COMPSCI 677 Lab 3 - Spring 2023

Design Document

By
Swati Agrawal

Medha Goel

Overview

The Stock Bazaar application is a distributed system that consists of three main services: the Catalogue service, the Order service, and the Front-end service. The Catalogue service stores information about available stocks, and the Order service manages the buy and sell orders. The Front-end service acts as an API gateway that handles user requests and communicates with the other services.

To achieve crash failure tolerance, the system replicates the Order service into multiple replicas. The Front-end service is designed to detect crashes of the leader replica and re-elect a new leader from the available replicas. When a crashed replica comes back online, it synchronizes with the other replicas to retrieve the missed orders.

Architecture

Catalogue service

The Catalogue service is a simple key-value store that stores information about available stocks. It exposes a RESTful API that allows the Front-end service to retrieve information about stocks.

Order service

The Order service consists of three replicas that are started with unique ID numbers and their own database files. The replicas communicate with each other using a replication protocol that ensures consistency between the replicas. When a new trade order is received by the leader replica, it propagates the order to the other replicas to maintain data consistency. The order service exposes a RESTful API that allows the Front-end service to place buy and sell orders, as well as retrieve information about orders.

Front-end service

The Front-end service is responsible for handling user requests and communicating with the other services. When a user requests information about a stock, the Front-end service first checks its local cache to see if the information is available. If not, it forwards the request to the Catalogue service and stores the result in the cache for future use. When a user places a buy or sell order, the Front-end service forwards the request to the leader replica of the Order service.

To ensure crash failure tolerance, the Front-end service periodically checks the availability of the leader replica. If the leader is unresponsive, the Front-end service re-elects a new leader from the available replicas. When a crashed replica comes back online, the Front-end service synchronizes the missed orders with the replica.

Replication protocol

The replication protocol used in the Order service is a simple log-based replication protocol. Each replica maintains its own local database file that stores the orders received by the replica. When a new trade order is received by the leader replica, it appends the order to its local database file and propagates the order to the other replicas.

When a crashed replica comes back online, it retrieves the latest order number from its local database file and asks the other replicas for the missed orders since that order number. The other replicas respond with the missed orders, which the crashed replica appends to its local database file to catch up with the other replicas.

Testing and Evaluation with Deployment on AWS

For testing and evaluation, we first developed and ran unit tests for each microservice to ensure that they are functioning correctly in isolation. Then, we performed integration testing to ensure that the microservices are working together as expected.

Next, we performed load testing to evaluate the system's performance under heavy load. This involved simulating a large number of concurrent users and measuring the response time and throughput of the system. We also performed stress testing to evaluate how the system handles resource exhaustion and other failure scenarios.

For deployment on AWS, we first containerized each microservice using Docker and deployed them to Amazon Elastic Container Service (ECS) or Kubernetes. We then set up a load balancer to distribute traffic among the different replicas of the order service. We also set up monitoring and alerting to detect and respond to issues in real-time. Finally, we performed end-to-end testing to ensure that the deployed system is functioning correctly in a production-like environment.

Conclusion

The Stock Bazaar application with crash failure tolerance is designed to ensure that the system remains available and consistent in the face of crashes. The replication protocol used in the Order service ensures that the data remains consistent across replicas, while the Front-end service's leader election algorithm ensures that the system remains available even if the leader replica crashes. When a crashed replica comes back online, it synchronizes with the other replicas to ensure that it has the latest data.