Movie Review System

Rajat Sharma
40196467

Amanpreet Singh
40221947

Simran Kaur
40221666

Ashish Upadhyay
40225754

1 ABSTRACT

A movie review system is a tool that allows users to search for and read reviews of movies. The system typically includes a database of movie information and reviews, as well as a user interface that allows users to search for and view reviews. The system may also allow users to write and submit their own reviews and may include features such as rating scales or the ability to filter reviews by certain criteria (e.g. genre, release date). The goal of a movie review system implementing distributed system properties is to provide users with a convenient way to access a wide range of reviews and opinions about movies, helping them to make informed decisions about which movies to watch, in an easy fast, and reliable way. All our group members are movie and TV show enthusiasts. We love to watch good movies together and review them later. As computer science students, this allowed us to implement our hobby into a real-world product. Our goal is to make implement distributed system properties such as transparency, fault tolerance, scalability, and concurrency through this movie review system and make it fitter for a real-world use case.

2 INTRODUCTION

Our project uses Flask as the primary backend framework. Flask can be used in a distributed system by deploying multiple application instances across multiple servers. This can be done using a load balancer such as Kubernetes, which distributes incoming requests evenly among the available servers. To set up a distributed system with Flask, we have followed the following steps: 1. Set up the servers: We have used an AWS instance for creating the primary server.

2. Deploy the application: Deploy the Flask application to the server manually. It can also be done using a deployment tool such as Ansible, Fabric, or Docker.

3. Configure the load balancer: We have used Kubernetes for automatic load balancing.

4. Set up a database: We have used Cassandra as our fault-tolerant database with no single point of failure.

Apache Cassandra is a distributed database management system that is designed to handle large amounts of data across many commodity servers, providing high availability and horizontal scaling. Cassandra is particularly well-suited for use in distributed systems because it is designed to be fault-tolerant and able to handle data replication across multiple nodes.

In a distributed system, Cassandra can be used to store and manage large amounts of data that need to be distributed across multiple nodes. This can be useful in a variety of scenarios, such as:

1. Storing and processing large amounts of data from multiple sources: Cassandra's distributed architecture makes it easy to ingest and process data from multiple sources in real time.

2. Building scalable, high-performance applications: Cassandra's ability to scale horizontally means it can handle large amounts of data and high levels of concurrency without sacrificing performance.

3. Enabling high availability: Cassandra is designed to be highly available and can automatically handle the failure of nodes in the event of a hardware or software failure.

Overall, Cassandra is a powerful tool for building distributed systems that need to handle large amounts of data and provide high availability and scalability.

Kubernetes is an open-source container orchestration platform that is designed to manage the deployment and scaling of containerized applications in a distributed environment. It is often used in distributed systems to provide automation, scaling, and high availability for applications.

In a distributed system, Kubernetes can be used to manage the deployment, scaling, and orchestration of containerized applications across multiple servers. It provides a range of features that make it easy to deploy and manage applications in a distributed environment, including:

1. Automatic scaling: Kubernetes can automatically scale the number of replicas of a containerized application up or down based on demand, ensuring that the application has the resources it needs to function correctly.

2. Self-healing: Kubernetes can automatically detect and replace failed containers, ensuring that the application remains available even in the event of a hardware or software failure.

3. Load balancing: Kubernetes can automatically distribute traffic evenly among the available replicas of an application, helping to ensure that the application can handle large amounts of traffic.

3. Deployment and rollback: Kubernetes provides tools for deploying and rolling back applications, making it easy to

manage the lifecycle of applications in a distributed environment.

Overall, Kubernetes is a powerful tool for managing the deployment and scaling of containerized applications in a distributed system. It can help to automate many of the tasks involved in managing a distributed system, making it easier to build and maintain distributed applications.

3 SYSTEM

The following is the system design of our movie review system: Cassandra database: The Cassandra database would be used to store movie information and user reviews.

1. Flask web application: The Flask web application would provide the user interface for the movie review system. It would allow users to search for movies, view movie details and reviews, and submit their own reviews. The Flask application would communicate with the Cassandra database to retrieve and store data.

2. Kubernetes cluster: The Kubernetes cluster would be used to deploy and manage the Flask web application. It would provide features such as automatic scaling, load balancing, and self-healing to ensure that the application is highly available and can handle large amounts of traffic.

3. Load balancer: Kubernetes includes a built-in load balancer that can is used to distribute incoming traffic evenly among the available replicas of our service. When you create a service in Kubernetes, you can specify the number of replicas you want to run and the load balancer will automatically distribute traffic among them.

4. Monitoring and alerting: Azure monitor is used to monitoring the performance and availability of the movie review system. An alerting system, such as Alertmanager, could be used to send notifications in the event of any issues or outages.

5. The dataset: We have used the IMDb Review Dataset – ebD from Kaggle (7.78GB). Our system stores data and fetch results according to this dataset. The dataset has:

1. Total records = 5, 571, 499
2. Total shows = 453, 528
3. Users = 1, 699, 310

This is just one possible design for a movie review system using Cassandra, Kubernetes, and Flask. There are many other ways that these technologies could be combined and configured to meet the system's specific needs.

4 DEMO SCENARIO

We create three virtual machines on the Azure Cloud Platform and set up CassandraDB on each. We then link these "nodes". We have hosted our flask application on one of these nodes and used a script to insert data from JSON files to this Cassandra database. The Flask app is then connected to Cassandra. We have a front with a home page where users can search for movie reviews. The backend route displays the top 8 movie reviews, renders the result, and shows it on the front page. We demonstrate the following distributed properties in our demo:

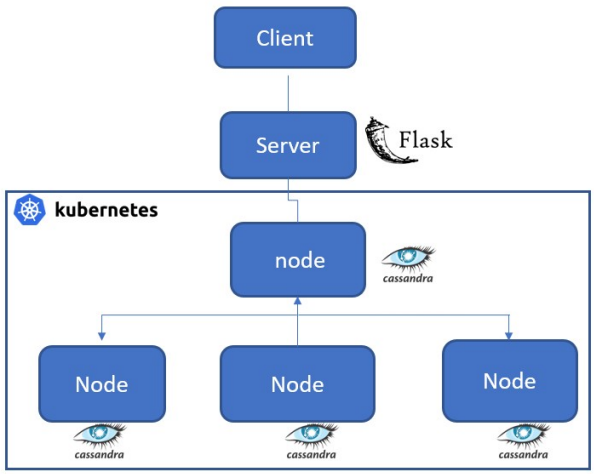

Figure 1: A visual depiction of the system design of our Movie Review System

1. Scalability: We demonstrate scalability by using Kubernetes. Currently, we have created virtual machines and we have created one of them as a master and the other as a worker. However, with more resources, we can automate the entire process and make the entire system scalable.

2. Fault tolerance: We demonstrate fault tolerance by first running a query on our application. We then shut down one node and run the same query again. If we get the same result as our first query, it shows that even with one node down, the system works as expected.

3. Transparency: We demonstrate that our system comprises multiple VMs working in synchronization to provide the expected result. Then we run a query and demonstrate that there's no visibility of the internal working of the processing by the system. Thus the whole system appears to be a single entity rather than individual components.

4. Data Replication: Our Cassandra nodes have a replication factor of 3 and the data stays available as long as at least one node is up and running.

5 CONCLUSION

In conclusion, a movie review system that is built using Cassandra, Kubernetes, and Flask can provide a scalable, highly available platform for storing and accessing movie information and reviews.

The Cassandra database can store large amounts of data and support high levels of concurrency, making it well-suited for storing movie information and user reviews. The Flask

web application can provide a user-friendly interface for searching for movies, viewing movie details and reviews, and submitting new reviews. The Kubernetes cluster can manage the deployment and scaling of the Flask web application, providing features such as automatic scaling, load balancing, and self-healing to ensure that the system can handle large amounts of traffic and remain highly available.

Overall, this distributed system can provide a convenient and reliable platform for accessing movie information and reviews, helping users to make informed decisions about which movies to watch.

A distributed system for movie review analysis could be used to recommend films to users based on their preferences and past viewing history. This could help companies create personalized experiences for their customers, and drive engagement and loyalty. Companies in the entertainment industry could use a distributed system for movie review analysis to gain insights into the performance and reception of their films. This could help them make data-driven decisions about marketing and distribution strategies. Movie review data could also be used to understand consumer preferences and behavior. This could be useful for companies looking to optimize their product or service offerings.

6 REFERENCES

[1] Cassandra Documentation:
https://cassandra.apache.org/doc/latest/
[2] Flask Documentation:
https://flask.palletsprojects.com/en/2.2.x/
[3] Kubernetes Documentation:
https://kubernetes.io/docs/home/
[4] Azure Documentation:
https://learn.microsoft.com/en-us/azure/