Lab 3 - Final Project Report

Danilo Vaz - 2313099

January 1, 2024

1 Benchmarking MySQL stand-alone vs. MySQL Cluster

The first part of the project was setup and benchmarking a stand-alone distribution and a cluster distribution of the Mysql database. At first I tried to use a t2.micro ec2 instance for the cluster but I a faced the following error *Child process terminated by signal 9* while setting up the workers. After some troubleshooting I identify the problem as a lack of ram and therefore I used a m4.large ec2 instance all instance in the cluster and stand-alone distribution. For benchmark I used the sakilla database and the sysbench tool for measuring the read and write operations latency.

1.1 MySQL stand-alone

For setting up the stand-alone distribution a shell script was past through the *userdata* parameter to be executed when the instance is initialized. This script includes the installation of Mysql database, sakilla db, sysbench and execute the benchmark.

1.2 MySQL Cluster

For setting up the cluster distribution two different shell script were used one for the manage and other for the workers. They were past through the *userdata* parameter to be executed when the instance is initialized. The manager script includes the installation of Mysql cluster, sysbench and the initialization of the server. The worker script includes the installation of Mysql and the connection with the manager. Additionally is necessary to finish the setting the installation of manager manually because some commands requires user's actions. The script that needs to be executed manually is the *manualScript.sh*, its responsible for finish the installation of Mysql client, sakilla and execute the benchmark.

2 Implementation of The Proxy pattern

2.1 Proxy setup

For the setting up the proxy instance I used a nested script because the way I implement the proxy application the proxy needed the public ip of the gatekeeper hard-coded inside the app code and since the public ip is generated in runtime a nested script was needed to put the gatekeeper ip inside the proxy's script code. The proxy was launched in t2.large ec2 instance.

2.2 Proxy server

The proxy was implemented as a flask server. This server have only one route in which at first the proxy opens a ssh connection with the manager node, afterwards according to the method (direct hit, random or custom) a connection with the database will be established then the proxy will pose the query it received from the request and finally return the result to the client but prior to all these operation the client ip will be checked if its not the gatekeeper's public ip the service will be denied.

2.3 Proxy security

To improve security in the proxy a exclusive security group was created, this security group was designed to allow traffic coming only from the gatekeeper. Additionally the proxy server uses https, a protocol with encryption to handle packet sniffing attacks.

3 Implementation of The Gatekeeper pattern

3.1 Gatekeeper setup

For the setup of the gatekeeper I used an t2.large ec2 instance initialized by a shell script passed in the *userdata* parameter. Inside the script are all the commands to install python, create the flask server and to run the server.

3.2 Gatekeeper server

The gatekeeper was implemented as a flask server. This server has a list of all the public ips that are allowed to have access to the service, which is hard-coded and not modifiable. It also have 2 routes one to process the client's request and check if it comes from a allowed source and check the sql query to verify if its a valid query if both requirements are true then the request is forwarded to the secured host (in my implementation the proxy application). The other route is responsible for setting up the public ip of the proxy instance this route can only be accessed once, right after the launch of the proxy and only by an allowed ip, any other request for this route will be denied by the gatekeeper. Additionally another route could be implemented to add new ips (clients) to the allowed ip list.

3.3 Gatekeeper security

To further improve security in the gatekeeper the server uses https protocol, an encrypted protocol to better protect against packet sniffing attacks and also checks in all routes the source of the request to verify if comes from a client or from an unknown source.

4 Describe clearly how your implementation works

4.1 Benchmark

For the first part of the project I have implement 4 scripts to setup and benchmark the MySQL stand-alone and cluster distribution. 2 scripts are related to the manager node, one is passed to the instance during the initialization through the *userdata* parameter but because of the necessity of interactions to the user during the installation of the MySQL client another script to finish the installation was required in this script the .deb packages are installed and the benchmark is executed. For the worker node there is another script through the *userdata*. The last script is for the stand-alone and is also passed through the *userdata* parameter and it contains all the commands to install and benchmark the MySQL DB. All the instances used for the experiments were m4.large

4.2 Cloud Patterns

For the second part of the project was required the implementation of 2 cloud patterns: Gatekeeper and Proxy. Both were implemented as flask servers. The Gatekeeper server work as the frontier of the application it validates the source of the incoming request and the content of the request, if the validation doesn't return true the request will be denied but in case it does the gatekeeper will forward the request to the proxy server. The proxy server is responsible for process the request and retrieve the data from the data base but prior to the it will check the origin of the request to see if matches the public ip of the gatekeeper and then create a connection with the database and retrieve the data using the method select by the client.

The workflow of my implementation is:

1. Connect to the client
2. Create security group and key-pair
3. Launch worker nodes
4. Launch manager node
5. Launch Gatekeeper
6. Create security group for the proxy
7. Launch Proxy
8. Do a request to the *initiate* route of the Gatekeeper to inform proxy public ip
9. Executed the manual script in the manager node to finish the installation of the MySQL.

5 Summary of results and instructions to run your code

5.1 Results

5.1.1 Benchmark

```
ubuntu@ip-172-31-2-2:/opt$ ndb_mgm -e show
Connected to Management Server at: 172.31.2.2:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     3 node(s)
id=2    @172.31.2.3  (mysql-8.0.31 ndb-8.0.31, Nodegroup: 0, *)
id=3    @172.31.2.4  (mysql-8.0.31 ndb-8.0.31, Nodegroup: 0)
id=5    @172.31.2.5  (mysql-8.0.31 ndb-8.0.31, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @172.31.2.2  (mysql-8.0.31 ndb-8.0.31)

[mysqld(API)]   1 node(s)
id=6    @172.31.2.2  (mysql-8.0.31 ndb-8.0.31)

ubuntu@ip-172-31-2-2:/opt$ cat /home/ubuntu/results.txt
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 6
Initializing random number generator from current time


Initializing worker threads...

Threads started!

SQL statistics:
    queries performed:
        read:                            237510
        write:                           67860
        other:                           33930
        total:                           339300
    transactions:                        16965  (282.65 per sec.)
    queries:                             339300 (5653.05 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          60.0185s
    total number of events:              16965

Latency (ms):
         min:                                    8.18
         avg:                                   21.22
         max:                                   90.85
         95th percentile:                       26.68
         sum:                                360038.13

Threads fairness:
    events (avg/stddev):           2827.5000/5.91
    execution time (avg/stddev):   60.0064/0.00
```

Figure 1: Cluster benchmark results

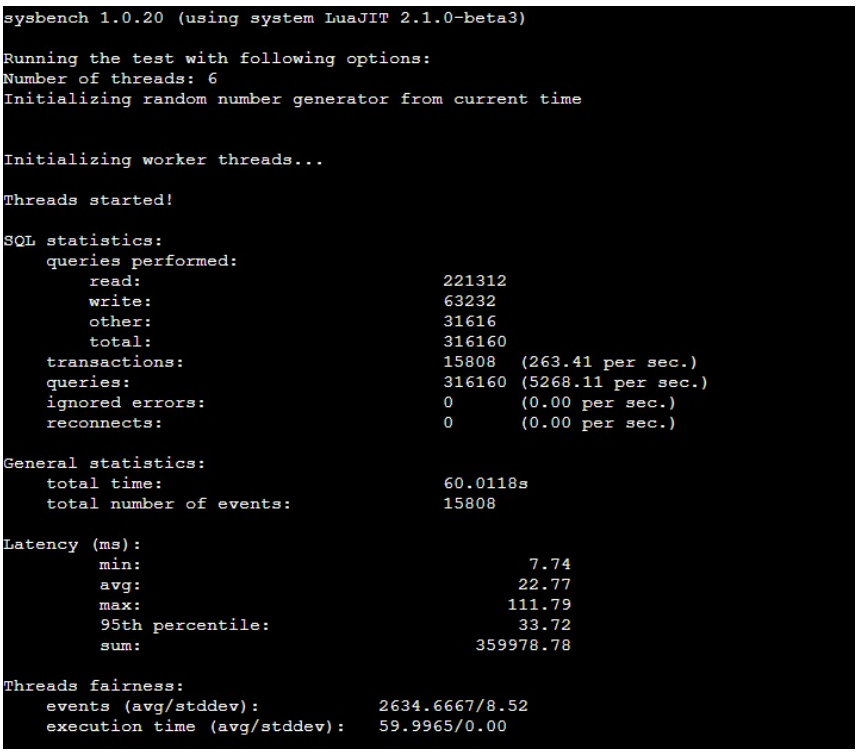

Figure 2: Stand-alone benchmark results

As we can see both results are similar regarding to latency. Looking at minimum latency value we can see that the stand-alone had a better result with 7.74ms against 8.18ms but looking to the the others statistics we can observe that the cluster has a better result with lower average, maximum and 95th percentile implying that the cluster has more constant and performative response to the queries.

5.2 Run the code

- Clone repo from Github
- The following values need to be stored in ∼/.aws/credentials:
  - AWS_ACCESS_KEY_ID
  - AWS_SECRET_ACCESS_KEY
  - AWS_SESSION_TOKEN
- cd to cloned repo
- cd to FinalLab folder

Once all that is done we run the following command

```
# Runs the application
python Main.py
```

Listing 1: Python script to run the application

Runs all the functions and scripts to setup as well as launch the Gatekeeper and the Proxy. After that run the *manualScript.sh* manually in the manager node terminal.