CSCI 5253 Datacenter Scale Computing

Cloud Based Movie Recommender System

Amit Baran Roy
Aparajita Singh

Contents

1. Project Overview
2. Project Goals
3. Components of The Project - Purpose and Advantages and Disadvantages
4. Architecture and Interaction of components
5. Capabilities
6. Limitations and Future Work
7. References

Project Overview

The aim of our project is to build a cloud based recommendation system for recommending movies. A recommender system is used to recommend items to a user based on the judgement whether they would prefer the item or not. This is done by predicting the future preferences of the user on the basis of past preferences and the preferences of similar users.
Recommendation systems are used by companies such as Amazon and Netflix to recommend new products and movies to their users. These recommendations help save time for these companies in discovering the best products, movies etc to display in front of the user. They are also helpful in increasing the possibility that more products are viewed or bought by customers. From the customer's perspective, the recommendations give them a personalized experience, which increases engagement and delight for the application. The number of options available on the internet are overwhelming in the current time, therefore recommender systems help provide relevant options to the customers. They benefit the service provider by increasing engagement and potential revenue.
The goal of our project is to create such a recommender system for movies using collaborative filtering approach. We have used the MovieLens dataset. The dataset contains 25 million user ratings for 62000+ movies. For an interactive experience for online users, we have used Streamlit. To make our application scalable, it is hosted as a Docker container on GCP and deployed to Kubernetes cluster using GKE.

Project Goals

Phase 1

1. Obtain movie ratings dataset from MovieLens
2. Perform data cleaning and data filtering using Pandas library

Phase 2

1. Write a program to fetch URL of the movie poster from IMDB and save the URL in a new column in the database
2. Write code in python for providing recommendations for a user based on previous preferences of the user and preferences of other users - use collaborative filtering

Phase 3

1. Create signup system for users (using Streamlit)
2. Create login system for users (using Streamlit)
3. Create logout system for users (using Streamlit)
4. Save movie ratings to redis database
5. Fetch data from redis database using REST
6. Create rating movies functionality for each logged in user
7. Create functionality for user to view previously rated movies
8. Save intermediate and final recommendations in redis database
9. Create “Show More” functionality in the system so that the user can view the next set of recommendations or movies
10. Make it run in local system

Phase 4

1. Scale up the application by deploying on Docker and Kubernetes

Phase 5

1. Perform testing to ensure that the system is working
2. Obtain recommendations from the working system

Components of The Project

(A) Language: Python

Python is easier to use than other languages. It is also the language that is supported by Streamlit framework. Python has good support for other components of our project as well. Python has vast support and options for libraries. It is a good language for data science projects such as ours.

A disadvantage of using Python is its speed. It is generally slower than other languages.

(B) Libraries: Pandas, Pickle/JsonPickle, HashLib

Python library pandas is widely used for data cleaning, analysis, manipulation and visualization. It was used in our project for easy data manipulation operations such as merging and selecting.

Pickle/JsonPickle helped us to store dataframes or other results easily as objects in Redis. This is also used for data transfer across REST calls.

Hashlib makes it easy to create a hash and save passwords for all users.

(C) Algorithm: Collaborative filtering

There are mainly two types of recommendation systems: content based and collaborative filtering. We have used collaborative filtering in our approach. Collaborative filtering techniques find similar groups of users and provide recommendations based on similar tastes within that group. It is based on the assumption that a user might be interested in what similar users are interested in. In collaborative filtering, there are two approaches: first is user based and the second is item based. User based is based on user similarity and item based is based on similarity among items. In our project, we have used user based collaborative filtering. We can find similarity between users through distance and similarity measurements such as euclidean distance, pearson correlation, cosine similarity, etc. We have used Pearson correlation in our project. An advantage of Pearson correlation is that it is invariant to scaling.

the person who is logged in the system is the active user. The first step is to obtain some movie ratings from the active user. Then, based on these ratings, the next step is to find the top X neighbours. This is done by filtering out users that share the most movies in common with the active user. The next two steps are calculating the similarity score and generating recommendations. Pearson correlation is used to find similarity between the top neighbours and the active user. We have used top 50 similar users to the active user. For generating recommendations, the first step is to calculate the weighted average of the ratings of the movies using Pearson correlation as the weight. This is done by multiplying the movie rating by the similarity index and then dividing by the result by the sum of weights. The top recommendations are the sorted list of this weighted average. To make the recommendations more relevant, we

have also taken into consideration the number of users that have rated the particular movie highly, that is a rating above 4. The previous recommendations found are rearranged on the basis of the movie popularity and then displayed to the active user.

Advantages of our approach:
Collaborative filtering takes user's ratings and similar user's ratings into consideration. Our approach also focuses on recommending movies that are rated highly by more number of users. An advantage of Pearson correlation is that it is invariant to scaling. Collaborative filtering is capable of adapting to the user's interests and ratings which may change over time.

Disadvantages of our approach:
As in any recommendation system, the algorithm requires some data from the user before it can start computing recommendations. Also, it requires at least some other users to have ratings for the same movies that the active user has already rated/watched. Therefore, proper data is required before recommendations are computed.

(D) Google Cloud Platform

Google Cloud Platform was selected as the platform for deploying our datacenter application. We are more familiar with GCP than AWS while AWS would have had a higher learning curve. GCP is also more cost effective than AWS. We can also use the Google provided student credits for using the platform free of cost. The UI of GCP is also easy to navigate. In addition, GCP has quite extensive documentation available.

(E) Google Cloud Storage

Google Cloud Storage has advantages such as security and disaster recovery. It is also cost effective and accessible. We used Google cloud storage to store datasets as well as docker containers are uploaded to google container registry that uses Google Cloud storage buckets.

(F) Docker Containers

Docker containers are advantageous over virtual machines as they are lightweight. Containerized applications virtualize the operating system and not the hardware. Containers are faster to create and are easily accessible from cloud to any machine.

(G) Google Kubernetes Engine

Kubernetes provides a system for automating deployment, scaling and management of containerized applications. Kubernetes helps scaling up applications by giving the ability to run containers across multiple nodes. GKE also provided the load balancer service that helps our application to go public quickly using public IP and handle multiple server requests.

Disadvantage of GKE can be it is bit costly than may be using other public facing deployment options like serverless.

(H) Redis Database

Redis is used as our database. We initially saved user ratings in redis database. Then, when we compute recommendations, the intermediate steps and the user personalized recommendations are also saved in the database. Redis is also used for saving login information. A disadvantage of Redis is the master slave architecture. The database crashes if the master node fails and we lose data.

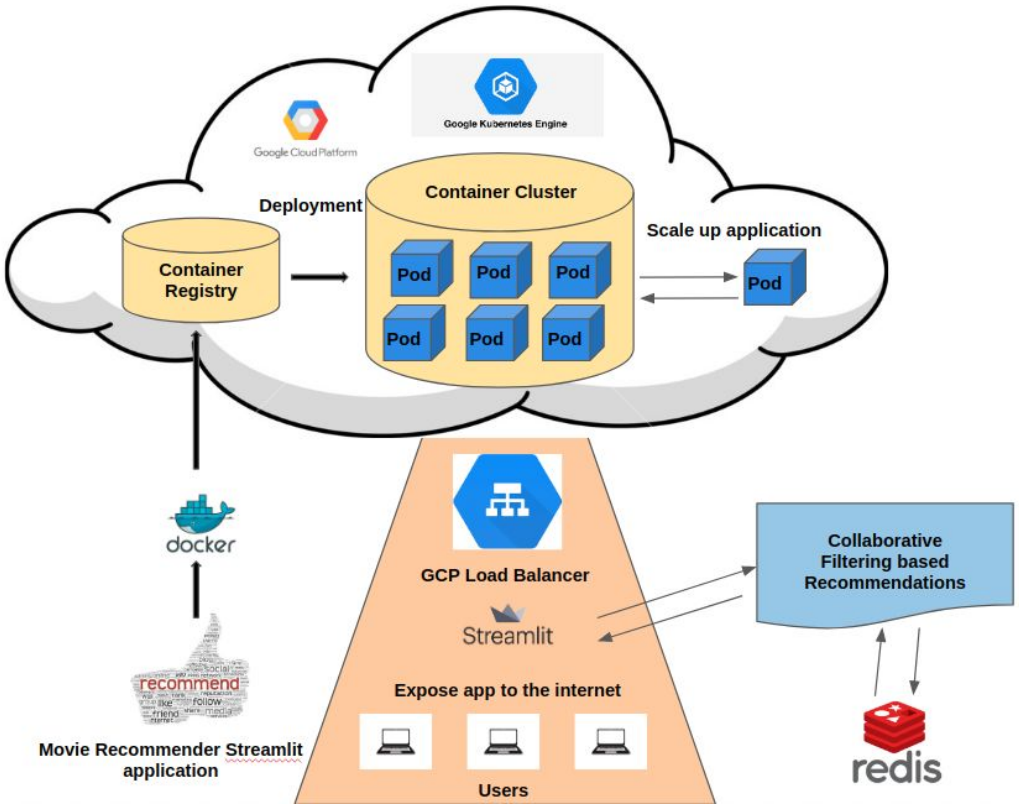
(I) REST

REST is easier to write and understand. REST is used for fetching data and recommendations from the backend and for saving data in the database. We have chosen REST over RabbitMQ as we had to make synchronous calls to the backend worker.

(J) Streamlit

Streamlit is an app framework specifically designed for machine learning and data science projects. It is free, open source and easy to learn. It is compatible with Python libraries such as pandas and numpy. It provides a rich user interface for our project.
It does not have functionalities like its alternatives Flask and Django. However, for our specific task, it was appropriate. Since it is new in development, it lacks advanced session handling and caching techniques. Another disadvantage is the app reruns every time an action is performed which makes the application slower.

Architecture and Interaction of Components



The user interactive Streamlit application is written in Python and it communicates with the collaborative filtering based recommender application hosted on a Flask server using synchronous REST calls. We used Flask along with REST because we wanted our application to be synchronous instead of using message queuing with RabbitMQ.

The Streamlit application and the flask application running the recommender algorithm are then built into two separate docker images and pushed as docker containers on Google Container Registry of GCP. The two docker containers along with another Redis container are then deployed on a Kubernetes cluster using GKE.

We created service configurations for all the workloads to generate the external IPs so that they can communicate within themselves.

We have used Redis extensively to store user personalized recommendations and many other interim results.

A kubernetes Ingress service helps to expose the application to external users by providing an external IP. It also acts as a load balancer service on GKE to handle multiple server requests.

When the cluster receives a request, the load balancer routes it to one of the Pods in the Service, which then returns an instance of the Streamlit app to the user.

The app has a signup and login feature so that every user can have their personalized recommendations. We have used the hashlib python library to store the hash of passwords in Redis.

Users have to select atleast 5 genres and then rate at least 5 movies after which the collaborative filtering recommendation algorithm is executed by the backend worker after receiving a REST request.

The application can be scaled up by adding additional pods to the cluster. We can also delete the service and container cluster to avoid incurring unwanted charges. The computed recommendations for an user are updated in Redis database as well as users can see the rated movies in the app dashboard.

Evaluation:- We have used a match percent factor for evaluation. It means that the top recommended movies are seen by the majority of the similar users and have received high ratings. This makes the recommendations high quality.

Working System Video:- **https://www.youtube.com/watch?v=cyC2jlzdmn4**

Github Code:-
https://github.com/AmitProspeed/Cloud-Based-Movie-Recommender-System

Capabilities

- Our algorithm is able to provide recommendations for an active user after obtaining at least 5 ratings from the user. This is required so that the collaborative filtering approach can have some data before it computes similar users and provides recommendations.

- Our front end can handle signup functionality. It allows a user to sign up using username and password.

- Our front end can handle login functionality. The user can login using the same credentials they used for signing up.

- Our front end can handle logout functionality.

- Our system is capable of saving new ratings for the active user to the redis database.

- Our system is capable of utilizing REST to fetch from and save data to the redis database.

- Our system is capable of utilizing kubernetes ingress service because of which it is available to external users by providing external IP.

- When the active user is rating movies, our system is capable of showing 24 movies at a time. The user can also click on the “show more” button to view the next 24 movies.

- When the active user is viewing recommendations, our system is capable of showing 24 recommendations at a time. The user can also click on the “show more” button to view the next 24 recommendations.

- The user can also view the movies they have previously rated.

- When the active user is viewing recommendations, our system shows the match percentage for indicating how much the user might like the particular movie.

Limitations and Future Work

- Currently, our system is not adding the current user's rating to the original dataset so that it can be computed for recommendations for another active user. The original dataset remains static. In the future, we would like to compute recommendations in an online setting, where the original dataset also keeps updating with new ratings from the signed up users.

- One of the limitations of our system is it is bottlenecked by Redis single point of failure. The service stops if the Redis master pod fails. Also, too many pods interfere with user sessions. We have to address these issues as future work.

- We would like to perform more extensive testing techniques like A/B testing to evaluate the recommendations provided by our algorithm. We would need feedback or surveys from different kinds of users for the recommendations.

- At the moment, our system displays a single list of movies as recommendations. While not a limitation, it is something that can be improved upon. We could display the movies by genre like it is done on platforms like Netflix.

References

1. https://blog.insightdatascience.com/building-a-scalable-online-product-recommender-with-keras-docker-gcp-and-gke-52a5ab2c7688
2. https://medium.com/swlh/how-to-build-simple-recommender-systems-in-python-647e5bcd78bd
3. https://www.kdnuggets.com/2016/02/nine-datasets-investigating-recommender-systems.html
4. https://medium.com/fnplus/evaluating-recommender-systems-with-python-code-ae0c370c90be
5. https://medium.com/@betz.mark/ten-tips-for-debugging-docker-containers-cde4da841a1d
6. https://kubernetes.io/docs/tasks/debug-application-cluster/debug-running-pod/
7. https://grouplens.org/datasets/movielens/