University of Tartu

Advanced Databases

Spring 2024

Project #2: TPC-C benchmarking

What is benchmarking?

Benchmarking is an approach to assessing a relevant feature of interest of a computer program. In the context of database management systems, benchmarks can be seen as quantitative empirical means to compare the performance of two or more DBMSs.

TPC-C Benchmark

Transaction Processing Performance Council (TPC) has a stack of benchmarks that simulate different processing scenarios for data management systems and applications. TPC-C is one such benchmark that is concerned with transactions processing. TPC-C is an on-line transaction processing (OLTP) benchmark. TPC-C involves a mix of five concurrent transactions of different types and complexity. The database is comprised of nine types of tables with a wide range of records and population sizes. TPC-C is measured in transactions per minute (tpmC). The benchmark portrays the activity of a wholesale supplier. The benchmark simulates common OLTP transactions on a warehouse which include: placing a new order, making a payment, an order-status transaction, a delivery transaction, and a stock level transaction.

The benchmark runs several instances of these transactions and the implementation should compute the transactions per minute metric.

Working with MySQL transaction storage engines

MySQL supports variousstorage engines (InnoDB, MyISAM, Memory, CSV, Archive, Blackhole, NDB, Merge, Federated, and Example). The default storage engine in MySQL is InnoDB. To see which ones are available and supportedby your server, use this command:

```
mysql> SHOW ENGINES\G
```

PostgreSQL on the other hand supports only one storage engine, so TPC-C Benchmarking will not make sense with PostgreSQL. However, as a Bonus with additional marks, a task for the benchmark to work for the Postgres database is also added in this project.

An Example Implementation

In this GitHub repo, https://github.com/AgilData/tpcc, you will find one implementation of TPCC in Java that is designed for the MySQL database. It implements a Java class for each transaction type. In the configuration file, you specify parameters for the benchmark.

Notes to consider about the above implementation:

1. Use the default "JDBC" mode, this will directly load the generated data to your created "tpcc" DB.
2. Make sure to change the password in the properties file (tpcc.properties) to the MySQL-engine configured password

Tasks Required

1. The primary task is to run the benchmark with the default configuration MySQL with the **InnoDB** storage engine and store the performance results. **[10 points]**

2. Next, you need to change the load generated against the database. This can be done by changing the following variables in the tpcc.properties file:
    - CONNECTIONS=30 -- the number of concurrent threads that will issue SQL statements to the database,
    - RAMPUPTIME=45 -- the time taken to spawn all the threads.
    - DURATION=60 -- the time left for all threads to issue commands to the database before they are stopped.
      By tuning these parameters, you can control the total number of transactions that are issued against the database. For example, if you increase the number of connections and shorten the ramp-up time, you will notice that TpmC will start to increase until a certain point where the database is saturated and the number will not increase more.

   [20 points]

3. You can tweak some of the transactions configuration parameters, not yet discussed in the lab session(take a look on file upload on moodle on CC in MySQL) and rerun the benchmark to see the effect on the results. For example, you can change the values of the following parameters:
    - innodb_flush_log_at_trx_commit
    - innodb_deadlock_detect
    - innodb_lock_wait_timeout
      [20 points]

4. You need to repeat steps 1 and 2 above but with the storage engine **MyISAM**. Store the results. To do that, you have to either drop the current database "tpcc" or create a new one, e.g. tpccMyISAM, and change the engine option in the create_tables.sql file and make sure to change the database name there to tpccMyISAM should you choose to create another database. In the tpcc.properties file, you have to change the database name in the connection string, the "JDBCURL", replacing tpcc with tpccMyISAM. Then you have to run the TpccLoad to populate the new database. You can then rerun the benchmark with the different combinations for CONNECTIONS and RAMPUPTIME. You might observe exceptions telling that some thread is trying to violate a primary key constraint. That is OK. The key generator from the benchmark my generate some duplicates and the database should protect the integrity of the data. These

exceptions do not stop the execution of the benchmark. At the end, the benchmark will start the measuring phase and will deliver the measures. You can note the TpmC for MyISAM and compare it to InnoDB setup using the same combination of parameter values, e.g., CONNECTIONS, RAMPUPTIME. **[20 points]**

5. Plot the results for each metric as bar charts for comparison. The bars are grouped by the storage engine type where each bar reports the value for the respective configuration of connections, ramp up time and database parameters. **[30 points]**

6. **[Bonus/ Optional]** Adapt the benchmark to work for the Postgres database. You will at least need to change the JDBC connection string and you might need to make changes to the SQL statements in the TpccStatements.java file. **[20 points]**

Some hints for the Bonus section:

Adapting the benchmark for PostgreSQL might require the following changes, according to https://www.percona.com/blog/2018/04/19/sysbench-tpcc-supports-postgresql-no-really-this-time/

"1. It appears that PostgreSQL does not support the tinyint and datetime data types. We had to use smallint and timestamp fields, even if using smallint makes the database size bigger.
*2. PostgreSQL does not have a simple equivalent for MySQL's SHOW TABLES. The best replacement we found is select * from pg_catalog.pg_tables where schemaname != 'information_schema' and schemaname != 'pg_catalog'.*
3. PostgreSQL does not have a way to disable Foreign Key checks like MySQL: SET FOREIGN_KEY_CHECKS=0. With PostgreSQL, we needed to create and load tables in a very specific order to avoid Foreign Keys violations.

4. PostgreSQL requires you to have a unique index name per the whole database, white MySQL requires it only per table. So instead of using:
CREATE INDEX idx_customer ON customer1 (c_w_id,c_d_id,c_last,c_first)
CREATE INDEX idx_customer ON customer2 (c_w_id,c_d_id,c_last,c_first)
we used
CREATE INDEX idx_customer1 ON customer1 (c_w_id,c_d_id,c_last,c_first)
CREATE INDEX idx_customer2 ON customer2 (c_w_id,c_d_id,c_last,c_first)
5. PostgreSQL does not have a STRAIGHT_JOIN hint, so we had to remove this from queries. But it is worth mentioning we use STRAIGHT_JOIN mostly as a hack to force MySQL to use a correct execution plan for one of the queries.
6. PostgreSQL is very strict on GROUP BY queries. All fields that are not in the GROUP BY clause must use an aggregation function. So PostgreSQL complained on queries like SELECT d_w_id,sum(d_ytd)-w_ytd diff FROM district,warehouse WHERE d_w_id=w_id AND w_id=1 GROUP BY d_w_id even when we know that only single value for w_ytd is possible. We had to rewrite this query as SELECT d_w_id,SUM(d_ytd)-MAX(w_ytd) diff FROM district,warehouse WHERE d_w_id=w_id AND w_id=1 GROUP BY d_w_id."

Deliverables

1. A report about the running results of the benchmark for the different configurations. The report shall briefly describe the configuration of MySQL. It shall also describe the changes you made to the configuration. Finally, the report shall include the plots described in the requirements section.

2. All raw data generated by each run of different benchmarks should also be submitted in a compressed file.