CS487-Project-part2

By Jianqiang Hao and Zhi Zhang

System selection

We choose to work on mysql and postgresql, because they are both commonly used by the industry, and provide solid documentation for the research purpose, and the PSU server hosts both of these two systems, so that we can compare their performance while isolating the hardware factors.

System research

Types of indices

- PostgreSQL
    - B-tree
        - By default, the CREATE INDEX command creates B-tree indexes, which can handle equality and range queries on data that can be sorted into some ordering, and it can also be used to retrieve data in sorted order. This is not always faster than a simple scan and sort, but it is often helpful.
    - Hash
        - Hash indexes can only handle simple equality comparisons.
    - GIST
        - Gist can comprises many different indexing strategies depending on the operator class, and can be used for several two-dimensional geometric data types or optimizing "nearest-neighbor" searches etc.
    - SP-GIST
        - It is similar to GIST, but permits implementation of a wide range of different non-balanced trees.
- Mysql
    - Just like PostgeSQL, Mysql also uses B-tree as default, and provides hash indexes.
    - Fulltext
        - This is a type of index that can be used for text lookups, they do not work by comparing values, it’s more like a keyword searching.

Types of join algorithm

- PostgreSQL
    - NESTED LOOP JOIN
        - The right relation is scanned once for every row found in the left relation. This strategy is easy to implement but can be very time consuming. (However, if the right relation can be scanned with an index scan, this can be a good strategy. It is possible to use values from the current row of the left relation as keys for the index scan of the right.)
    - MERGE JOIN
        - Each relation is sorted on the join attributes before the join starts. Then the two relations are scanned in parallel, and matching rows are combined to form join rows. This kind of join is more attractive because each relation has to be scanned only once. The required sorting might be achieved either by an explicit sort step, or by scanning the relation in the proper order using an index on the join key.
    - HASH JOIN
        - The right relation is first scanned and loaded into a hash table, using its join attributes as hash keys. Next the left relation is scanned and the appropriate values of every row found are used as hash keys to locate the matching rows in the table.
- Mysql
    - NESTED LOOP JOIN
        - Just like what PostgreSQL would do.
    - BLOCK NESTED LOOP JOIN
        - Using buffering of rows read in outer loops to reduce the number of times that tables in inner loops must be read.

Buffer pool size/structure

- Mysql
    - MySQL uses innoDB as it's default storage engine, in which the buffer pool is divided into pages that can potentially hold multiple rows, the size of each page is 16KB by default which is same as page size on disk, and the buffer pool is implemented as a linked list of pages. Page replacement algorithm is LRU(Least Recently Used).
- PostgreSQL
    - The buffer pool layer stores data file pages, such as tables and indexes. The buffer pool is an array, each slot stores one page of a data file. Indices of a buffer pool array are referred to as buffer_ids.The buffer pool slot size is 8 KB, which is equal to the size of a page. Thus, each slot can store an entire page, and the buffer pool size can be configured. Page

replacement algorithm is clock sweep. This algorithm is a variant of NFU (Not Frequently Used) with low overhead; it selects less frequently used pages efficiently.When reading or writing a huge table, PostgreSQL uses a ring buffer rather than the buffer pool. The ring buffer is a small and temporary buffer area.

Time measuring mechanism

We use the phpPgadmin to connect to the postgreSQL, and use the phpMyadmin to connect to the MySQL, both systems are hosted on the server of PSU, the query execution time would be measured and displayed by the php tools automatically.

Performance Experiment Design

1. Compare equality-based query and range-based query performance across systems.
    a. Experiment specifications:
        i. This experiment compares equality-based query and range-based query performance on two selected systems.
        ii. Using a 10,000 tuple relation (TENKTUP1).
        iii. select * From TENKTUP1 where unique1 between 0 and 99;/select * From TENKTUP1 where unique1 = 105;
        iv. No parameters changed in this test.
        v. We expect they will have similar query execution time, because they probably would use the similar index(B-tree and hash) to plan the query.

2. Compare join algorithm performance across systems.
    a. Experiment Specifications
        i. This experiment compares join algorithm performance on two selected systems.
        ii. Using a 10,000 tuple relation (TENKTUP1).
        iii. Use Wisconsin Bench queries 10 and 11. Run queries 10 and 11 on your two selected systems and compare the performance. Also, look at the query execution plans and report what types of joins the systems use. Note queries 10 and 11 assume/require that there are no indices on the relations.
        iv. No parameters changed in this test.
        v. We expect the PostgreSQL would have a better performance, because it probably would use hash join to execute the query which MySQL won't, which could give us insights on how much the hash join would impact the performance.

3. Compare aggregation performance and scalability across systems.
    a. Experiment specifications:
        i. This experiment tests the performance for each system using sum(), count() in different size files.
        ii. Using a 100,000 tuple, a 10,000 tuple relation (TENKTUP1), and a 1000 tuple relation.
        iii. Select count(*) from relation Group by four; / Select sum(two) from relation Group by four;
        iv. No parameters changed in this test.
        v. We expect time to count less than the time of sum function, because sum function possibly needs to use more memory to save data. For using different sizes of files, we expect the query execution time won't grow linearly because the query planner would switch scan strategy(from sequence scan to index scan) when the data size is greater than a certain amount.

4. Compare performance bulk updates with different selectivities across systems.
    a. Experiment specifications:
        i. Test the bulk update performance of two selected systems, using queries with 25% selection, 50% selection, 100% selection.
        ii. Using a 10,000 tuple relation (TENKTUP1).
        iii. Perform the following queries:

25% selection:

```
Update TENKTUP1
Set two = 0
Where four = 1;
```

50% selection:

```
Update TENKTUP1
Set four = 0
Where two = 1;
```

100% selection:

```
Update TENKTUP1
Set four = 0
Where two >= 0;
```

        iv. No parameters changed in this test.
        v. We expect the query execution time of both systems won't be similar, because they have totally different buffer structure.

Lessons learned

When we want to complete highly efficient software, we need to use the corresponding efficient tools to serve us. The purpose of our experiment is to analyze the two most popular database systems and find the most suitable application environment for them. In this process, it is difficult to find their direct differences only through the execution of simple SQL statements. Therefore, we designed experiments to isolate other influence conditions as variables to make it easier to identify performance differences. Learning the principles of database implementation is very useful to help us explore different database systems and quickly understand the structure of different databases.

References

https://www.postgresql.org/docs/9.2/indexes-types.html
https://www.postgresql.org/docs/9.0/planner-optimizer.html
https://dev.mysql.com/doc/refman/8.0/en/innodb-buffer-pool.html