AWS LAB Individual: MySQL Server - Standalone & Master-Cluster - including a Proxy & Gatekeeper Cloud Design on AWS

December 23, 2022

Nick Reiter - nick.reiter@polymtl.ca

Self-Decleration

I hereby declare that I have worked on the given project as precise and responsible as possible. Unfortunately, I was not completely able to finish the whole task due by a lack of knowledge concerning network communication to create a self-implemented Gatekeeper and Proxy pattern. I am Industrial Engineer and familiar with advanced programming tasks. Nevertheless, the second part exceeds my skills. Since the requirement of such in-depth knowledge was not pre-defined within the course's term schedule, I please you to take my academic background into account.

Abstract

This report presents the written elaboration of the third and individual assignment for the Advanced Cloud Computing lecture of École Polytechnique de Montréal (LOG8415E). The first section of the report documents two adoption models of a MySQL server on an Ubuntu device by using Sakila's benchmark data set. The second section explains the creation and connections of a Gatekeeper and Proxy cloud design pattern within the context of a MySQL server. These two cloud architectures determine the scalability and safety of our final implementation. Besides the practical elaboration, the theoretical basics of the cloud architectures are also explained.

1 Introduction

The target of the assignment was to create a semi-automatic script that is creating a standalone and master-slave MySQL server. The Proxy and the Gatekeeper pattern are meant to be initialized automatically, too. Results of the previous assignment are used and complemented by additional code snippets. Furthermore, the scope of this project requires a detailed description of the utilized cloud architecture.
The following sections of the documentation will explain each step more in detail in order to solve the previously described tasks. The first section focuses the MySQL server implementation. In particular, the differences background knowledge of technologies and other important terms are given. This is followed by the instance set-up, providing the hardware environment on AWS on which the benchmarking on Linux, Hadoop and Spark is running. Thirdly, the outcomes of the comparison and the influencing theoretical functioning are depicted. The second section mainly covers the development of a suitable solution for the social network problem. The first subsection explains, how MapReduce jobs can solve the social network problem. This is done by considering mapper and reducer separately. The algorithm is written in Java and the explanation of its structure is revealed in the second subsection. Lastly, a recommendation sample set is given to provide insights into the outcomes of the algorithms.

2 MySQL Server Benchmarking

2.1 Background

- *MySQL:* MySQL is a popular open-source Relational Database Management System (RDBMS) based on the Structured Query Language (SQL). SQL is a standard language used to create, update, delete, and retrieve information from a database. It is an attractive choice for web-based applications due to its scalability, reliability, and flexibility. It can be used to store and retrieve data from large databases quickly and efficiently.

- *Standalone Database Server:* A standalone database server is a single database server that is not connected to any other server or database. It is installed on a single computer and is used to store, organize, and manage the data of a single organization or application. A standalone database server is responsible for managing the data and the queries made against it. It is also responsible for providing security, scalability, and performance for the data.

- *Master-Slave Database Server:* A master-slave database server is a type of database replication in which a single master database server replicates data to multiple slave database servers. The master server is responsible for all read and write operations, while the slave database servers act as replicas that receive updates from the master server. This setup can help improve scalability, reliability, and performance, as well as provide increased availability in the event of a failure.
- *Sakila Database:* The Sakila database is a sample database designed to help users learn how to use MySQL. It is a free and open-source database created by Oracle. The Sakila database consists of 16 tables that contain data related to a fictitious movie rental store. The tables contain data such as customer information, film information, rental information, and payment information.
- *Sysbench:* Sysbench is a cross-platform and multi-threaded benchmarking tool used for testing and evaluating the performance of Linux systems.

2.2 Instance set-up

The Github repository providing all necessary code and information is available at:
https://github.com/elcaballero69/cc-individual-assignment

We automated the creation of five t2.micro instances. The instances are using us-east-1a availability zone and security group that allow complete SSH and HTTP traffic. The set-up for the standalone and master-slave MySQL server must be conducted manually through an SSH tunnel, e.g. PuTTY, and can be found in the associated shell scripts in the GitHub folder. Therefore, a connection for each instance must be built. The standalone server can be easily initiated and benchmarked with the help of sysbench by copying and paste the shell script. For the master and slave servers, the DNS IP address must be changed, which is indicated in the shell file. For a better understanding, a schedule is given below:
Standalone:

1. Run the `mysql_standalone.sh` script line by line in the terminal of your SSH tunnel

Master-Slave:

1. Note the private IP DNS addresses for your master and slave servers on the AWS console
2. Replace the private IP DNS addresses of the master server in line 49 of `mysql_master_init.py`
3. Replace the private IP DNS addresses of the slave servers in lines 56, 60, and 64 of `mysql_master_init.py`
4. Run the `mysql_master_init.py` script line by line of the server with the name *Master* in the terminal of your SSH tunnel
5. Replace the private IP DNS addresses of the master server in line 24 of `mysql_slave.py`
6. Run the `mysql_slave.py` script line by line of the servers with the name *Slave_1/2/3*
7. Start the *Master* server with the commands of `mysql_master_start.py`

8. Download Sakila and benchmark by inputting commands of `mysql_master_sakila.py` line by line. Change to current master's private IP DNS address between line 12 and 22.

2.3 Performance comparison of Standalone vs. Master-Slave

The results of the sysbench tests are outputted within the end of the bash files - look chapter 2.2. For the sake of clarity, only the read-write benchmark is used in this chapter.
We can see in the table below that the standalone had nearly the half of conducted transactions compared with its cluster counterpart. By analyzing the latency of processed queries, it can be said that the master-slave can handle request queries faster. The overall processing time amounts to around 11 seconds. What is striking, is that read (70%), write (20%) and other queries (10%) are equally distributed amongst both deployment methods. Nevertheless, the total amount of transactions is nearly doubled for the standalone deployment. Regarding the total amounts of transactions per sec, the master-slave deployment is again in lead. Accordingly, the standalone server processes 103.31 transactions per second, whereas the master-slave deployment computes 221.76.
From the results, we can derive that the cluster deployment reveals advantages in performance. Mainly, this can be traced back due to the utilization of four instances compared to one single instance of the standalone server. Indeed, the master-slave architecture uses a distributed computing approach, which provides four times more computing power but only doubles performance power.

Metric	Standalone	Master-Slave
Write Queries	10942 (20%)	5877 (20%)
Read Queries	38297 (70%)	20570 (70%)
Other Queries	5471 (10%)	2038 (10%)
Total Amount of Queries	54710	29380
Ignored Errors	0	0
Reconnects	0	1
Total Time	11.265s	11.189s
Transactions	1215	2398
Transactions/Sec	103.31	221.76
Latency (ms) - Avg.	9.86	4.31

3 Cloud Pattern Architectures

Within the sake of scalability, safety, performance, and response time, an additional cloud design pattern must be inserted as a predecessor of the master-slave MySQL server. Theoretical knowledge claims the Gatekeeper as most appropriate. Since we design the cloud design pattern completely by ourself, two patterns are required: Gatekeeper focuses on the security, whilst the Proxy concentrates on the performance-increasing distribution of write and read requests.
As already stated in the self-declaration of this report, it has been unable for me to implement the logic of the two patterns. Notwithstanding, theoretical functioning with its target factors is explained below more in detail.

3.1 Gatekeeper

The gatekeeper cloud design pattern is a software architecture pattern that is used to control access to the database server of our cloud environment. It involves the use of a gatekeeper component that sits between clients and the Proxy server. The gatekeeper is responsible for enforcing access control policies, authenticating clients, and managing the flow of requests to the underlying resources.
One of the main benefits of using the gatekeeper pattern is that it helps to centralize and standardize the management of access to the server. This can make it easier to implement security measures and to monitor access to the server. In our case, the Gatekeeper's firewall, unused services, ports, and even IPtable should be refined, tuned, and well cared. It can also help to reduce the complexity of the overall system by providing a single point of control for managing access.
When a client makes a request, the gatekeeper checks the request against the configured access control policies to determine whether the client is authorized to access the requested resource. If the client is authorized, the gatekeeper allows the request to proceed and forwards it to the appropriate resource. If the client is not authorized, the gatekeeper blocks the request and returns an error or appropriate response to the client.
Another important role of the gatekeeper is to manage the flow of requests to the underlying resources. This can involve routing requests to the appropriate resource, load balancing requests across multiple resources, or performing other types of request management tasks. The gatekeeper may also be responsible for monitoring the health and availability of the underlying resources and redirecting requests to alternative resources if necessary.
Overall, the Gatekeeper cloud design pattern is a useful way to manage access to our Proxy server in a cloud environment, helping to improve security and reduce complexity. It allows organizations to control access to resources in a centralized and standardized way, making it easier to implement and maintain security measures and to monitor and track access to resources. It is used in conjunction with other security measures, such as firewalls, the monitoring of unused services and ports, and IPtables to provide a comprehensive security solution for cloud-based resources.

3.2 Proxy

Our MySQL proxy server is a software component that sits between the Gatekeeper and the MySQL master-slave server, acting as an intermediary for database requests. The proxy server receives requests from clients, processes them, and then forwards the requests either to the slave or the master server. Write requests are handled by the master and passed to the slaves, whereas read requests are directly routed to the slaves. In order to program our own customized proxy server, we make use of three different implementations to manage the write and read requests between the entities:

- Direct Hit: Incoming write requests are directly forwarded to the MySQL master node and there is no prevalent logic to distribute the data.
- Random: The Proxy server will randomly choose a slave node on MySQL cluster and forward the read request to it.
- Customized: Measures the ping time of all the slave servers and forwards the message to one with less response time.

There are several advantages why we use a MySQL proxy server within the cloud architecture. One of the main reasons is to improve performance and scalability. The proxy server reduces the load on the underlying servers and improve response times for clients.
Overall, a MySQL proxy server is a useful tool for managing access to a MySQL database in a cloud environment. It can help to improve performance, scalability, and security, as well as providing other types of functionality such as monitoring, logging, and data manipulation

4 Conclusion

4.1 Result summary

To put it in a nutshell, the benchmarking time of the Sakila database is exemplarily shown for a standalone and a master-slave MySQL server in read-write mode. From the outcomes, it can be concluded that the performance of a standalone MySQL server and a master-slave MySQL server configuration can vary depending on the workload and the specific requirements of the application. In our sysbench's read-write case a cluster deployment is prefered. In general, a standalone MySQL server may be more suitable for applications with a high volume of writes and a low volume of reads, while a master-slave MySQL server configuration might be more suitable for applications with a high volume of reads and a lower volume of writes. In addition, the theoretical functioning of a Gatekeeper and Proxy cloud design pattern contributes to the development of an advanced database cloud architecture statement, targeting the improvement of influencing like scalability, security, and performance.

4.2 Code instructions

This section provides a step-by-step instruction to run the given implementation with ease.

1. Download the AWS Command Line Interface and install it
2. Learner Lap ≫ AWS Details ≫ AWS CLI Show
3. Execute `aws configure` in the command line
4. Copy and paste the suiting credentials from step 2 and *region=us-east-1; output=json*
5. Unzip the compressed project file to your location of wish
6. Learner Lap ≫ AWS Details ≫ Download PEM
7. Navigate with your command line to the location where you stored the project folder
8. The script requires you to pip install `boto3` via the command line. Ignore if already done.
9. Run the Python script with `python3 script.py` in your command line
10. Follow the instructions given in chapter 2.2