POLYTECHNIQUE DE MONTRÉAL

LOG8415e

ADVANCED CONCEPTS IN CLOUD COMPUTING

Mohamed Ben-Faras

2090619

December 31, 2023

Introduction

This personal project is separated into 2 parts. The first part consists of conducting a comparison between MySQL Stand-Alone and MySQL Cluster. The library used for benchmarking is sysbench, and the database employed is Sakila. We aim to achieve the most comprehensive results in this part by running both the cluster and the standalone version in a similar environment. The second part involves implementing cloud patterns: Proxy and Gatekeeper using the MySQL Cluster that has already been implemented and explaining the implementation details in depth.

1.1 MYSQL Stand-Alone Benchmark

For the MySQL Stand-Alone configuration, I needed to create a single t2.micro instance and execute a script named 'stand_alone_config.sh' to set up the instance and prepare it for benchmarking. The library used for benchmarking is sysbench, and the parameters are as follows:

- Six threads run simultaneously.
- The table size we are working on is 100,000.

The benchmarking results are as follows:

Queries	Amount
Read	253190
Write	72340
Other	36170
Total	361700

Table 1 : Amount of time a certain querie was performed

Latency	Time(ms)
Min	6.76
Avg	19.91
Max	86.70
Sum	360049.98

Table 2 : The duration of the latency

Threads Fairness	Avg/stddev
Event	3014.1667/9.51
Exec. Time	60.0083/0

Table 3 : Threads fairness

1.2 MySQL Cluster Benchmarking

For the MySQL Cluster configuration, I needed to create four t2.micro instances where one serves as a master, and the other three act as workers. Depending on the role, the script running during the instance initialization is different (cluster_master_config.sh / cluster_slave_config.sh). Once the four instances have been set up and are ready for benchmarking, the library used for benchmarking is sysbench, and the parameters are as follows:

-Six threads run simultaneously.
-The table size we are working on is 100,000.

The benchmarking results are as follows:

Queries	Amount
Read	336994
Write	96208
Other	48112
Total	481314

Table 1 : Amount of time a certain querie was performed

Latency	Time(ms)
Min	4.38
Avg	14.97
Max	156.28
Sum	359954.35

Table 2 : The duration of the latency

Threads Fairness	Avg/stddev
Event	4006.8333/7.31
Exec. Time	59.9924/0.00

Table 3 : Threads fairness

1.3 Analysis of the Benchmarking

The MySQL Cluster exhibited a higher transaction rate of 400.60 per second compared to the Stand-Alone configuration, which achieved 301.30 transactions per second. The latency for the MySQL Cluster was slightly lower, with an average latency of 14.97 ms compared to the Stand-Alone's 19.91 ms. Threads fairness analysis suggests that the MySQL Cluster had a more consistent distribution of events among threads with an average of 4006.83 events per thread, while the Stand-Alone had an average of 3014.17 events per thread. Overall, the MySQL Cluster demonstrated better transaction throughput and lower latency, making it a favorable choice for scenarios with higher transactional demands. The Stand-Alone configuration, while still performing well, showed slightly lower throughput and a marginally higher latency in comparison. The choice between the two configurations should be based on specific use case requirements and scalability considerations.

2. Proxy Cloud Pattern

We are going to implement a proxy pattern that requires having a t2.large EC2 instance between the client and the master node to process commands. For the proxy pattern, we are tasked with three implementations:

- Direct hit: we send the request directly to the manager
- Random hit: we send the request directly to a random worker
- Customized: we send the request to the server who has the lowest ping

It is important to note that I lost access to my AWS account after implementing the direct hit solution. As a result, I had limited resources to test my code, and I relied heavily on documentation for guidance.

2.1 Direct hit

For the direct hit, the implementation is carried out through the proxy, which sends commands to the master node. The process involves establishing a connection from the local machine to the proxy using the private key. Once connected to the proxy, a channel is opened to create a connection to the master node. This requires the proxy's private key and public key, as well as the public key of the target. Subsequently, I execute the commands that enable connection to the MySQL cluster and the execution of SQL queries.

The reason for establishing the connection in this manner, from the local machine to the master node, is to avoid the need to transfer the code to the proxy server and move the private key as well. This approach reduces the likelihood of encountering issues and enhances security, all while achieving the same objective. Finally, I write the result of my query on the local machine to verify its success.

2.2 Random hit

For the random hit, it follows the same structure as the direct hit. The difference is that I'll utilize the random module to randomly select a specific slave worker node that will process the query.

2.3 Customized hit

For the customized hit, the implementation is similar to the direct hit, but with a key difference. In this case, we check which server has the smallest ping. A function is called to determine this, and we process the queries on the chosen server.

3.Gatekeeper pattern

The Gatekeeper pattern consists of adding 2 t2.large instances where one will be the Gatekeeper and the other one will be the trusted host. In order to improve the security , only communication through the port 22 will be accepted by the gatekeeper. For the trusted host, we are going to accept the former configuration o the firewall and allow communication trough port 22 and 3306 for MYSQL cluster

4. Instructions to run your code

There are a few requirements that should be followed if we want to test this project. We need to have Python, Paramiko, boto3, and AWS CLI installed on the machine. Also, it is necessary to have a private key that will allow us to access our instances through the terminal.

4.1 MYSQL Standalone vs MYSQL Cluster

We need to start by choosing private IP addresses that will make the configuration easier. We chose IP addresses that start with 172.31.30.X. This is gonna be very important because it will make the creation of sh files easier. Next, run creation_micro.py, which will create 1 EC2 instance that has already been configured thanks to the stand_alone_config.sh file and 4 EC2 instances that have been configured on their side thanks to cluster_master_config.sh and cluster_slave_config.sh. Once this has been done, we need to obtain the public and private IPs of the proxy and the public IP of the master node. Afterward, run benchmark_micro.py, which will create 2 files containing the results of the benchmarking of MySQL Cluster and MySQL Standalone.

4.2 Proxy

For the proxy, we need to run the file creation_large.py to start the proxy. Then, we execute proxy_pattern.py with the argument, which is a query enclosed in single quotes. It should look like this:

py run proxy_pattern.py ‘SELECT * from staff;’

This will have the effect of creating 3 text files which will contain the result of the query because of the implementation. It is important to wait a little bit because there is a sleep time between running every implementation.

4.3 Gatekeeper

For the Gatekeeper, we need to follow the exact same thing we did for the proxy. Then, we will run trusted_host_set_up.py in order to set up the trusted hosts. Afterwards, we need to use scp in order to move the private key and the gatekeeper_pattern.py to the gatekeeper instance. And then, we will run gatekeeper_pattern.py on that instance that will reproduce a direct hit.

5. Github Access

https://github.com/BFaras/AWS-Project