Polytechnique Montréal

LOG8415 : Final Project

Advanced Concepts in Cloud Computing

Scaling Databases and Implementing Cloud Patterns

Author
Aleksandar Stijelja (1959772)

December 23, 2022

1 Abstract

In this lab assignment, I was tasked to setup a MYSQL stand-alone and MySQL cluster on Amazon EC2 and to implement and deploy an architecture by adding the Proxy pattern. In the first part of this lab assignment I installed, configured, and benchmarked a MySQL stand-alone using the Sakila Sample Database[1]. Then, I set up a MySQL cluster which I also committed a benchmark on, in order to analyse and compare the performance between stand-alone and cluster. Next, I tried applying my newly acquired skills from the course in order to implement the proxy Cloud pattern.

Keywords: AWS EC2, MYSQL Cluster, Benchmark, Sakila, Cloud Patterns

2 MYSQL Stand-Alone Benchmark

Regarding the benchmarking of the MYSQL stand-alone, I've made a script that launches a t2.micro EC2 instance on AWS. Also, the MYSQL server as well as sakila and sysbench, which is the tool used for the benchmark, have been installed in that instance. Once the the stand-alone server has been all set up, I've used sysbench to benchmark using the following paramaters:

- Run type: read and write
- Table size: 100 000
- Database: Sakila
- number of threads: 6
- Maximum time: 60 seconds
- Maximum requests: 0

In the following tables, here are the results from the benchmark of the MYSQL Stand-Alone:

Table 1: Stand-Alone Queries Performed

Queries Performed	MYSQL Stand-Alone
read	237454
write	67844
other	33922
total	339220
transactions	16961

Table 2: Stand-Alone Latency

Latency (ms)	MYSQL Stand-Alone
min	7.86
avg	21.22
max	112.19

3 MYSQL Cluster Benchmark

Regarding the benchmarking of the MYSQL cluster, I've made a script that launches four t2.micro EC2 instances on AWS. The cluster will have 1 master with 3 slave nodes. The master node is used for the write operations then replicated to the slaves, while read operations may be spread on multiple slave databases. Regarding the actual installation and configuration of the MYSQL on the cluster, I've heavily based myself on article [2]. To facilitate the configuration, all the instances for the cluster where launched on the (172.31.16.0/20) subnet. Here are the private DNS addresses of the instances when launched:

- Master: ip-172-31-17-1.ec2.internal
- Slave node 1: ip-172-31-17-2.ec2.internal
- Slave node 2: ip-172-31-17-3.ec2.internal
- Slave node 3: ip-172-31-17-4.ec2.internal

In the following tables, here are the results from the benchmark of the MYSQL cluster:

Table 3: Cluster Queries Performed

Queries Performed	MYSQL Stand-Alone
read	151214
write	43204
other	21602
total	216020
transactions	10801

Table 4: Cluster Latency

Latency (ms)	MYSQL Stand-Alone
min	19.90
avg	33.34
max	675.88

4 MYSQL Stand-Alone vs MYSQL Cluster

Now lets do a comparison of the between the MYSQL stand-alone vs MYSQL cluster. First, lets analyse the queries performed. When we look at tables 1 and 3, more specifically the total of queries, we can see that the stand-alone does about 1.5x more queries than the cluster. This came as a surprise as a cluster should be theoretically be doing more queries than a stand-alone. Requests are spread unto four nodes in our cluster, with the slaves doing the reads whilst master does the writes.

Now lets look at the latency results from table 2 and 4. if we compare the average latency, in milliseconds, between the stand-alone and the cluster, the cluster has an approximately 1.5x greater latency than the stand-alone. This can somewhat be explained by the additional network latency from the requests being routed across the 4 nodes.

It should be noted that, although the the stand-alone seems at first glance to be showing better performance than the cluster, it remains the cluster should be the optimal choice. In a more real life scenario, if we were to develop an

application using a relational database (MYSQL), chances are high that the database would be subject to very high quantities of requests per day, going as far as millions. A cluster would be better in this situation with its scalability due to being able to adapt to increase in demand by scaling out, simply by adding more instances as nodes.

5 Proxy Pattern

The proxy pattern uses data replication between master/slave databases to route requests and provide read scalability on a relational database, such as MYSQL. Write requests are handled by the master and replicated on its slaves, while read requests are processed by slaves. I've made a script that launches a t2.large instance and installs all necessary dependencies, such as sshtunnel and pymsql, who will be connecting and routing requests to the cluster. Also, because the cluster was launched on the (172.31.16.0/20) subnet, I made sure in the implementation of the proxy it is also launched on the same subnet, since we want the proxy and the MYSQL cluster to be in the same subnet. Once the script completes its setup, I SSH'ed into the proxy instance.

There were three strategies that the proxy was supposed to be able to complete, as follows:

- Direct hit: It simply directly forwards incoming requests to MySQL master node
- Random: The proxy randomly chooses a slave node on MySQL cluster and forward the request to it.
- Customized: The proxy measures the ping time of all the servers and forward the message to one with less response time.

Regarding the strategies that were tasked to implement, unfortunately I was only able to make the direct hit work, as the other two were riddled with connection problems to the nodes. Due to time constraints (I started this project on the 16th of december after the final exam), I wasn't able to debug the problems.

6 How To Run The Code

6.1 Preparation

You just need to make sure you have the following dependencies:

- Python 3
- boto3
- paramiko

Then make sure to install AWS CLI, if you already don't have it. Update you AWS credentials.

After you have git cloned the repo, take your "labsuser.pem" key and replace the "labsuser.pem" key from the git repo. Obviously don't commit and push the change. We just need your "labsuser.pem" key so we can ssh into instances and scp files as well. Make sure that your "labsuser.pem" key is chmod 400.

6.2 Set up and benchmark standalone

Make sure you are in the directory that has the python files, then run the following command:

```
python3 standalone.py "subnet-XXXXXXXXXXXXXXXXX"
```

Make sure to put the (172.31.16.0/20) subnet in double quotes as argument in the command. The script will take care of all the set up (security group and standalone instance), then once all is set up and ready it will benchmark the standalone and SCP the benchmark results file to the local machine. Afterwards, the script will end by terminating the standalone and deleting the security group.

6.3 Set up and benchmark cluster

Make sure you are in the directory that has the python files, then run the following command:

```
python3 cluster.py "subnet-XXXXXXXXXXXXXXXXX"
```

Make sure to put the (172.31.16.0/20) subnet in double quotes as argument in the command since our master and nodes will have private up addresses from this subnet. The script will take care of all the set up (security group and cluster instances), then once all is set up and ready it will benchmark the cluster and SCP the benchmark results file to the local machine.

6.4 Proxy

To test the proxy, you should run the cluster first with the command above, and not type the 'terminate' in the terminate once the message comes up. That way we will have the master and its 3 slaves running. Once that done, run the following command to set up the proxy instance:

```
python3 proxy_setup.py "subnet-XXXXXXXXXXXXXXXXX"
```

Again, make sure to put the (172.31.16.0/20) subnet in double quotes as argument in the command as we want the proxy to be in same subnet as the cluster.Regarding the strategies that were tasked to implement, unfortunately I was only able to make the direct hit work, as the other two were riddled with connection problems to the nodes. Due to time constraints (I started this project on the 16th of december after the final exam), I wasn't able to debug the problems.Direct hit is very straightfoward. It simply means the proxy will always target the master node to send queries.

After the proxy instance is all set up, the script will scp your "labsuser.pem" file into the proxy. Once you get the 'Ready!' message on the terminal, you can SSH into the proxy. Once in the proxy, git clone this git repo, then copy the proxy.py script from the repo into , where the labsuser.pem key is, such as this:

```
cp LOG8415-FinalProject/proxy.py ~
```

Make sure to chmod 400 labuser.pem.

The you can run the proxy. Make sure to put in the command the SQL query in double quotes, like this:

```
python3 proxy.py "SQL QUERY" >> results.txt
```

All prints from the proxy will be sent to the results.txt file for cleanliness. To test that we can actually get and write into the sakila db you can run the following three commands:

```
python3 proxy.py "SELECT * FROM actor" >> results.txt
```

```
python3 proxy.py "INSERT INTO actor VALUES (999,'DOE','JOHN','1999-06-30 09:09:09')" >> results.txt
```

```
python3 proxy.py "SELECT * FROM actor" >> results.txt
```

After running these 3 commands, you can "nano results.txt", you will see at first see a list of 200 values of actors, then the 2nd command will write a new John Doe actor at key 999, and you will see a 2nd list in results.txt where at the bottom you will see the newly added 201th element.

Figure 1: First Read Query of Actor List

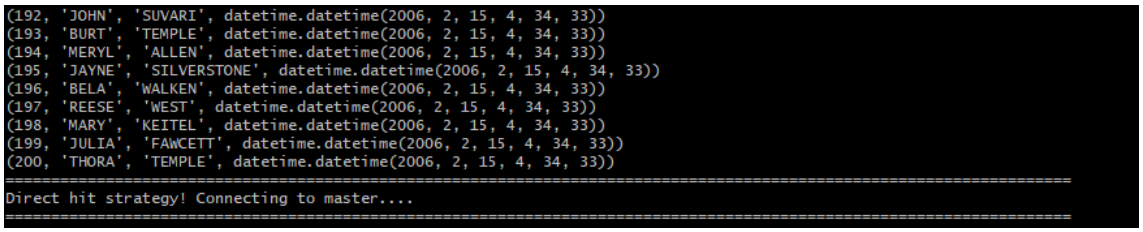
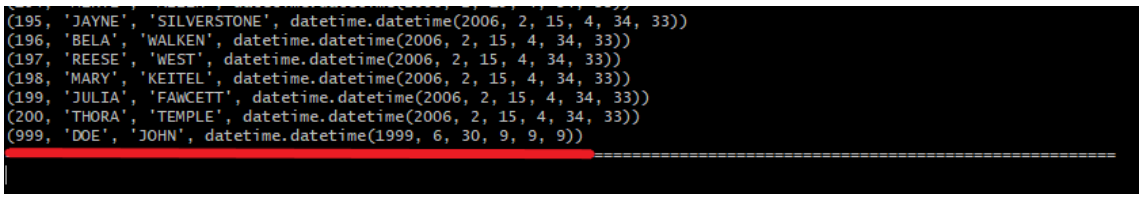


Figure 2: Second Read Query of Actor List After the Insert Query



7 GitHub Repository

The GitHub repository is the following:

`https://github.com/AleksStijelja/LOG8415-FinalProject`

8 Bibliography

1. Sakila Sample Database. https://dev.mysql.com/doc/sakila/en/
2. How To Create a Multi-Node MySQL Cluster on Ubuntu 18.04. https://www.digitalocean.com/community/tutorials/how-to-create-a-multi-node-mysql-c
3. Sysbench benchmark. https://www.jamescoyle.net/how-to/1131-benchmark-mysql-ser