = '20MB'; // with clustered index and explicit work_mem setting
SET enable_hashagg = on; // enable hashing aggregation
SET enable_indexscan = on;
EXPLAIN ANALYZE SELECT COUNT(unique2) FROM tenktup1
WHERE unique2 < 20000
GROUP BY tenPercent
HAVING tenPercent > 5;
```

3) Max aggregate with 10 partitions

```
SET  work_mem = '20MB'; // with nonclustered index and explicit work_mem setting
SET enable_hashagg = on; // enable hashing aggregation
SET enable_indexscan = on; // enable indexscan
EXPLAIN ANALYZE SELECT MAX(unique1) FROM tenktup1
WHERE unique1 > 90000
GROUP BY tenPercent
HAVING tenPercent > 5;
```

d. If you are doing Option 2, what parameters will you use for this test? How will the parameters be set/varied?

We will use following parameter: enable_indexscan(boolean), enable_hashagg(boolean), work_mem(integer)

We can set these parameters as follows,

```
SET work_mem = '20MB';
SET enable_hashagg = on;
SET enable_indexscan = on;
```

e. What results do you expect to get?

The expected result would be that the execution time for queries with hashing aggregation and with indexing takes less execution time, than queries with no indexing and with no hashing aggregation.

3. Test the different join algorithms performance:

a. What performance issue you testing?

1) In this, we are going to explore how join performs when enabling and disabling parameters like enable_hashjoin(boolean), enable_mergejoin(boolean), enable_nestloop(boolean)

2) time before and after using the parameter value. I.e execution time require for normal sql query without optimization and execution time of sql query with optimization using parameter.

b. What data sets will you include in the test? Will you use a single size data set or will you use multiple data sets of different sizes?

1) We are using multiple size data sets with 10,000, 100,000, 100,000 tuples in each Table.
2) We are going to use queries other than wisconsin benchmark. Will run query on same dataset. Few queries will be run enabling parameters and some with parameters disabled.

c. What queries will you run?

We will run selectivity query which will contain range selection 1% or 10% on clustered index, non-clustered and no index

1)
```
SET enable_nestloop = on; // with clustered index and with nested loop join, by
                                        // default on
EXPLAIN ANALYZE SELECT * FROM TENKTUP1, TENKTUP2, ONEKTUP
WHERE (TENKTUP1.unique2 = TENKTUP2.unique2)
AND (ONEKTUP.unique2 = TENKTUP1.unique2)
AND (ONEKTUP.unique1 < 9000)
AND (ONEKTUP.unique1 > 5000)
AND (TENKTUP1.unique2 < 10000)
AND (TENKTUP1.unique1 > 10000)
AND (TENKTUP1.unique1 < 40000)
```

2)
```
SET enable_hashjoin = on; // with unclustered index and with hashjoin by default on
SET enable_nestloop = off;
EXPLAIN ANALYZE SELECT * FROM TENKTUP1, TENKTUP2, ONEKTUP
WHERE (TENKTUP1.unique2 = TENKTUP2.unique2)
AND (ONEKTUP.unique2 = TENKTUP1.unique2)
AND (ONEKTUP.unique1 < 9000)
AND (ONEKTUP.unique1 > 5000)
AND (TENKTUP1.unique2 < 10000)
AND (TENKTUP1.unique1 > 10000)
AND (TENKTUP1.unique1 < 40000)
```

3)
```
SET enable_hashjoin = off; // with clustered index and with nested loop join
SET enable_mergejoin = on;
SET enable_nestloop = off;
EXPLAIN ANALYZE SELECT * FROM TENKTUP1, TENKTUP2, ONEKTUP
WHERE (TENKTUP1.unique2 = TENKTUP2.unique2)
AND (ONEKTUP.unique2 = TENKTUP1.unique2)
AND (ONEKTUP.unique1 < 9000)
AND (ONEKTUP.unique1 > 5000)
AND (TENKTUP1.unique2 < 10000)
AND (TENKTUP1.unique1 > 10000)
AND (TENKTUP1.unique1 < 40000)
```

d. If you are doing Option 2, what parameters will you use for this test? How will the parameters be set/varied?

We will use following parameter: enable_hashjoin(boolean), enable_mergejoin(boolean), enable_nestloop(integer)

We can set these parameters as follows,

```
SET enable_hashjoin = on;
SET enable_mergejoin = on;
SET enable_nestloop = on;
```

e. What results do you expect to get?

The expected result would be that the execution time for queries with hash join and with merge join takes less execution time, than queries with nested loop join.

4. Test on work_mem parameter:

a. What performance issue you testing?

1) In this, we are going to explore how changing the default amount of memory through work_mem impacts the sorting of the result sets.
2) Time before and after using the parameter value i.e, execution time required for normal sql query without optimization and execution time of sql query with optimization using parameter.

b. What data sets will you include in the test? Will you use a single size data set or will you use multiple data sets of different sizes?

1) We are using multiple size data sets with 10,000, 100,000, 100,000 tuples in each Table.
2) We are going to use queries other than wisconsin benchmark. Will run query on same dataset. Few queries will be run enabling parameters and some with parameters disabled.

c. What queries will you run?

1)
```
RESET work_mem; // with default memory of 4MB
EXPLAIN ANALYZE SELECT DISTINCT(t1.unique3)
FROM TENKTUP1 t1 INNER JOIN TENKTUP2 t2
ON t1.unique1 = t2.unique1
ORDER BY t1.unique3;
```

2)
```
SET work_mem = '32MB';  // with more amount of memory assigned
EXPLAIN ANALYZE SELECT DISTINCT(t1.unique3)
FROM TENKTUP1 t1 INNER JOIN TENKTUP2 t2
```

```
ON t1.unique1 = t2.unique1
ORDER BY t1.unique3;
```

3) RESET work_mem; // with default memory of 4MB

```
EXPLAIN ANALYZE SELECT DISTINCT(unique3)
FROM TENKTUP1 t1
WHERE unique3 < (SELECT unique3
                 FROM TENKTUP2 t2
                 WHERE t1.unique1 = t2.unique1
                 AND t2.unique2 < (SELECT unique2
                                   FROM TENKTUP1 t3
                                   WHERE t3.unique1 = t2.unique1))
AND (MOD(onepercent, 2) = 0)
ORDER BY unique3;
```

4) SET work_mem = '64MB'; // with more amount of memory assigned

```
EXPLAIN ANALYZE SELECT DISTINCT(unique3)
FROM TENKTUP1 t1
WHERE unique3 < (SELECT unique3
                 FROM TENKTUP2 t2
                 WHERE t1.unique1 = t2.unique1
                 AND t2.unique2 < (SELECT unique2
                                   FROM TENKTUP1 t3
                                   WHERE t3.unique1 = t2.unique1))
AND (MOD(onepercent, 2) = 0)
ORDER BY unique3;
```

d. If you are doing Option 2, what parameters will you use for this test? How will the parameters be set/varied?

We will use following parameter: work_mem(integer)

We can set these parameters as follows,

```
SET work_mem= '64MB';
```

e. What results do you expect to get?

The expected result would be that the execution time for sorting queries with more memory assigned is more than those sorting queries with default amount of memory assigned.

Lessons Learned: Include lessons learned or issues encountered

1. The effectiveness of using index on queries and how they will impact and improve the performance by reducing the execution times.
2. Gained better knowledge about various parameters that affects query planning and resource consumption and realized that using required parameters for respective queries would greatly improve performance.
3. After experimenting through various queries with different join algorithms, we realized Hash join and merge join performed better than nested loop join.

4. Aggregation queries through hashing would perform better.
5. Increased understanding of database system performances and the impacts of database implementation techniques on performance.
6. Initially we had issues in understanding few resource consumption parameters and their usage.
7. It took time to understand how to frame queries to test performance through resource consumption parameters.

References:

1. The Wisconsin Benchmark: Past, Present, and Future - David J. DeWitt, Computer Sciences Department, University of Wisconsin

2. Portland State University CS 587 course material

3. www.python.org

4. www.postgresql.org

5. www.postgresqltutorial.com

6. www.postgresql.org/docs/9.4/runtime-config-resource.html

7. www.postgresql.org/docs/9.4/runtime-config-query.html

8. wiki.postgresql.org/wiki/Python