LOG8415
Advanced Concepts of Cloud Computing
Session Project

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

1 Identification

Students name: Maximiliano Falicoff 2013658

Date of submission: Thursday December 28^{th} 2023

2 Benchmarks

2.1 Standalone

Number of threads	6
queries performed	295260
transactions	14763
queries	295260
ignored errors	0
reconnects	0
total time	60.0238s
total number of events	14763
min	9.78
avg	24.39
max	132.07
95th percentile	34.33
sum	360009.65
events (avg/stddev)	2460.5000/4.35
execution time (avg/stddev)	60.0016/0.00

Table 1: SQL statistics and General statistics

2.2 Cluster

Number of threads	6
queries performed	410717
transactions	20535
queries	295260
ignored errors	1
reconnects	0
total time	60.0192s
total number of events	20535
min	5.23
avg	17.53
max	100.34
95th percentile	31.37
sum	359997.91
events (avg/stddev)	3422.5000/9.98
execution time (avg/stddev)	59.9997/0.01

Table 2: SQL statistics and General statistics

2.3 Analysis

The cluster outperformed the standalone SQL instance in almost every metric, especially in throughput and efficiency, where it handles a significantly higher number of queries and transactions. It also shows an advantage in response time, with lower average, minimum, and maximum latencies, and better performance under load.

For scenarios requiring high throughput and efficient handling of large volumes of queries and transactions, the cluster would be the preferred choice. Its ability to maintain lower latencies under load also makes it suitable for applications where response time is critical.

3 Proxy Implementation

When a request arrives to the proxy, it carries a raw SQL query. For monitoring purposes, we log the request's details, including its type and origin. The core of the process lies in how the application handles this SQL query. If the request does not specify a method type and is a read operation (like a 'SELECT' query), the application selects the best node to execute the query. This selection is based on the current load of the available nodes, ensuring efficient distribution of read requests. In order to check the lowest load, we iterate through all instance id's and check the cpu load and pick the lowest one.

In cases where the request specifies a method type or if the query is a write operation (anything other than 'SELECT' or 'SHOW'), the application routes it differently. For example, a write operation is always directed to the manager node for execution, ensuring data consistency and integrity. We support the following methods:

- direct: connect directly to the manager ip, like a write query
- random: we select a random node from the manager to all the workers
- custom: get get the ping from each worker and select the lowest one

The application uses SSH tunnels for secure communication with the database nodes. Once the appropriate node is selected, it establishes a connection, executes the query, and fetches the results. These results are then returned to the requester. The application also logs the outcome of the request, which includes details about the query execution and its results.

4 Gatekeeper Implementation

The gatekeeper pattern has two nodes. The first node acts as our API endpoint where we define some routes that are publicly available, in our case we defined the following routes:

- /film Returns all the films
- /movie/<id> Returns movie with the specified id
- /actors Returns all the actors
- /actor/<id> Returns actors with specified id
- /customer/<customerId>/rentals Returns list of rentals for a specific customer
- /raw Send a raw SQL query

Each request can have a additional query parameter being the method type: custom, random, direct, but these are only valid for read queries and will be ignored if it's a write query since it be forwarded to the master.
These endpoints are then converted to raw SQL queries and are forwarded to our trusted host. In our security groups, the trusted host can only be accessed by a traffic source originating from the gatekeeper, so.a client would not be able to contact the trusted host directly.

The trusted host acts as a filter for malicious or faulty queries, we defined a method where we check if the query respects our rules. We have defined the following rules:

- Define a list of allowed tables for `INSERT`, `UPDATE`, and `DELETE` operations. These tables include: `actor`, `address`, `category`, `city`, `country`, `customer`, `film`, `film_actor`, `film_category`, `film_text`, `inventory`, `language`, `payment`, `rental`, `staff`, and `store`.
- Check for the presence of forbidden keywords in the query, excluding `INSERT`, `UPDATE`, and `DELETE`. Forbidden keywords include `drop`, `exec`, `execute`, `create`, `alter`, `grant`, `revoke`, and `truncate`. If any of these are present, the query is considered unsafe.
- If the operation is `INSERT`, `UPDATE`, or `DELETE`, check if it is performed on an allowed table. Use a regular expression to extract the table name and verify if it's in the list of allowed tables.

- Perform a basic check for SQL injection patterns. Look for patterns like `--`, `;`, `/`, `\bOR\b`, and `\bAND\b`. If any of these patterns are found, the query is considered unsafe.
- If none of the above conditions are met, the query is considered safe.

If the query satisfies all the rules, it will then be forwarded to the proxy.

In order to deploy these nodes, I have defined a GitHub workflow so that when changes are made to their respectful directories, a docker image will be created and published to a docker registry. In our ansible playbook, we install docker and start the service and pull the image from the repository and start the image.

5 Overall Implementation

In terms of deploying the entire stack the flows is as follows:

The docker images for the proxy, trusted host and gatekeeper are automatically built by the github CI pipeline and pushed to a registry.

Our terraform configuration will deploy our ec2 instances with their respective security groups so that trust boundaries are set, for example myself from my local computer would not be able to contact anything except the gatekeeper. Our terraform configuration will also generate all .env files needed for our docker images, we also generate a ssh key to be able to connect to our machines and aansible inventory file that has all ip's of our machine that we need to configure.

We then run our ansible playbook for our cluster manager first since I encountered error when deploying th e workers at the same time. Our manager will be setup using ansible. Once the manager is setup, we can parallelize the setup of the workers, the proxy , the trusted host and the gatekeeper to speed up the deployment time. In our ansible playbooks for the workers, we setup the mysql cluster management server Dameon pointing to our master's IP. For our docker based systems, ansible will install docker mount the env perviously generated and start the containers.

All of this can be done with a single command with no further intervention (see below for instructions)

6 Summary

7 How to deploy

7.1 Requirements

In order to deploy the project you need the following packages to be installed and make sure they are added to the PATH environment variable:

- terraform
- aws-cli
- ansible

- make

You then need to populate your aws_credentials with the correct credentials.

7.2 Scripts available

Depending on what you want to deploy, there are a few commands available there is a makefile with all the commands available, here are the important ones:

- *make cluster-deploy* will deploy everything, meaning the gatekeeper, trusted host, the proxy and the mysql cluster (manager + worker), it will then configure them using ansible, this may take a bit of time due to the fact that the manager must be deployed before the workers. Afterwards everything should be deployed.
- *make benchmark* will deploy the standalone instance benchmark it, destroy the instance, then do the same for the cluster, during the after each benchmark the results are copied to a local folder.

Afterwards, there is a smaple python script under */src/apps/client* where there are some predefined queries to run, you should install the dependancies under the rot folder of the application: *pip install -r requirements.txt*

8 Video Link

https://youtu.be/XOmWRtt2JNc