Realtime Movie Recommendation System

Final Project for: **CMPT 732@ SFU**
Instructor: **Greg Baker**

Student: **Mohammad Mazraeh**
Student Id: **301376403**
Fall 2018

https://github.com/MohammadMazraeh/movielens-recommender

Table of Contents

Problem Definition

The goal of the project is to implement a real-time movie recommender system. User's historical rating for the movies is the main source of training recommender system. The overall architecture of the system should have the following properties:

1- **Scalable**: The technologies and the architecture of the project should be scalable for much bigger users.
2- **Fault tolerant**: If one server in each technology goes down, the whole recommendation service should keep working.
3- **Zero down time in upgrades**: When using newer trained models, we should not stop the running recommender system.

Methodology

In this project I've used Spark, Kafka, Cassandra, Elasticsearch and Kibana.
In the following sections I will describe why I've selected each technology and how it is being used in the architecture. In **Figure 1** an overview of the whole system has been depicted.

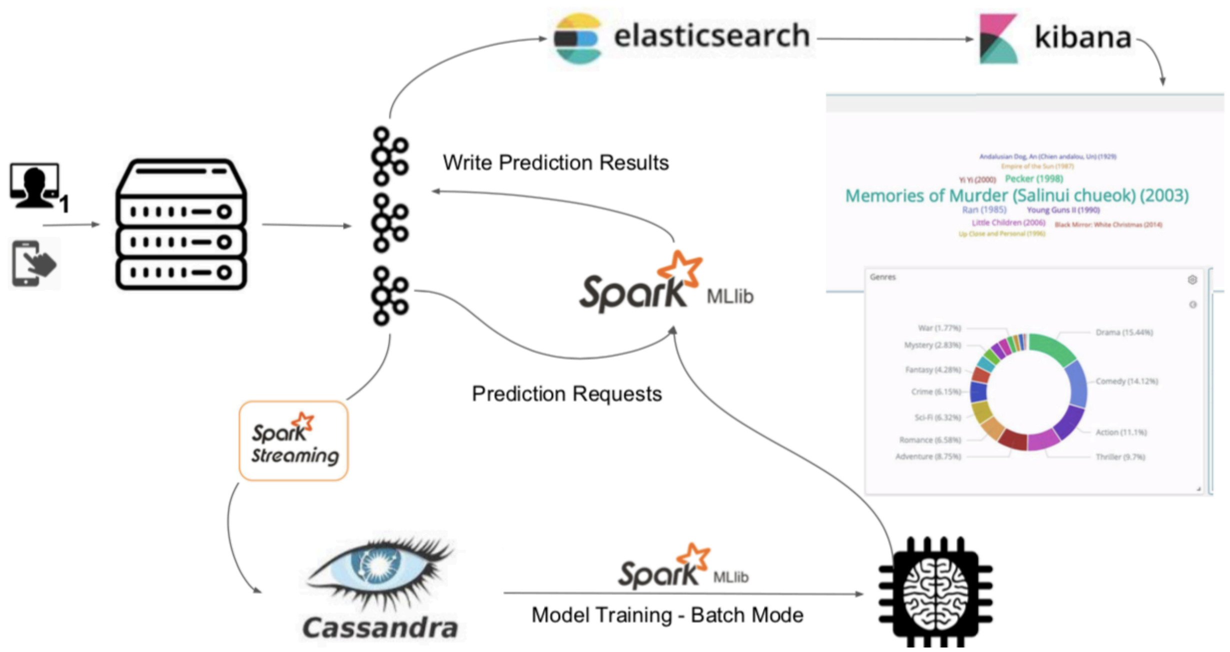


Figure 1 - Movie Recommender Architecture

Data Collection Layer

For data collection I'm just getting a static dataset and writing it into the Kafka. For this part the project requires a message queue with the following properties:

- **Scalable**: It should be easily scalable by just adding new nodes

- **Fault tolerant**: It should be able to handle the situation when some of the servers go down
- **Hard disk level data storing**: There is always a chance of loosing the connection between message queue and the consumers or may be the consumers don’t work that well. So we need a technology which stores the data in the hard disk with an specific retention policy.

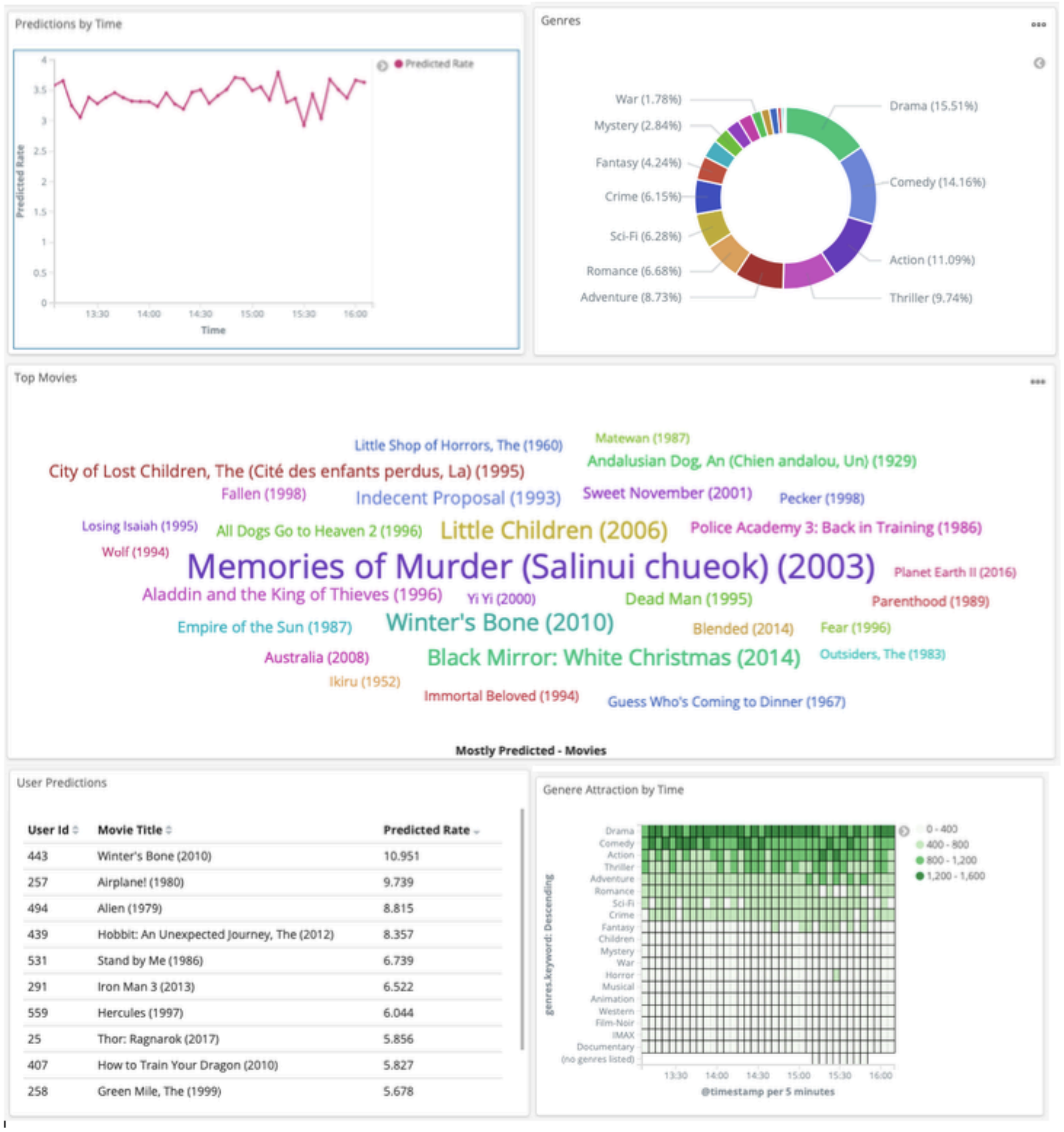
Kafka is a great message queue with this 3 features. Additionally it can feed multiple consumers for the same topic at the same time (complete set of data for that topic) and it is good specially when we can monitor our data in stream.
The only problem with Kafka is about it’s **delivering guarantee**. By default, it supports at least once guarantee and with some special cases it can be configured with exactly once delivery. However, this feature does not affect our application but may be important for other type of applications.

Ingestion Layer
The ingestion layer is responsible for gathering the user events and store them in a way that can be used later for training machine learning models or other possible applications. There is an data_loading/event_stream_generator.py which reads the data from static data base, changes some field names and creates a Json format of the data. Then it writes the data into a Kafka topic.

Model Training
Many machine learning applications can not be trained on a stream of data and it is related to the math behind them. The ALS algorithm too which is based on matrix factorization should be trained on all of the data at once. There is an app recommender/train_batch.py which trains an ALS model from the data which is stored on Cassandra in data ingestion phase. Then it writes the model into file system to be used by predictor app.
There should be timestamps and comparisons for the models so the predictor knows which model is the best at each time but I didn’t implemented an automation of that part yet.

Prediction
The predictor application which is implemented in recommender/stream_predictor.py is an Spark structured-streaming app which load the best trained model when it starts. Then it reads the prediction requests from one topic on Kafka and writes the result back to another topic. For a given pair of user_id and movie_id, it generates a prediction rate for that pair. Then it joins the result with other tables to get extra information like genres and title of the movie to write to Kafka for final visualization.

Visualization
To have real-time dashboards, the Elasticsearch-Kibana pair is always a good choice. Elasticsearch’s query language has some limitations in comparison with MongoDB, but having the Kibana with it, it is a good choice for real-time dashboarding. However, It is not a wise option for the dashboarding tool of a whole company for BI team.



User Id	Movie Title	Predicted Rate
443	Winter's Bone (2010)	10.951
257	Airplane! (1980)	9.739
494	Alien (1979)	8.815
439	Hobbit: An Unexpected Journey, The (2012)	8.357
531	Stand by Me (1986)	6.739
291	Iron Man 3 (2013)	6.522
559	Hercules (1997)	6.044
25	Thor: Ragnarok (2017)	5.856
407	How to Train Your Dragon (2010)	5.827
258	Green Mile, The (1999)	5.678

Figure 2 - Dashboard in Kibana

Problems

In this section I will discuss some problems and questions I've encountered during doing the project and how did I solved them.

The speed of processing

One problem with the recommender system is that it should be real-time asynchronous processing or synchronous processing. The way I handled it was to define two topics in Kafka, one for recommendation requests and another one for prediction results. recommender/stream_predictor.py app reads request from one topic and writes the result to another topic.

In my project I used Logstash to transfer the predictions to Elasticsearch. So, in the website application we can always query top rated movie predictions for specific user from

Elasticsearch. Using this trick, we always have some results for recommendation and we are updating them as fast as we can.
The reason I'm writing the result to Kafka instead of directly into Elasticsearch is that the prediction topic can have different consumers and in this architecture we can set different pipelines (e.g. Logstash) to write the predictions into different data sources for different purposes.

Rollover to new Models

One problem was about how to deploy a new model without stopping the predictions. Thanks to Kafka and how it devides the partition between two consumer with same group-id, we can simple start the new job and it would predict half of the requests. Then we can gracefully shutdown the previous job and Kafka would assign it's partitions to the new model.

Streaming Dataset

I couldn't find a live endpoint to get stream of user interactions and their historical data about rates to movies. So, I should prepare the streaming dataset myself. There were two options to simulate and streaming dataset:

1- Write a script to generate data into message queue by a specific rate
2- Write all the test dataset to Kafka and limit the rate when reading from it

I used the second option. I used the data_loading/event_stream_generator.py in my project to write some data into Kafka. Then I used **maxOffsetPerTrigger** option in predictor spark app to limit the processing rate of messages. Using this trick, I simulated a streaming interaction with website.

Results

The main thing I learned was not only about the technologies, but about the concerns when we want to deploy in production. I studied more about the fail-over recovery in spark and checkpointing. Also reviewed a couple of data bricks blog posts and Spark+AI summit videos. My work was mainly about doing the traditional recommendation in stream and so I didn't do any evaluations in terms of accuracy or other measures.

Project Summary

Title	Score (out of 20)
Getting the data	5
ETL	7
Problem	14
Algorithmic work	5
Bigness	20
UI	10
Visualization	20
Technologies	15