LOG8415
Advanced Concepts of Cloud Computing
Scaling Databases and Implementing Cloud Design Patterns

Vahid Majdinasab
Département Génie Informatique et Génie Logiciel
École Polytechnique de Montréal, Québec, Canada
vahid.majdinasab[at]polymtl.ca

Student(s)
Ana Karen López Baltazar (2311301)

GitHub:
https://github.com/karen-baltazar/LOG8415

December 28, 2023

1 Implementation Notes

Before proceeding with the detailed description of the implementation of this project, it should be clarified that the current implementation has not been thoroughly tested. This fact falls squarely on my responsibilities, as I did not adequately prepare to avoid the situation where I would lack the means and credits necessary to perform extensive testing due to a lack of foresight on my part. I acknowledge this situation and accept the possible sanctions, as I will not be able to fully verify the functionality of the implementation.

In this report, my focus is on providing a detailed conceptual explanation of how this implementation operates and the theoretical steps required for its execution. It is important to note that the development of this implementation was based on consulting various tutorials and materials, which will be detailed in the references. I thank you in advance for your understanding in this matter.

2 Benchmarking MySQL stand-alone vs. MySQL Cluster

2.0.1 Explanation

The overall approach to implementation focuses on automating the process. The necessary instances are first created in AWS and then the relevant configuration files are copied according to the role of each instance. In the `scripts.sh` file, benchmarking corresponds to the first three steps. The order of creation of the instances is as follows: stand-alone, master node, slave nodes and proxy.

Below is a brief explanation of each file:

- `scripts.sh`: This main script is responsible for creating the instances in AWS, updating the IP addresses in the scripts, and copying configuration files to the corresponding instances.
- `aws_setup.py`: This script is responsible for creating instances in Amazon Web Services (AWS). This script uses the `boto3` Python library to interact with the AWS API and create EC2 instances. [1]
- `master_setup.sh`: This script runs on the master node and configures the instance to function as the primary node in the cluster. [3]
- `slave_setup.sh`: This script runs on each slave node and configures the instances to function as secondary nodes in the cluster. [3]
- `mysql_stand_alone.sh`: This script is used in stand-alone environment and configures the instance to run MySQL independently. [2]
- `benchmark.sh`: This script performs performance tests and saves the results to a text file (`/home/ubuntu/results.txt`). [4] [6]

2.0.2 Execution Steps

To perform benchmarking (after executing `scripts.sh`), follow these steps:

Stand-alone environment:

1. Connect via SSH to the stand-alone instance.
2. Run `mysql_stand_alone.sh` on the corresponding instance.
3. Run `benchmark.sh` to perform performance tests and save the results to `/home/ubuntu/results.txt`.
4. To view the results, navigate to the `/home/ubuntu/` directory and open the file using a text editor such as `nano` or `cat`.

Cluster Environment:

1. Connect via SSH to the master node instance.
2. Run `master_setup.sh` on the master node.
3. Connect via SSH to each slave node and run `slave_setup.sh`.
4. Reconnect to the master node and run `benchmark.sh` to perform performance tests. The results will be saved in `/home/ubuntu/results.txt`.
5. To view the results, navigate to the `/home/ubuntu/` directory and open the file using a text editor such as `nano` or `cat`.

2.0.3 Results

In terms of results, although there is no practical demonstration of the implementation, instead what we can do is make inferences from them based on the characteristics and nature of the MySQL environments. Specifically, MySQL Cluster is designed to offer high availability and improved performance compared to a single node configuration [3]. When examining transaction statistics, which indicate the number of transactions completed and the speed in transactions per second [6], one would expect the cluster environment to outperform the stand-alone environment.

However, to obtain practical validation, testing in a real environment and verifying performance under varied loading conditions would be necessary.

2.0.4 Important note

During the execution of the scripts, errors may be encountered in the stand-alone environment related to the specific user configuration. Likewise, there may be errors or problems related to the configuration of the private IP addresses of the cluster nodes. It is recommended to verify and ensure the correct configuration of private IP addresses in configuration scripts such as`master_setup.sh` and `slave_setup.sh` before execution. Additionally, you are encouraged to review the IP address settings in the `proxy_setup.py` script later in the process. The accuracy of these settings is crucial to guarantee test results.

3 Cloud Patterns: Proxy

In this case, the approach taken was to develop a Python file to run on the proxy instance, where, depending on the configuration or implementation chosen, the query is processed and sent to the appropriate node in the cluster. For this configuration file `proxy_setup.py`, MySQL Connector/Python [7] is used to connect to the cluster. As mentioned above, it is necessary to verify that the IP addresses in the file correspond to the private IP addresses of our cluster.

3.1 `proxy_setup.py`

The `proxy_setup.py` script acts as an intermediary between the MySQL cluster instances. Depending on the selected implementation (`direct_hit`, `random`, `customized`), it processes the queries and sends them to the corresponding node in the cluster.

3.2 Execution Steps

Make sure you have completed the cluster configuration before running the proxy.

1. Connect via SSH to the proxy instance.
2. Run the following commands to install the dependencies:

```
sudo apt update
sudo apt install python3
pip3 install mysql-connector-python
pip3 install ping3
```

3. Run the Python file with the desired implementation:

```
python3 proxy_setup.py <implementation>
```

   Replace `<implementation>` with one of the following options: `direct_hit`, `random`, `customized`.

3.3 Important note

Note that in the current implementation, the SQL queries are predefined in the `proxy_setup.py` script for each execution case (direct_hit, random, customized). This means that the proxy behavior is designed to execute specific queries based on the selected strategy.

For greater flexibility, it is recommended that you make adjustments to your code to make queries dynamic or accept the query as an input parameter. This will allow the script to handle variable queries based on the specific needs of the application.

4 Cloud Patterns: Gatekeeper

5 References

1. Boto3 Documentation
   `https://boto3.amazonaws.com/v1/documentation/api/latest/index.html`
2. How To Install MySQL on Ubuntu 20.04:
   `https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-20-04`
3. 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-cluster-on-ubuntu-18-04`
4. Sakila Sample Database:
   `https://dev.mysql.com/doc/sakila/en/sakila-installation.html`
5. Sakila Database Installation [**ALT**]:
   `https://www.sqliz.com/sakila/installation/`
6. Benchmark MySQL server Performance with Sysbench:
   `https://www.jamescoyle.net/how-to/1131-benchmark-mysql-server-performance-with-sysbench`
7. Connecting to MySQL Using Connector/Python:
   `https://dev.mysql.com/doc/connectors/en/connector-python-example-connecting.html`