CS 487/587 DB Implementation
Project Pt. 2
Minjin Enkhjargal
Yan Li

SYSTEM RESEARCH

Memory:

1. work_mem (integer):
    a. The default value of work_mem is 4MB. This is the amount of memory that can be used for hash tables and internal sort operations (such as when using ORDER BY and merge joins).
2. huge_pages (integer):
    a. This parameter is available for Linux only at the moment. It enables or disables the use of huge memory pages. Using huge pages may increase performance by spending less CPU time on memory management by using smaller page tables There are three choices.
        i. huge_pages set to *try*: server will attempt to use huge pages
        ii. huge_pages set to *on*: server will use huge pages, but if it huge pages cannot be used, the server will not start up
        iii. huge_pages set to *off*: server will not use huge pages.
3. temp_buffers (integer):
    a. This parameter determines the maximum number of temporary buffers that can be used during a database session.

Query Planner:

1. enable_indexscan (boolean):
    a. Defaulted to on - This enables or disables the use of index-scan plan types by the query planner. If disabled, index-scan will not be used during queries. During some queries, this may not make a huge difference, however for queries on tables that are using indexes, index-scan can hugely optimize the query performance.
2. enable_mergejoin (boolean):
    a. Defaulted to on - This enables or disables the use of merge-join plan types by the query planner. If disabled, the query planner will not use merge-joins
3. enable_hashagg (boolean):
    a. Defaulted to on - This enables or disables use of hashed aggregation plan types by the query planner. If disabled, it will not use hashed aggregation plan types.

EXPERIMENT DESIGN

Experiment 1

1. Sizeup
    a. Experiment Specifications
        i. **Purpose:** Explores how the system responds to different sizes of relations
        ii. **Data:** Use 1k, 10k, 100k, 1000k relations.
        iii. **Queries Used:** Use query 7 and run the query on the different size relations, compare how query execution time changes as the database size changes.
        iv. **Parameters:** No parameters changed in this test.
        v. **Expected Results:** Query 7 uses a clustered index and is looking for when unique2 = 2001. Since it is looking for one tuple and uses an ordered index, we expect execution to be fairly quick. We expect performance to become slower for larger relations, but not substantially.

Experiment 2

2. Test 2 queries with and without a hash aggregate plan type
    a. Experiment Specifications
        i. **Purpose:** Explores how queries are affected when the query planner either uses or does not the hash aggregate plan type
        ii. **Data:** Use 100K tuple relations.
        iii. **Query Used:** Use Wisconsin Benchmark queries Query 20 (no index) and Query 23 (with clustered index), and Query 21 (no index) and Query 24 (with clustered index). Run each query with the enable_hashagg turned on and off.
        iv. **Parameters:** enable_hashagg (boolean).
        v. **Expected Results:** The expected result is that when the enable_hashagg is turned on, these queries will likely perform faster, since they use an aggregate operator. On the other hand, when the enable_hashagg is turned off, the queries will likely run slower. When this parameter is disabled, it will likely default to a merge sort on disk, which will cause the execution to be slower.

Experiment 3

3. Test the 3 join algorithms used by PostgreSQL.
    a. Experiment specifications
        i. **Purpose:** This experiment will explore how the 3 join algorithms perform for the same query.
        ii. **Data:** Use 10K and 100K tuple relations.
        iii. **Queries Used:** Use Wisconsin Bench query 10 (no index) and 13 (clustered index). We will run the query 3 times. For each round, we disable 2 parameters while enabling 1.
        iv. **Parameters:** *enable_nestloop, enable_hashjoin and enable_mergejoin*. These parameters are BOOL, and can be enabled and disabled.
        v. **Expected Results:** Query 10 and 13 uses a simple join and uses a clustered index. Since this query will use the 100K tuple, BPRIME will contain 10% of the data, which is 10K. Since the two tables 10K and 100K are different and larger in size, the hash-join algorithm should run the fastest. Merge join should come in second place in terms of performance, because it performs well executing a clustered index and larger data sets. The reason it'd be slower than hash join is that the tables vary too much in their sizes. Nested loop joins perform well when one table is significantly small and uses indices, since it requires fewer comparisons. In our experiment, nested loop joins will likely perform the worst, since the data sizes may be too large for this type of join algorithm to perform well. The second table BPRIME contains 10% of the first table (100K), which we do not believe is significantly smaller.

Experiment 4

4. Test clustered vs non clustered of two different sized tuple relations
    a. Experiment Specifications
        i. **Purpose:** Explores how two different sizes of relations would respond differently to clustered vs nonclustered indices.
        ii. **Data:** Use 1K and 100K tuple relations.
        iii. **Query Used:** Use Wisconsin Benchmark queries 7(clustered index), 15(non-clustered index). Run these queries on the 1k and 100k tuple relations. Compare each query on the different sized tuples to see their performances.
        iv. **Parameters:** No parameters changed in this test

v. **Expected Results:** The expected result is that the clustered index tuple relation will be faster than the non-clustered index for the 100K tuple relation. Since this is a larger data set, having a clustered index to determine the order of the data may allow the execution

LESSONS

At first we had a very blurred idea about how the parameters would be incorporated into our queries. We knew there were many useful parameters that could be used to alter the performance, but it took many resources other than the official Postgresql documentation for us to understand. We learned the parameters can be manipulated through a command line or the config file. Also, we learned how parameters can have an impact on the performance of a query. For example, disabling certain query planner parameters such as enable_hashagg, may cause a process to be executed slower or not as efficiently. Another thing that allowed us to understand the system better was learning what the default parameters and default memory sizes were. It made us critically think about why the query planner may use some optimizations over others or why some memory parameters are set to their default values. For example, the default value for work_mem is 4MB which seems to be a decent size since using a larger memory is expensive. Additionally, for a database session where multiple queries are running in parallel, each process would use that size of memory for work_mem and it can become very memory intensive. Therefore, when designing an experiment, it's important to take into consideration how the parameter can fit and influence the data.

References Cited

1. https://www.postgresql.org/docs/9.4/runtime-config-resource.html
2. https://www.postgresql.org/docs/9.4/runtime-config-query.html
3. PostgreSQL 10 High Performance: Expert techniques for query optimization ... - Enrico Pirozzi - Google Books
4. Loop Join in SQL Server 2008 - Stack Overflow
5. https://www.linkedin.com/pulse/loop-hash-merge-join-types-eitan-blumin