LOG8415
Final Project

Omar Khedr
Département Génie Informatique et Génie Logiciel
École Polytechnique de Montréal, Québec, Canada
omar.khedr@polymtl.ca

December 3rd 2024

1 Overall structure

The purpose of the assignment was to create a secure flow of SQL requests (READ or WRITE), going from an internet-facing t2 large GateKeeper instance, called GateKeeper, which validates the requests and forwards them to an internal and tightly secured t2 large GateKeeper instance, called TrustedHost. After which, the said request is forwarded to a t2 large Proxy instance, which redirects requests based on a pre-defined proxy pattern to one of 3 t2 micro instances in the cluster (Manager, Worker1, Worker2). As stated in the in the instructions for this final project, there are 3 potential proxy patterns: Direct (Sent directly to the Manager), Random (Sent randomly to one of the Workers), and Custom (Sent to the most responsive Worker). The cluster architecture is highlighted below:

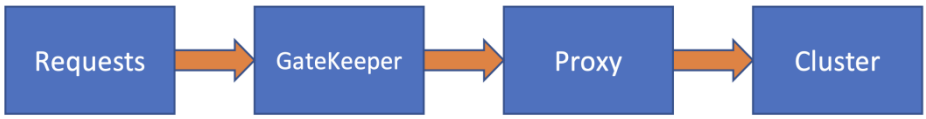

Figure 1: Cluster Architecture

A key part of this architecture is that write requests need to be uniquely handled by the Manager of the Cluster, where changes must be replicated on the corresponding Worker instances in the cluster.

2 MySQL standalone setup

In order to implement this architecture we needed MySQL on our Cluster instances. In addition, a database is needed to implement and test this workflow; to this end, the database sakila was used. In order to install MySQL standalone on our instances within the cluster, the following link was used: https://www.linode.com/docs/guides/install-mysql-on-ubuntu-14-04/.

After which, the sakila db was downloaded to the instances, and after connecting to MySQL server, we:

- Execute the sakila-schema.sql script to create the database structure.
- Execute the sakila-data.sql script to populate the database structure.

To implement the aforementioned steps, the following link was used:
https://dev.mysql.com/doc/sakila/en/sakila-installation.html

This is done in an automated fashion in the setUpMySQL.sh script, by running it on the instances using the paramiko library in Python. After the setup is complete, sysbench is installed and then run on each of the instances to benchmark that the setup is correct. Below, you will find the output of the aforementioned benchmark, as requested in the instructions of this final project:

```
SQL statistics:
    queries performed:
        read:                            118734
        write:                           0
        other:                           16962
        total:                           135696
    transactions:                        8481   (847.85 per sec.)
    queries:                             135696 (13565.54 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          10.0011s
    total number of events:              8481

Latency (ms):
         min:                                    0.83
         avg:                                    1.18
         max:                                   43.09
         95th percentile:                        2.00
         sum:                                 9975.71

Threads fairness:
    events (avg/stddev):           8481.0000/0.00
    execution time (avg/stddev):   9.9757/0.00
```

Figure 2: Output of desired sysbench benchmark

After setting up the MySQL standalone, Manager/Worker replication was implemented on the instances in the cluster. In order to implement this pattern, the following link was used:
https://www.digitalocean.com/community/tutorials/how-to-set-up-replication-in-mysql

Manager/Worker replication is implemented on our instances via the setUp*4Replication.sh scripts.

3 FastAPI setup

In order for the separate instances in this architecture to be able to forward requests to the upcoming instance, FastAPI applications were deployed on each instance on port 5000 using the paramiko library in Python, in conjunction with uvicorn and Boto3. Requests were sent through the architecture via http using the requests library in Python. Where logging is enabled as part of the benchmarking process, to show that requests are being properly handled by the instances.

Once the instance in the cluster receives the SQL query, it will be implemented on the MySQL server on that instance using the mysql.connector library in Python.

The Python scripts with the prefix app*.py, contain within them the necessary lines of code to run the FastAPI applications on their respective instances. These are deployed on the instances using the paramiko library in Python.

For this to work, a script getJsono.py is used, which outputs the private IPs of each of the instances by name in a file called: cluster_config.json. This file is then uploaded to all the instances, so the architecture can function in the desired fashion. Below, I will be outlining how the FastAPI stream works on each instance in the architecture:

- GateKeeper: An endpoint called /validate is used, in order to validate requests. In it, it checks whether the requests are in the right format, if they are, they are forwarded to the TrustedHost.
- TrustedHost: An endpoint called /process is used, to identify the desired course of action (READ or WRITE), as well as the proxy pattern (direct_write, direct_read, random, customized) in the request. Depending on the request and desired proxy, it is forwarded to the appropriate endpoint on the Proxy to be processed by the Manager / Workers.
- Proxy: Given that the Proxy instance has to handle so many potential requests and redirect them appropriately, it requires a more elaborate set of endpoints:
  - The /direct_write endpoint is used to forward WRITE requests directly to the Manager.
  - The /direct_read endpoint is used to forward READ requests directly to the Manager.
  - The /random endpoint is used to send the READ request randomly to one of the Workers.
  - The /customized endpoint is used to the send the READ request to the most responsive Worker, by pinging a /ping endpoint which is set up on both Workers.
- Manager: Has a /read endpoint for processing a direct_read and a /write for processing a direct_write.
- Workers: Has a /ping endpoint for receiving the aforementioned ping requests and a /read endpoint for receiving read requests.

4 Security

In order to ensure a streamlined and easy to understand security pattern, the following steps were taken. Please note that, in order for the architecture to work in the way outlined, security groups and their respective instances were started in the order listed below:

- Our internet-facing GateKeeper instance was created with its own security group, allowing incoming traffic from any IP, on port 5000 only. In addition, ssh on port 22 was permitted but only from my IP to allow us to deploy the FastAPI app via paramiko; this was done

programmatically by sending a get request to checkip.amazonaws.com, so should you run the submitted pipeline, it will only allow ssh from your ip.

- Our TrustedHost instance was created with its own security group as well, allowing incoming traffic only from the GateKeeper on port 5000 and ephemeral ports. In addition, ssh on port 22 is permitted during the setup from the IP running the pipeline; again, in order to deploy the FastAPI applications via paramiko. However, these are dropped after setting DROP rules in the IPtable setup, which allows traffic only from the GateKeeper instance (on port 5000) and only to the Proxy instance (on port 5000). Please find an output of the IPtable outputs in Figure 1, below:

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  172.31.18.130        anywhere             tcp dpt:5000
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             172.31.19.86         tcp dpt:5000
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
```

Figure 3: IPtable TrustedHost

- The Proxy instance was created with its own security group, allowing incoming traffic only from the TrustedHost on port 5000, as well as ssh on port 22 from the IP running the pipeline.
- The Manager and Worker instances all shared the same security group, where a more intricate ingress setup was necessary in order to implement Manager/Worker replication, and communication from the proxy, all the while securing the instances. The following was permitted for the instances in question:
  - ssh from port 22 from the IP running the pipeline.
  - ssh from port 22 from within the security group.
  - Port 3306 was open for instances within the security group to allow MySQL communication.
  - Port 5000 from the Proxy was permitted for FastAPI communication.

5 Benchmarking clusters

After all of the instances are running, Manager/Worker replication is set up on the Cluster instances, and FastAPI is running on all the instances, we are now ready to benchmark the architecture to show that it is working as desired.
To this end, we only WRITE and READ to the actor table in sakila, in the following fashion:

- WRITE: Writes my name - Omar Khedr - to the actor table in the sakila database. As requested, this is done 1000 times.
- READ: Reads the last entry in the actor table in the sakila database. As requested, this is done 1000 times for each potential proxy pattern.

These requests are sent to the GateKeeper instance in parallel using GNU parallel in conjunction with curl, where the total time it takes to execute is noted for benchmarking purposes in a file called benchmark_proxy_used.log; where proxy_used is the proxy being tested.
After which, the FastAPI log files of the 2 Worker instances and the Manager are downloaded, to benchmark if they successfully processed all 4000 requests. These will be downloaded automatically as part of the pipeline, and will be named ip_of_instance.log; where ip_of_instance is the IP of the Worker or Manager instance.

In addition, the actor table in the sakila database located in each of our instances is downloaded, to show indeed that my name - Omar Khedr - was written to the table 1000 times, and that this was replicated on the two Workers. Again, these will be downloaded automatically using mysqldump as part of the pipeline, and will be named actor_table_ip_of_instance.log.

6 Output

In this section I will be highlighting how the output of the benchmarking clearly shows that the architecture is working as expected. A status of 200 in the log files is used to note a successful processing by the endpoint. As we can see by the output below, Our Manager instance successfully processes 1000 direct_write and 1000 direct_read requests. In addition, we see that the other 2000 read requests using the random/custom proxy are distributed across the other two Worker instances, to add up to exactly 2000 successful requests:

```
(base) robomar@DESKTOP-FO0DKOA:/mnt/c/Users/Omar/Desktop/
cat 107.22.73.125.log | grep read | grep 200 | wc -l
1000
(base) robomar@DESKTOP-FO0DKOA:/mnt/c/Users/Omar/Desktop/
cat 107.22.73.125.log | grep write | grep 200 | wc -l
1000
(base) robomar@DESKTOP-FO0DKOA:/mnt/c/Users/Omar/Desktop/
cat 107.22.121.222.log | grep read | grep 200 | wc -l
1351
(base) robomar@DESKTOP-FO0DKOA:/mnt/c/Users/Omar/Desktop/
cat 34.201.242.246.log | grep read | grep 200 | wc -l
649
```

Figure 4: Successful handling of requests

To show that replication is working between the Manager and the Workers, we can simply compare the actor table which has been downloaded from the instances, and they should be identical. Below, you will find the output of the diff function, clearly showing the only difference being the time when the dump from mysqldump was completed:

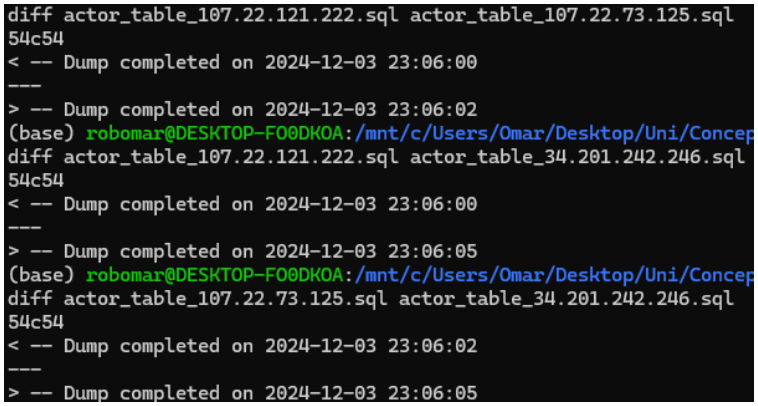

Figure 5: Successful Manager/Worker replication

After this, we can simply show the last few rows of the sql file, in order to show that my name is written 1000 times (Note: The number of rows in the actor table in sakila is 200):

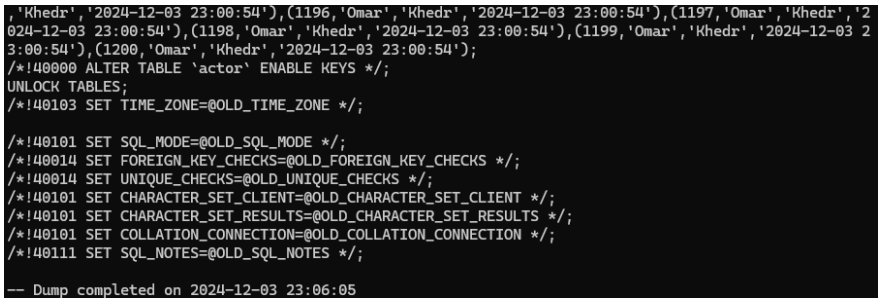

Figure 6: New actor table

As we can see, the last entry in the actor table is number 1200, where my name is inputted. Meaning my name was indeed written to the database 1000 times on the Manager, and replicated on the Workers.

7 Instructions to run the code

The codes are located in the following GitHub repository:

https://github.com/khedro/MySQL_cluster/

To run the code follow the following steps:

- Modify your /.aws/credentials file appropriately.
- Navigate to the folder with the scripts, and run: bash runPipeline.sh
- It may ask you to enter your user password at the beginning and end of the pipeline. Type your password and hit enter to continue.