Analyzing The Million Song Dataset
CS755 Project Report

Anuj Sampat
asampat@bu.edu

1. Introduction

Music is something that is almost universally liked and it is a big part of our lives. A look around on online music and video sites such as YouTube shows us that people are always searching for new music to listen to. That is what was the inspiration behind this project; to analyze the dataset and find new music to recommend to users.

2. The Million Song Dataset (MSD)

The MSD dataset contains detailed audio and meta-data for one million songs. This dataset was compiled by the music intelligence company EchoNest based out of Somerville, MA and totals 300GB in size. The dataset contains various information about a given song such as song name, artist name, artist location, tempo, time signature, key, mode etc. and is available publicly for download in the form of a tab separated text file. The figure below shows a part of the field list representing a song in the dataset:

max_tempo	no	no	0.0 < tempo < **500.0** (BPM)	the maximum tempo for the song
min_tempo	no	no	**0.0** < tempo < 500.0 (BPM)	the minimum tempo for the song
max_duration	no	no	0.0 < duration < **3600.0** (seconds)	the maximum duration of any song
min_duration	no	no	**0.0** < duration < 3600.0 (seconds)	the minimum duration of any song
max_loudness	no	no	-100.0 < loudness < **100.0** (dB)	the maximum loudness of any song
min_loudness	no	no	**-100.0** < loudness < 100.0 (dB)	the minimum loudness of any song
artist_max_familiarity	no	no	0.0 < familiarity < **1.0**	the maximum familiarity of any song
artist_min_familiarity	no	no	**0.0** < familiarity < 1.0	the minimum familiarity of any song
artist_start_year_before	no	no	1970, 2011, present	Matches artists that have an earliest start year before the given value
artist_start_year_after	no	no	1970, 2011, present	Matches artists that have an earliest start year after the given value
artist_end_year_before	no	no	1970, 2011, present	Matches artists that have a latest end year before the given value
artist_end_year_after	no	no	1970, 2011, present	Matches artists that have a latest end year after the given value
song_max_hotttnesss	no	no	0.0 < hotttnesss < **1.0**	the maxiumum hotttnesss of any song
song_min_hotttnesss	no	no	**0.0** < hotttnesss < 1.0	the minimum hotttnesss of any song
artist_max_hotttnesss	no	no	0.0 < hotttnesss < **1.0**	the maxiumum hotttnesss of any song's artist
artist_min_hotttnesss	no	no	**0.0** < hotttnesss < 1.0	the minimum hotttnesss of any song's artist
min_longitude	no	no	**-180.0** < longitude < 180.0	the minimum longitude of the primary artist location
max_longitude	no	no	-180.0 < longitude < **180.0**	the maximum longitude of the primary artist location
min_latitude	no	no	**-90.0** < latitude < 90.0	the minimum latitude of the primary artist location
max_latitude	no	no	-90.0 < latitude < **90.0**	the maximum latitude of the primary artist location

Figure 1

3. System Architecture

In order to process the entire MSD, a Hadoop Distributed cluster was set up on Amazon Web Services (AWS) ElasticMapReduce with AWS instances (virtual machines) of type c3.2xlarge. Each c3 instance type consisted of 8 virtual CPUs, 15GB RAM, 2X80 GB of SSD storage and was chosen to provide maximum compute power. The Hadoop cluster consisted of one master and ten slaves running Hadoop version 0.20.2. In addition, Amazon S3 storage was used to store the input MSD data and the final song output from the Hadoop HDFS.

MapReduce tasks were run to look for songs with user given input attributes such as tempo (slow, medium, fast), year of release, key (C, E flat, F sharp, etc.), mode (major or minor). The map task was responsible for matching songs with the given attributes and outputting the song with the following key, value:

key → song attribute such as hotness
value → song name, artist name, year of release

The reduce task sorted the map output based on the key and output the final list. In this way, songs were sorted based on attributes such as hotness etc.

In addition, Hadoop counters were used to keep track of various numbers such as song count and total dance-ability of the songs in the list. This allowed for calculating the average dance-ability of songs in a given song key (for ex. key of C) or time range (for ex. 1960-1979).

A web server hosting a web page that allowed users to interact with this system and search for songs based on attributes was also set up. The web page ran a PHP script that launched a Java program to submit a MapReduce job to the Amazon Hadoop cluster and read the final song output from the Hadoop HDFS/S3 storage and present it to the user.

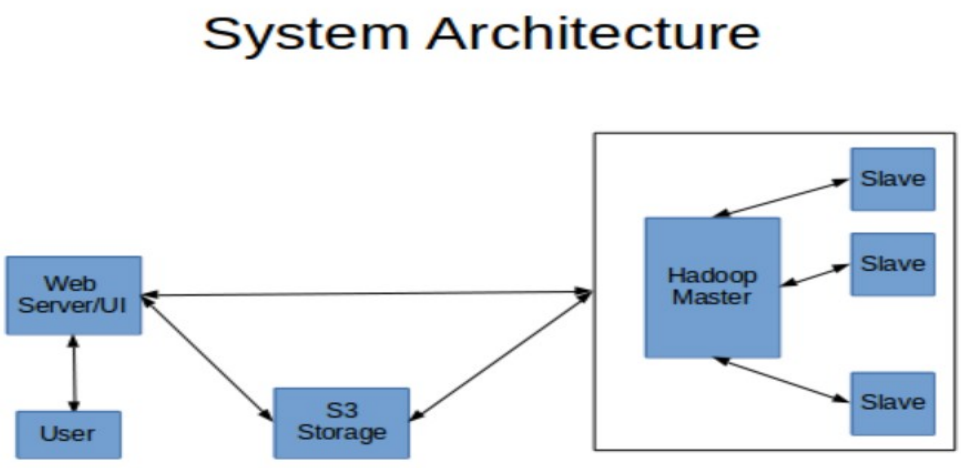

Figure 2

4. Results

Case 1

- Song attributes
  - Region: North America
  - Year: 1999 - 2014
  - Tempo: Slow
  - Key: E
  - Mode: Minor

A total of 8302 songs was found and the average dance-ability was low.

Case 2

- Song attributes
  - Region: North America
  - Year: 1999 - 2014
  - Tempo: All (slow, medium, fast)
  - Key: G
  - Mode: Major

A total of 210673 songs was found and the average dance-ability was high.

In both cases, the total time taken for the MapReduce job to process the entire 300GB MSD was around 13 minutes. An excerpt of the data obtained from the tests run in both these cases is attached with the report.

5. Conclusions and Future Work

It was found that the Amazon Elastic MapReduce cluster was ideal for processing the entire dataset. However, for smaller subsets of this data, the overhead of creating a job on the cluster and fetching output data from S3 storage was high and it resulted in slower presentation of results to the user. This could be solved by using a bigger cluster to speed up the data analysis; the S3 storage fetch time would still be an issue though.

The web server hosting the web UI could be part of a bigger system running multiple servers inside an Amazon Elastic Load Balancer with Queues for handling heavy demand.

Audio links to song samples can be presented to the user. Finally, content based filtering can be used to analyze the MSD for songs similar to a given input song.

(Please see next page for the list of files submitted with this report)

Files submitted with this report are as follows:

AWS_Driver.java: AWS driver to connect to an Amazon EMR cluster, submit a MR job and download the MR job output from Amazon S3 storage to local disk for presentation to the user.

MSD_Driver.java: MapReduce driver for launching a MR job on the Amazon EMR cluster.

MSD_Mapper.java and MSD_Reducer.java: The map and reduce functions.

msd.html: The HTML web UI that triggers a PHP script on user song search.

msd_input.php: The PHP script to execute the AWS_Driver to launch the MR job to get songs.

ui.png: A screenshot of the UI.

demoListContSlowMinE: Excerpt of the song results for the test case (1) described in section 4.

demoListContAllMajG: Excerpt of the song results for the test case (2) described in section 4.