AWS Final Project: MySQL benchmarking and proxy pattern

December 13, 2022

Victor Mesterton - victor.mesterton@polymtl.ca

Abstract

This report presents the written elaboration of the final project for the Advanced Cloud Computing lecture of École Polytechnique de Montréal (LOG8415E). The assignment was to first, set up and perform a benchmarking between a stand-alone MySQL server and a MySQL cluster. To discover performance differences between the DB setups. To conclude, the MySQL cluster performed better than the stand-alone server in terms of efficiency and amount of handled requests, with a downside that it requires more computing power.

1 Introduction

The final assignment was to learn how to set up MySQL on an Amazon Web Services EC2 instance. MySQL was chosen as it is a relational database that allows for both stand alone and distributed setups. In addition, we were supposed to practice creating a cloud design pattern, in this case, a proxy pattern. A python script using boto3 AWS SDK and shell scripts was used to set up the EC2 instances required for the databases and patterns.

The report is structured in the following way. Firstly, the MySQL server and cluster setups are discussed including their benchmarking against each other. Followed by conclusions and instructions on how to set up the MySQL databases.

The code used in the project is available at github: https://github.com/miniton98/CloudComputing

2 Benchmarking MySQL stand-alone vs. MySQL cluster

The stand-alone MySQL server and a MySQL cluster were benchmarked against each other to discover performance differences between the setups. The benchmarking was performed using Sysbench, which is an open source becnhmarking software. To perform the benchmarking was the Sakila example databases downloaded on both MySQL instances. The Sysbench software was then run on the Sakila databases to perform the benchmarking. The following sections will provide more detailed information regarding the MySQL instance setups.

2.1 MySQL Stand-Alone

As the name suggests is the stand-alone instance, a single EC2 T2.micro instance running MySQL 8. The instance including security groups are set up using a python script. During the instance creation stage is shell script passed to the instance as user data. The shell script is used to install all required software, such as the Sakila database, MySQL and Sysbech. MySQL was installed using secure installation, which included setting up a password for the root user, deleting all existing users, the testdb and flushing all privliges. After this was a new personal user created and the privliges to operate the sakila database. The new user was the used to run the sysbench benchmarking software. The entire process is automated and can be performed by simply running the provided python script.

2.2 MySQL Cluster

The MySQL cluster is a total of 4 EC2 T2.micro instances. One of the instances functions as the primary, while the three others are secondary data nodes. The primary is the main

instance that function as a load balancer for the three remaining nodes, that perform most of the queries. The cluster is a more complex set-up, as we need to install special packages and connect all instances to function together. All instances are created using a python script. However, to complete the set up are manual steps required, such as modifying files where after SSH:ing into the instance and running shell scripts.

2.2.1 MySQL Cluster set up

1. The primary instance required the most setting up. Firstly, the MySQL cluster manager node software needed to be installed, note this is a different software from the stand-alone MySQL. Secondly, we needed to define all the node, by uploading a file containing all the private IP:s. Where after we could start the manager node.

2. The next step was to configure all the data nodes. This required installing the MySQL cluster data node software. Where after a file was created to define the private IP of the manager node. Then the node could be started. This was repeated for all three data nodes.

3. The next step is to download MySQL on the management node and define the private IP. Where after MySQL is restarted and everything is checked to be running accordingly. We make sure the cluster nodes are connected and running.

4. Lastly, install Sakila and Sysbench on the primary node. These are required for benchmarking. We set-up the same test as for the stand-alone and run the benchmarking.

2.3 Performing benchmarking

Both benchmarking scenarios were performed using Sysbench and time based benchmarking. To perform the benchmarking we need to run 2 commands. The first command prepares the benchmarking. Here we define the properties we want to use. We are using read-write benchmarking on MySQL, using the Sakila database. Additionally, we create 6 tables and the table size is 100 000. After the prepare statement, we run the benchmarking. We define the same things as for the prepare statement with an addition that we use 6 threads, run the benchmarking for 60 seconds and lastly save the results in the defined text file. The same queries are used for the cluster to perform the same test.

```
sysbench oltp_read_write --db-driver=mysql --mysql-user=victor
   --mysql_password=password --mysql-db=sakila --tables=6 --
   table-size=100000  prepare|

sysbench oltp_read_write --db-driver=mysql --mysql-user=victor
   --mysql_password=password --mysql-db=sakila --tables=6 --
   table-size=100000 --threads=6 --time=60  run > SA.txt|
```

2.4 Performance comparison of stand-alone vs. cluster

This section describes the results of the benchmarking between the stand-alone server and the cluster. More precise numbers can be seen in Tables 1 and 2. These numbers are from a single iteration, thus, minor differences may occur depending on when the benchmarking is

executed. However, the differences between the stand-alone and cluster are so large that we can draw conclusions from them.

Comparing the efficiency and number of queries performed by both servers during the defined testing time of 60 seconds. The results can be viewed in table 1, where we can see that the cluster is performing more queries than the stand-alone. The increased efficiency can be seen in both read and write queries, including transactions.

Table 2 lists the differences in the latency between the set-ups. On average is the cluster performing better also with lower minimum latency. However, the max latency is multiple times larger than for the stand-alone, indicating worse worst-case performance. This might also be due to a slower warm-up and establishing the connections the first time.

The cluster performs better than the stand-alone. However, the cluster does not perform 3 to 4 times better, indicating that some performance is lost in the communication and assignment of queries. Thus, the stand-alone cluster performs on a higher level than the cluster nodes. Also considering that the cluster requires more instances and more resources. In general, it is a more complex setup, but it also provides some redundancy as it is possible to create replicas as backups. Lastly, we can most probably also get minor performance differences from the differences in software. The stand-alone cluster is running ubuntu 20.04 and MySQL 8 while the cluster is running ubuntu 18.04 and MySQL 7. Choosing between the set-ups is dependent on the requirements of the database server and the available resources.

SQL Statistics	Stand-Alone	Cluster
Read Queries	226 016	327 096
Write Queries	64 576	93 456
Other Queries	32 288	46 728
Total Queries	322 880	467 280
Queries/sec	5 378.91	7 786.02
Transactions	16 144	23 364
Transactions/sec	268.95	389.3

Table 1: Number of queries run on the databases during 60 sec.

Latency (ms)	Stand-Alone	Cluster
Min	8.38	4.03
Avg	22.3	15.41
Max	132.29	987.41
95th percentile	28.67	23.52

Table 2: The latency during benchmarking.

3 Proxy Design Pattern

The proxy design pattern was not implemented.

4 Conclusion

To conclude only the benchmarking between a stand-alone MySQL database instance and a distributed MySQL cluster was performed. From the benchmarking, it was possible to see the performance difference from horizontally scaling the MySQL database. The cluster was more efficient than the stand-alone. However, it also required a more complex setup and 4 instances compared to 1. The selection between the database types should be based on the use cases of the application. Furthermore, if a distributed system is required is it worth checking into if a NoSQL database would be a better fit.

5 Code instructions

This section provides a step-by-step instruction to run the stand-alone implementation with ease.

1. Download the AWS Command Line Interface and install it
2. Learner Lap ≫ AWS Details ≫ AWS CLI Show
3. Execute `aws configure` in the command line
4. Copy and paste the suiting credentials from step 2 and *region=us-east-1; output=json*
5. Unzip the compressed project file to your location of choice
6. Learner Lap ≫ AWS Details ≫ Download PEM
7. Copy the `labsuser.pem` file into the project directory of CloudComputing
8. Navigate with your command line to the location where you stored the project folder
9. The script requires you to pip install `boto3, paramiko` via the command line. Ignore if already done.
10. Run the Python script of the stand-alone implementation with `python3 SA_MySQL.py` in your command line

5.0.1 Running the MySQL Cluster set-up

1. Set-up everything for the AWS CLI and SDK as described in the previous steps 1-9
2. Run the Python script of the cluster implementation with `python3 Cluster_MySQL.py` in your command line
3. The implementation will print the private IP addresses of all created instances to the console. Copy and paste the private IP address of the primary node to the following places:
   (a) rows 11 and 31 in primary18.sh
   (b) row 6 in secondary18.sh
   (c) row 19 in primaryMYSQL.sh

4. Copy and paste the private IP address of the secondary nodes to rows 15, 20, 25 of primary18.sh (one IP per line)

5. SSH into primary node, password ubuntu, and run the entire primary18.sh script by copy-pasting it to the shell.

6. SSH into all secondary nodes, password ubuntu, and run the entire secondary18.sh script by copy-pasting it to the shell.

7. Go back to primary node. copy-paste rows 3-8 of primaryMYSQL.sh to the shell. Wait for the installation and insert desired root password for MySQL.

8. Run command `sudo nano /etc/mysql/my.cnf` on the shell, row 11, this opens up a new shell with text.

9. Copy and Paste rows 13-19 of primaryMYSQL.sh after all existing text.

10. press CTRL + O, press ENTER, and press CTRL + X to save and exit the made changes.

11. run `sudo systemctl restart mysql` and `sudo systemctl enable mysql`, rows 26-27

12. Now we can check if everything is set up correctly by running `ndb_mgm` in the root directory and then in the new shell insert `show`. Everything is OK if the output is similar to Figure 1

13. Run command `mysql -u root -p` and insert password. This opens up a new shell.

14. Alternatively check the status by running command `SHOW ENGINE NDB STATUS \G`, row 40 of primaryMYSQL.sh

15. Run the commands on rows 41-47 of primaryMYSQL.sh, then exit the shell by typing `exit`

16. Then run the rows 50-53, to perform the benchmarking

17. To read the benchmarking file run `cat Cluster.txt`, row 54, this will print the results into the shell

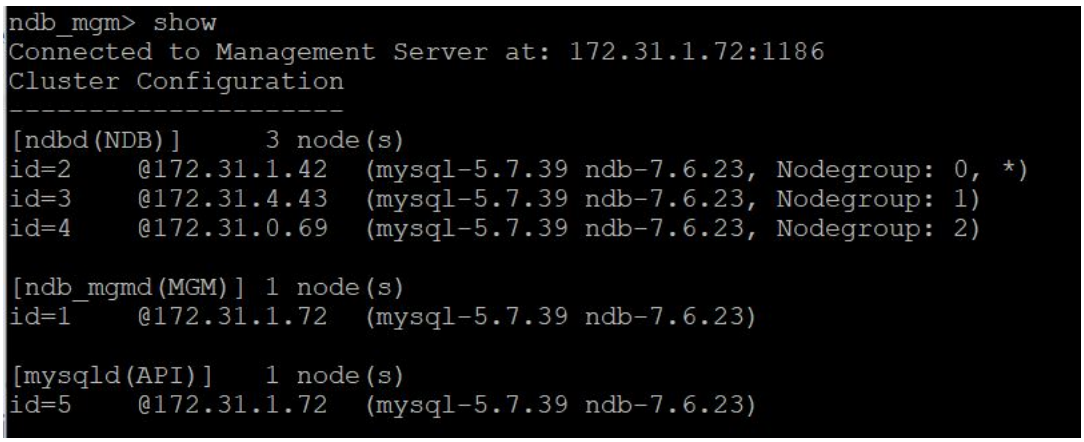

Figure 1: example output if cluster is set up correctly