AWS Individual Project

December 22, 2022

Aura Kiiskinen - aura-aksiina.kiiskinen@polymtl.ca - 2228758

Abstract

This report presents the written elaboration of the final project for the Advanced Cloud Computing course of École Polytechnique de Montréal (LOG8415E). The report focuses on benchmarking MySQL stand-alone vs. MySQL Cluster.

1 Introduction

The aim of the first part of the project was to create MySQL stand-alone instance, MySQL cluster as well benchmark the performance between the stand-alone and the cluster. On the second part, the aim was to create two cloud patterns, proxy and gatekeeper, and implement and deploy an architecture by adding these on the cluster. This report contains the detailed description of the first part of the project, how the implementation works and what is needed to run the code.

The first section of the report focuses on the stand-alone and cluster benchmarking. This section contains definitions of the related terminology as well as description of the instance set-up. It will also cover the results of the comparison. The second section covers the code instructions.

2 MySQL stand-alone vs MySQL Cluster

2.1 Background

- MySQL: SQL data base management system that is open source and developed by Oracle Corporation.
- MySQL Stand-alone: MySQL is installed on one instance and the storage is not shared between multiple instances.
- MySQL Cluster: Technology using distributed computing architecture and MySQL, cluster consists of data nodes that store data, management nodes that are responsible for configuration and monitoring as well as SQL node that is the MySQL server connecting to the data nodes.
- Sakila: Sample database providing a standard schema that can be used in examples such as in books or articles.
- Sysbench: Multi-purpose open source tool designed for Linux systems to help with benchmarking. In this project sysbench is used for database performance testing.

2.2 Instance set-up and benchmarking with sysbench

The Github repository providing all necessary code and information is available at: https://github.com/aurakiis/cc-poly-aura Use the 'script.py' script to create the instances.

I automated the creation of one t2.micro instance for the stand-alone as well four t2.micro instances for the cluster. The instances are using us-east-1a availability zone and security group that allows all traffic. All traffic is used since the cluster requires wider inbound rules in order to allow the communication between the instances.

2.2.1 MySQL stand-alone

The MySQL stand-alone is fully automated. While the single instance is created the user data contains all the required commands. This includes installing MySQL server and sysbench, secure MySQL installation commands, downloading sakila database, configuring the database as well as sysbench test. While the sysbench test is executed the script creates a file on the instance and stores the results on it so that they can later be retrieved and compared. On both, stand-alone and cluster, I run prepare and run commands with sysbench for the read-write test. The first one prepares the sakila database by creating the needed tables with the table sizes of 1000000. The second command is to run the test which will create the results files that can later be compared.

2.2.2 MySQL cluster

MySQL cluster consists of one management node and three data nodes. The management node also works as the SQL server. The setup is not fully automated, and after the data nodes have started, there is a need to connect to the management node and execute three files in order to finalize the MySQL server setup, sakila database configuration and run the sysbench test. This requires three commands on the management node and after the execution, a results file is created on the management node. The automatic part consists of multiple stages which will be described in the following paragraphs. These differ between the one management node and three data nodes.

Both data nodes and management node have the commands to download and extract the MySQL cluster binaries as well as the modifications on the path in their user data. On top of this the management node's user data has commands to create the deployment directory and my.cnf file which contains configuration for the MySQL daemon, including for example the port configuration. The data nodes' user data has command to create a data directory.

On top of the user data commands, I use paramiko in order to connect to the cluster's instances, write executable files and execute some of them automatically. I create five files for the management node and one for each data node. I decided to create executable files in order to run easily multiple commands on the instances, both automatically and later manually. Paramiko itself is not as adaptive to multiple commands and in order to prevent the need to connect many times to the instances and send commands in smaller batches, I decided to write files with the required commands on the instances and make them executable. For the paramiko connection, the user needs to download the .pem file from the AWS lab as well as add it to the root of the project. The script accesses the .pem file from the root and uses the public IP address of the instance for the connection.

First file on the management node has commands to install libncurses5 as well as to write config.ini file, which contains the configuration for the cluster, including the correct private IP addresses as hostnames for the management node and data nodes, node ids and paths to the data directories in the instances. The second file contains the commands to create the MySQL database as well as to start the management node. The first and second file are separated into different files because the first file is used for writing the config.ini file and the script had problems to recognize the end of file command. At the end, it required the removal of the

EOF-tag, so that the config.ini file was correctly written. The third file is ensuring the SQL node is started and fourth file has commands for secure MySQL installation, to download sakila database as well as to setup the sakila database. Finally the fifth file is to install sysbench and to run both prepare and run commands in order to execute sysbench test. While the run command is executed it will create a results.txt file on the management node which can be used to compare the results with the stand-alone. The last three files are separated into their own files because the start of the SQL node is a process that does not end automatically and executing it in a separate file ensures that the following commands are executed as well. Sakila and sysbench are separated in order to make it possible to experiment manually with the sakila database if needed before the sysbench test. However, the separation is not necessary and they could be combined. On top of the management node files, each data node needs an executable file which contains the commands to install libncurses5 and start the data node.

After the files are created, paramiko is used for executing the first two files on the management node as well as the files on data nodes to start them. Therefore the automation of the solution ends to the phase where the management node and the data nodes have started. After this the user needs to connect to the management node and execute the last three files. This phase is done manually since there were problems with the automatic execution. Manual execution works well and after that, the cluster is running and there are results of the sysbench benchmarking of the both MySQL stand-alone and MySQL cluster database performance.

2.3 Performance comparison of stand-alone vs cluster

After the sysbench tests are executed on both stand-alone and cluster, the user needs to retrieve and read the results.txt files in order to see the results. This can be done by connecting to the instances, to the stand-alone as well as to the management node of the cluster. The file should be found from the root of the both instances.

While I executed the sysbench read-write test and retrieved the results from both, the results showed that the stand-alone had less transactions than the cluster. The amount of transactions per second was higher for the cluster indicating better performance. While the cluster had around 215 transactions per second, the stand-alone had 109. Also, the total number of transactions was doubled in the cluster. The cluster had also less latency as the stand-alone meaning it was able to handle the queue faster. The average latency of the cluster was half of the stand-alone.

Both tests ran around 10 seconds and the division between the read, write and other queries were the same. This is necessary for the comparison. On both, standalone and cluster, 70% of the total amount of the queries were read queries, 20% were write queries and 10% were others. The amount of total queries within the cluster were double the amount of the stand-alone within the same time. Neither one of them had any ignored errors or re-connections.

The results indicate that the the performance of the cluster was better, doubling the amount of transactions within the same time as well as dropping the latency to half. However, the standalone is utilising one t2.micro instance while the cluster is using four t2.micro instances. Therefore while the performance is better with the distributed computing, it's still utilising

Metric	Stand-alone	Cluster
Transactions	1094	2152
Transactions (Per sec.)	109.24	215.12
Latency (ms) - Avg.	9.14	4.64

Table 1: Comparison of the stand-alone and cluster: number of transactions and avarage latency

Metric	Stand-alone	Cluster
Read Queries	15316 (70%)	30128 (70%)
Write Queries	4376 (20%)	8608 (20%)
Other Queries	2188 (10%)	4304 (10%)
Total Amount of Queries	43040 (100%)	21880 (100%)
Ignored errors	0	0
Reconnects	0	0
Total Time	10.0126s	10.0018s

Table 2: Comparison of the stand-alone and cluster: number of queries, ignored errors, reconnects and total time

four times more resources than the stand-alone while it's increasing the performance only by doubling it.

3 Conclusion

3.1 Result summary

The sysbench tests on both MySQL stand-alone and MySQL cluster indicate that the cluster performed better, if comparing the amount of transactions in total, per second as well as the decreased latency. However, the cluster is utilizing four times the same amount of resources than the stand-alone, while it consists of four t2.micro instances instead of one. The amount of transactions in the same time doubled with the cluster and therefore the performance increase of the cluster did not match with the increased amount of resource utilization. However it can be useful for jobs that require higher availability and scalability since the latency was decreased and the absolute performance was better.

3.2 Code instructions

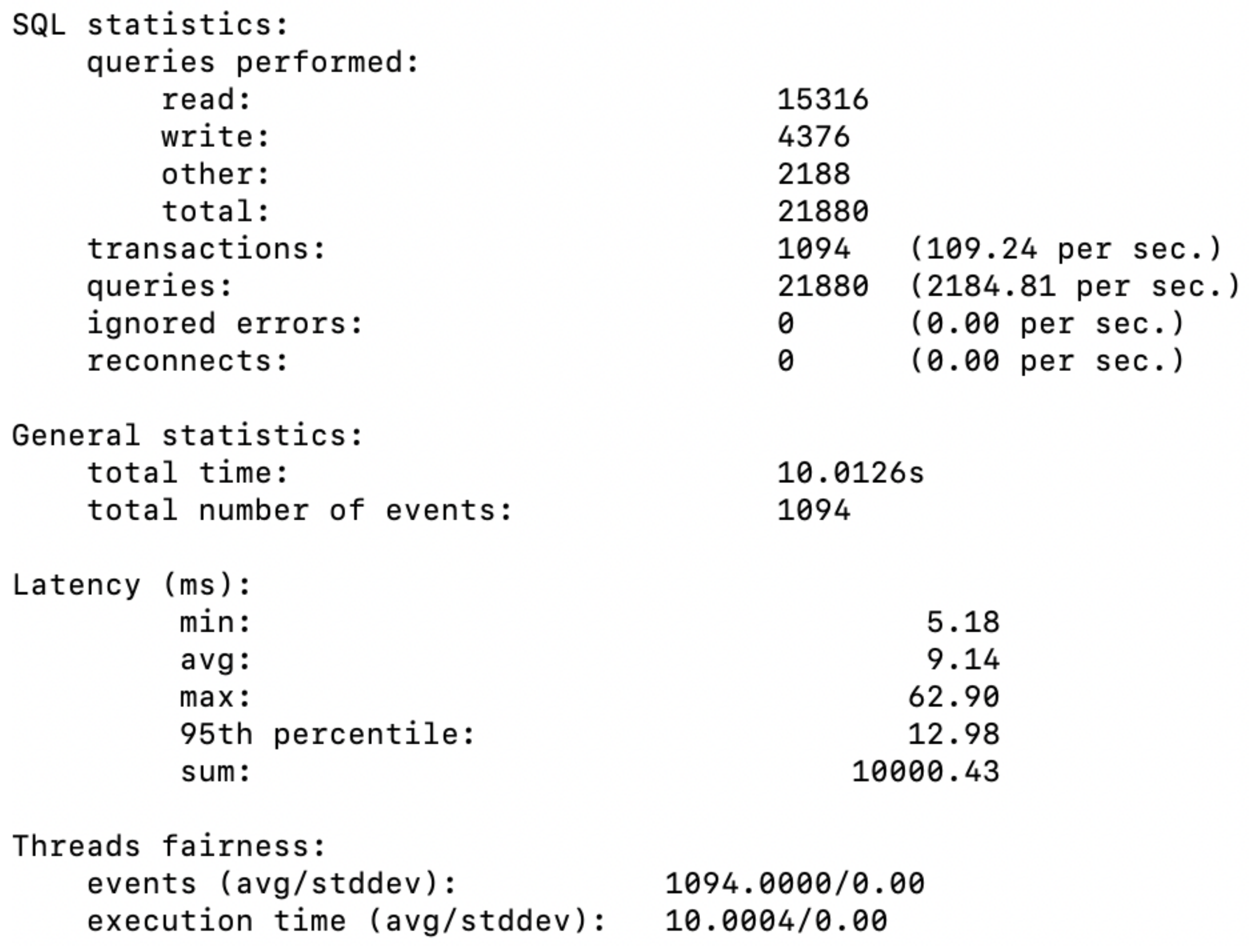
This section provides a step-by-step instruction to run the implementation.

1. Download the AWS Command Line Interface and install it
2. Learner Lab ≫ 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 wish
6. Learner Lab ≫ AWS Details ≫ Download PEM
7. Copy the `labuser.pem` file into the project directory of cc-poly-aura
8. Navigate with your command line to the root of the project
9. The script requires you to pip install `boto3, paramiko` via the command line. Ignore if already done.
10. Run the Python script with `python3 script.py` in your command line
11. After the script has finished ensure you are in the project root where you can find the .pem file and connect to the cluster with putty (Windows) or with the following command (Linux and Mac): `ssh -i labsuser.pem ubuntu@IP_ADDRESS`, the IP address to use for the connection is the one for `cluster` and printed in the terminal.
12. After the connection execute following command on the instance `/bin/bash mysql_execution.sh`
13. Execute command `/bin/bash mysql_execution2.sh`
14. Execute command `/bin/bash sysbench.sh`
15. Execute command `cat results.txt` to see the results
16. Read through the results or manually copy the results if needed for further comparison
17. Open another terminal window and navigate to the project root where you can find .pem file
18. Connect to the stand-alone instance with putty (Windows) or with the following command (Linux and Mac): `ssh -i labsuser.pem ubuntu@IP_ADDRESS`, the IP address to use for the connection is the one for `stand-alone` and printed in the terminal.
19. After the connection execute following command on the instance `cat results.txt` to see the results
20. Read through the results or manually copy the results if needed for further comparison
21. Run command `exit` on both instances in order to close the connections

4 Annexes

4.1 Results of the sysbench test on stand-alone



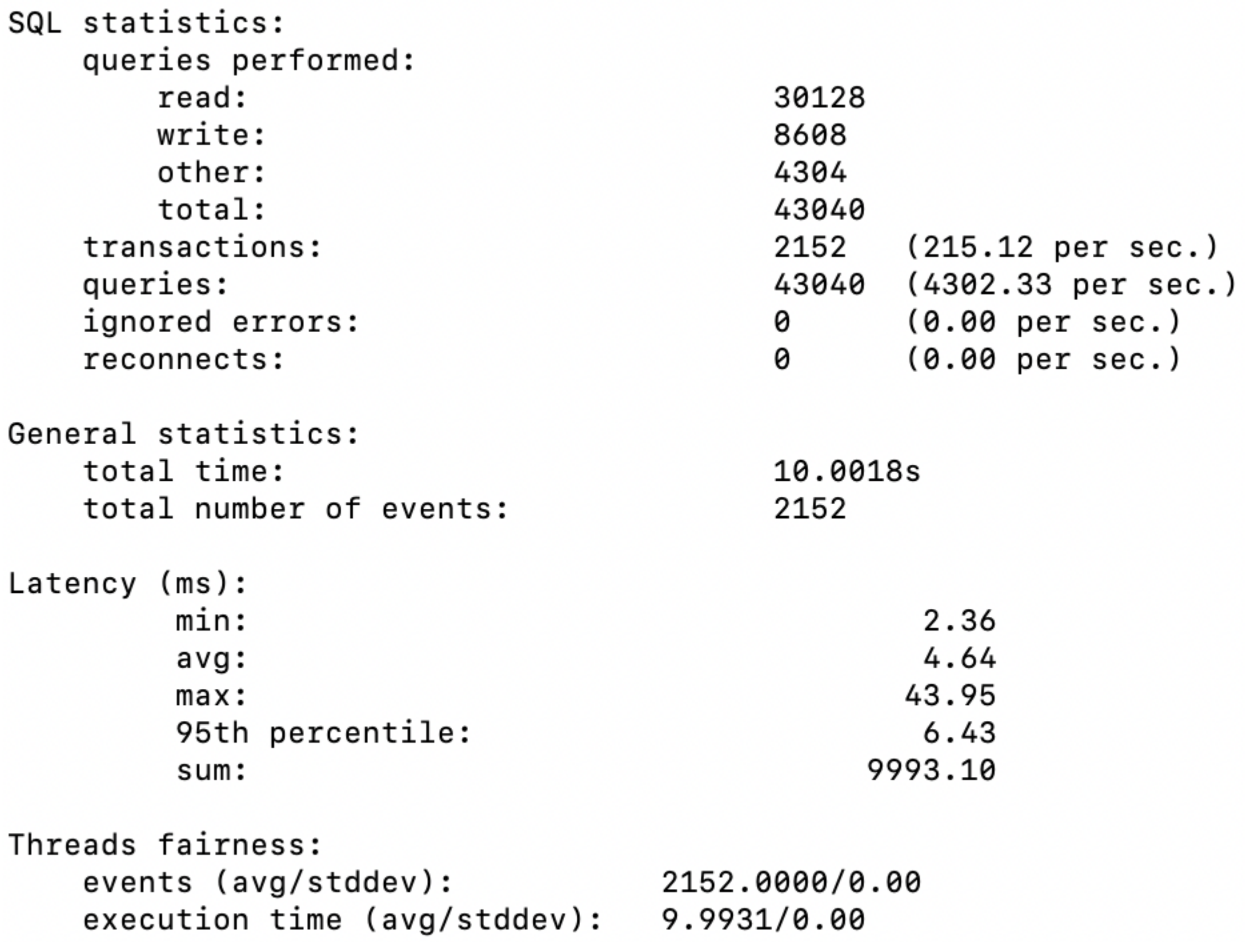
```
SQL statistics:
    queries performed:
        read:                            15316
        write:                           4376
        other:                           2188
        total:                           21880
    transactions:                        1094   (109.24 per sec.)
    queries:                             21880  (2184.81 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          10.0126s
    total number of events:              1094

Latency (ms):
         min:                                    5.18
         avg:                                    9.14
         max:                                   62.90
         95th percentile:                       12.98
         sum:                                10000.43

Threads fairness:
    events (avg/stddev):           1094.0000/0.00
    execution time (avg/stddev):   10.0004/0.00
```

4.2 Results of the sysbench test on cluster



```
SQL statistics:
    queries performed:
        read:                           30128
        write:                          8608
        other:                          4304
        total:                          43040
    transactions:                       2152   (215.12 per sec.)
    queries:                            43040  (4302.33 per sec.)
    ignored errors:                     0      (0.00 per sec.)
    reconnects:                         0      (0.00 per sec.)

General statistics:
    total time:                         10.0018s
    total number of events:             2152

Latency (ms):
         min:                                    2.36
         avg:                                    4.64
         max:                                   43.95
         95th percentile:                        6.43
         sum:                                 9993.10

Threads fairness:
    events (avg/stddev):           2152.0000/0.00
    execution time (avg/stddev):   9.9931/0.00
```