ohary Rakotomalala 2028622

LOG8415E
Scaling Databases and Implementing Cloud Design Patterns

J

Polytechnique Montreal

December 28, 2023

Github: https://github.com/grolox69/LOG8415E-TPS

1 Benchmarking MySQL stand-alone vs. MySQL Cluster

For the MySQL stand-alone we have

2 Implementation of The Proxy pattern

We have proxy server built using Flask to facilitate communication with a MySQL Cluster. Its primary objective is to act as an intermediary, directing incoming SQL queries to various MySQL servers within the cluster. The script employs SSH tunneling, leveraging the paramiko and sshtunnel libraries for secure communication with the MySQL servers. Server details, including names, user names, public DNS names, IP addresses, and states, are stored in a CSV file, and the get$_server_info$function retrieves this information based on server names.

The Flask web application created by the script features a single endpoint (/endpoint) designed to handle both GET and POST requests. The gatekeeper$_request$ function processes incoming SQL queries, implementing three distinct routing strate
$direct hit, random hit,$ and customized hit. In the direct hit strategy, queries are sent directly to the MySQL master node. The rand

The script further establishes SSH tunnels to slave servers for query execution through the create$_ssh_tunnel$ function. Additionally,
$80, making it externally accessible. However, it's important to note that this script is designed for educational purposes, and in a pr

3 Implementation of The Gatekeeper pattern

For the gatekeeper we use Flask web framework to create a simple HTTP client that listens for requests on the /endpoint route. Upon receiving requests, the script extracts the chosen implementation and SQL query. It then validates the implementation, and if deemed valid, forwards the request to a predefined proxy server using the requests library. The chosen implementation and SQL query details are printed for logging purposes, and the content of the received response from the proxy server is returned to the client. The script runs as a standalone application on all available network interfaces, making it externally accessible on port 80. It serves as a straightforward client interface for interacting with a proxy server handling SQL queries.

4 Describe clearly how your implementation works

We used Fabric, a python SSH library that establishes connections to hosts to run a suit of commands. We deployed a flask app for the orchestrator on port 80 using Nginx and Gunicorn. The flask app running on the orchestrator handle

incoming request to "/new-requests"

When a new request is received, it spawns a new thread to handle the request using the handleRequest function.

In the handleRequest function, it checks for the availability of a free container using the getFreeContainer function. If a free container is available, it processes the request immediately by calling the processRequest function. This involves updating the status of the chosen container to "busy," sending a request to the container, updating the status back to "free," and returning the result. If no free container is available, it adds a placeholder (0) to the requestQueue and returns a message indicating that the request is queued.

5 Summary of results and instructions to run your code.

6 Sample of what our cluster returns for an incoming request

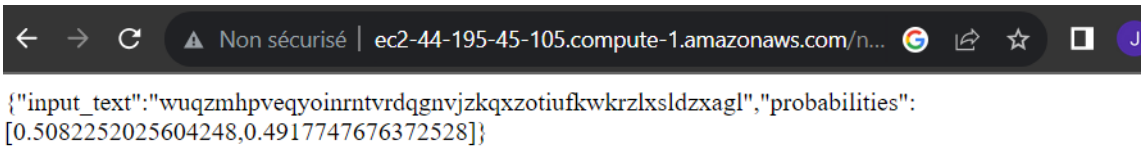

Figure 1: Cluster response for an incoming request

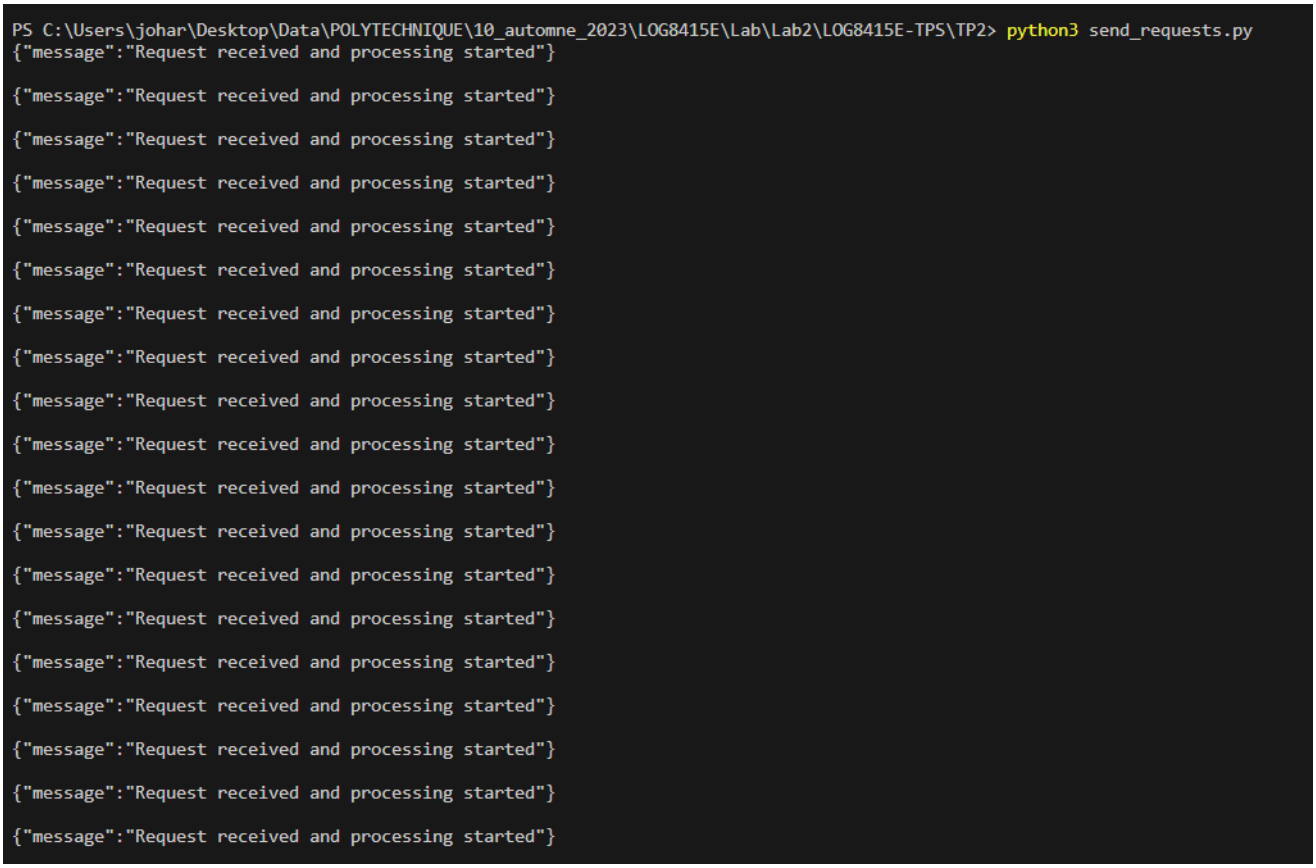

Figure 2: Output of send_requests.py

7 Summary of results and instructions to run our code

The instruction to run the code is on the read me