Music Recommendation System

Marios Lykiardopoulos
Amey Bhole
Orestis Divintari

April 2018

1 Project Summary

The main aim of this project is to create a scalable application which recommends music to the users based on their listening history.

2 Architecture

The Architecture consists of two phases one for Batch processing and Real-time Processing (Streaming). We used one Mongo Database to read and store the results generated for both Batch as well as Real-time processing. For real-time processing we used kafka for data streaming. We created a cluster of Docker Engines with 3 Virtual Machines running docker (Docker Swarm) on Google Cloud. The ubuntu-docker vm is the leader in the swarm and the docker-wrk1, docker-wrk2 are the workers in the swarm. Mongo Database runs on ubuntu-docker vm, kafka cluster runs on the ubuntu-docker vm and docker-wrk1. Spark cluster runs on the cloud with one master and two slaves.

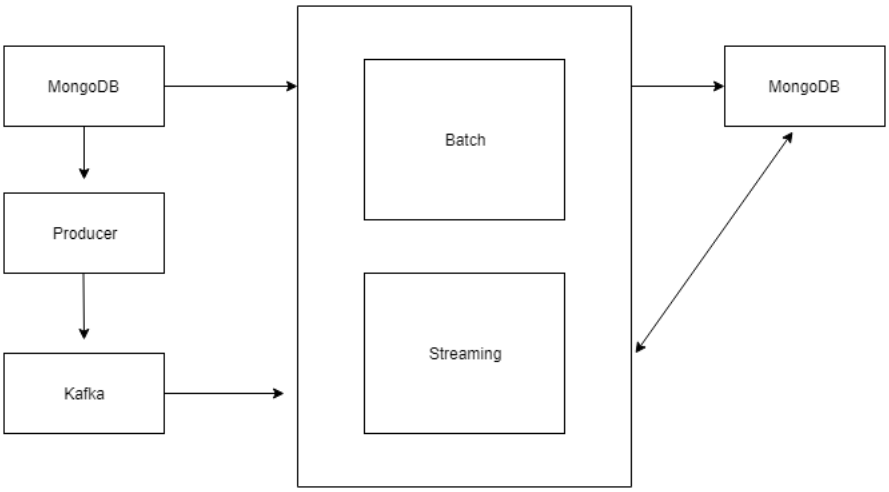

Figure 1: Architecture graph of the project

3 Dataset

The dataset use in the project is a subset of the 1 million song dataset.[1]
Dataset description:
1. 2 Million rows
2. 76,000 unique users
3. 10,000 unique songs

4 Algorithm

For Historical data to produce recommendations we used a combination Collaborative Filtering (ALS algorithm) and Global base line estimator. The ALS Model provides the basic recommendations which are then optimized using the Global baseline estimation.

4.1 Collaborative Filtering

Collaborative technique aims to fill in the missing entries of a user-item association matrix. spark.ml currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. spark.ml uses the alternating least squares (ALS) algorithm to learn these latent factors. Using the ALS implementation

from the Mlib predictions were generated for the top 30 songs and then combined with

4.2 Global Base Estimator

The Global baseline estimator is calculated using the following formula:

Overall mean listen counts + listen count deviation of user + listen count deviation of songs
Listen count deviation of user = (Average listen count of user x) - Overall mean listen counts
Listen count deviation of song = (Average listen count of song y) - Overall mean listen counts

5 Technologies Used

1. Spark: Apache Spark is a lightning-fast analytics engine for big data and machine learning. Apache Spark has as its architectural foundation the resilient distributed dataset(RDD). It is also characterized by a read-only multiset of data items distributed over a cluster of machines that is maintained in a fault-tolerant way.[2]

2. Kafka: Apache Kafka is a distributed streaming platform. Kafka aims to provide a unified, high-throughput, low latency platform for handling real time data feeds. In our project we deployed a Kafka cluster running on Google Cloud in order to deploy the streaming part of our project.[3]

3. Scala: Scala is a programming language providing support for functional programming and a strong static type system. We used Scala because it gives us the choice of scaling up our projects when we need it.[4]

4. Mongo Database: Mongo DB is a document database that provides scalability and flexibility to the user. Mongo DB is used for reading the data to generate the recommendations and to store the recommendations results. We created a database called MyHist, that includes 5 collections(Dataset, MetaData, StreamingData, HistResults and StreamingResults). The Dataset consists of the Userid , Songid, listen counts, the Userid and Song id were indexed using String indexer. MetaData contains the details of the songs such as artist name, song name , year etc. StreamingData consists of data stored which is received from the stream. HistResults and StreamingResults contains the results obtained for the recommendation.

5. Docker Swarm: A Docker Swarm is a group of machines that are running Docker and joined into cluster. We used three machines running docker in

order to make our project scalable by adding or removing machines from the swarm.

6. Google Cloud: In order to use more resources and gain more computational power we chose to use Google cloud.

References

[1] Welcome! (n.d.). Retrieved April 01, 2018, from https://labrosa.ee.columbia.edu/millionsong/

[2] Apache Spark. (2018, March 30). Retrieved April 01, 2018, from https://en.wikipedia.org/wiki/ApacheSpark

[3] Apache Kafka. (2018, March 27). Retrieved April 01, 2018, from https://en.wikipedia.org/wiki/ApacheKafka

[4] Scala (programming language). (2018, March 30). Retrieved April 01, 2018, from https://en.wikipedia.org/wiki/Scala(programminglanguage)