LOG8415: Advanced Concepts of Cloud Computing

Final Project

Scaling Databases and Implementing Cloud Design Patterns

By: Gervais Presley Koyaweda (Matricule: 2305686)

Presented to:
Vahid Majdinasab
Fall 2024

Contents

Chapter 1

Introduction

With the rise of distributed systems and cloud infrastructures, optimizing database management has become essential to ensure high performance in complex environments. This project explores the implementation of three SQL query routing strategies—Direct, Random, and Customized—in a distributed cluster environment deployed on AWS EC2.

The objective is to evaluate the impact of these strategies on load balancing, latency, and the reliability of SQL operations under real-world conditions using the Sakila database. Each stage of the deployment and benchmarking has been carefully documented, with key outputs stored in dedicated files for reproducibility.

Chapter 2

Report

2.1 1. Benchmarking MySQL with Sysbench

To evaluate the baseline performance of MySQL, Sysbench was used to simulate a variety of workloads. This tool provides insights into the database's capacity to handle concurrent read and write operations. The initial benchmarking helped identify potential bottlenecks and set performance expectations for subsequent cluster configurations.

2.2 2. Implementation of The Proxy Pattern

The Proxy pattern was implemented using a dedicated EC2 instance. The proxy manages query routing based on three strategies:

- **Direct:** Routes all queries to the manager instance.
- **Random:** Distributes queries randomly among the worker instances.
- **Customized:** Selects the fastest worker based on response times.

The proxy ensures efficient query distribution while abstracting the underlying cluster complexity from the client.

2.3 3. Implementation of The Gatekeeper Pattern

The Gatekeeper pattern acts as a security layer to validate incoming SQL queries. Implemented as an EC2 instance, it ensures that only authorized queries reach the proxy. This validation prevents harmful or unauthorized operations, adding an extra layer of security to the cluster.

2.4 4. Benchmarking the Clusters

The performance of the cluster was evaluated using the `benchmarking_requests.py` script. This script executed 1000 SQL queries for each routing strategy, with a mix of read (SE-

LECT) and write (INSERT) operations. The results, recorded in `benchmark_results.txt`, are summarized in the table below:

Strategy	**Operation Type**	**Success (%)**	**Total Time (s)**	**Avg Time (s)**
Direct	Read	100	148.40	0.1484
Direct	Write	100	132.00	0.1320
Random	Read	100	129.68	0.1297
Random	Write	100	159.22	0.1592
Customized	Read	100	161.34	0.1613
Customized	Write	100	138.49	0.1385

Table 2.1: Benchmark Results for Routing Strategies

Interpretation of Results

The results highlight the trade-offs between the strategies:

- **Direct:** Provides the best performance for write-intensive workloads due to its centralized routing.
- **Random:** Achieves the lowest average read time, making it ideal for evenly distributed read-heavy environments.
- **Customized:** While introducing some overhead for response time calculations, it performs well in heterogeneous environments where worker performance varies.

2.5 5. Implementation Description

The implementation involved the following steps:

1. **VPC Setup:** A secure network environment was created using the script `get_vpc.py`.
2. **Subnet Configuration:** Subnets were retrieved and configured using `get_subnet_id.py`.
3. **Security Group Creation:** Security rules were implemented with `create_security_group.py`.
4. **EC2 Instance Launch:** Instances for Manager, Workers, Proxy, Gatekeeper, and Trusted Host were launched using `create_instances.py`.
5. **Cluster Configuration:** MySQL services were configured with `setup_manager_and_workers.py`.
6. **Service Deployment:** Proxy, Gatekeeper, and Trusted Host were set up using `setup_cluster_2.py`.

2.6 6. Summary of Results and Instructions to Run the Code

The project demonstrated the effectiveness of various routing strategies in a distributed cluster environment. Each strategy offers unique advantages depending on workload characteristics. The provided scripts and configuration files allow for easy replication of the setup.

To run the project:

1. Execute the scripts in the following order:
    - `get_vpc.py`
    - `get_subnet_id.py`
    - `create_security_group.py`
    - `create_instances.py`
    - `setup_manager_and_workers.py`
    - `setup_cluster_2.py`
    - `benchmarking_requests.py`
2. Review the generated log files and benchmark results for insights.