DIRECTED ACYCLIC GRAPH USING APACHE SPARK

TEAM 7:

MONODEEP KAR, VINISH CHAMRANI, AKSHAY PHADKE,

DANIEL MORTON, DYLAN SLACK, ZEHENG CHEN.

PROBLEM DEFINITION

- We are planning to use Directed Acyclic Graphs in Apache Spark in order to make song recommendations from current song.
- Use MapReduce, Machine Learning, and Content Based filtering for song recommendation
- Recommend items to user similar to previous items listened by user frequently
- Example: Recommend songs with same singer(s), genre, etc.

TASKS ACCOMPLISHED

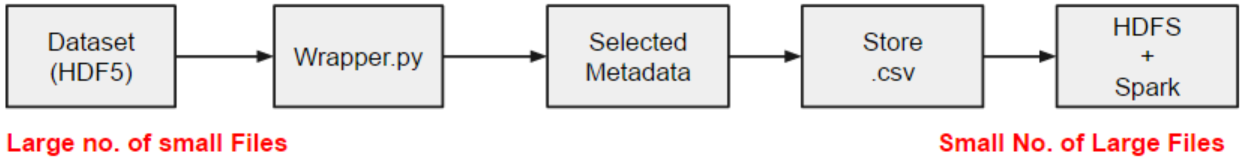
- **Million Song Dataset:** We studied the structure and format of the dataset and identified the metadata we can use for the recommendation system

- **Installing Apache Spark**: We used Ambari Server to setup the Spark application to check if it runs correctly and ran a few sample programs in Spark.

- **MapReduce Jobs**: We ran a few MapReduce jobs to find out the artist with most number of songs (Most recorded artist) and to return all the songs on which a particular artist worked on either individually or in collaboration.

MILLION SONG DATASET

- Available at labrosa.ee.columbia.edu/millionsong/pages/getting-dataset
  - It is a distributed database of size of about 300 GB
- We worked on the subset of given database “Million Song Subset” which contains information of about 10,000 songs which is about 1.8 GB in size.
- The dataset itself is in the HDF5 file format and comes with various wrappers to retrieve data from this file format.
  - A versatile data model that can represent very complex data objects and a wide variety of metadata.
- Using python wrapper which searches through the directory to find all files with extension .h5, we retrieved some of the required metadata from the entire subset and stored it as CSV file.

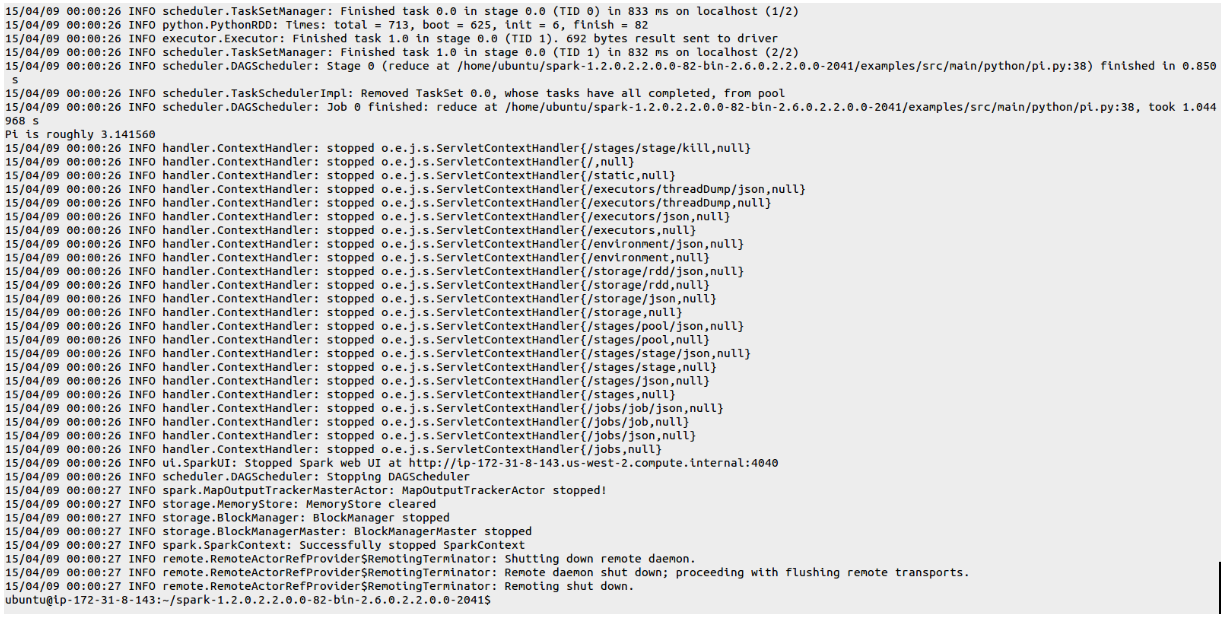


MILLION SONG METADATA

- The dataset has about 46 fields of metadata out of which we identified 10 for our purpose of song recommendation:

1. **artist_name**: Can be used for querying as well as clustering
2. **artist_playmeid**: Can be used for querying
3. **danceability**: Can be used to determine type of song and for clustering
4. **duration**: Can be used for clustering
5. **similar_artists**: Can be used for direct recommendations
6. **song_hotness**: Can be used to assign priority to the recommended songs
7. **tempo**: Can be used for clustering
8. **song_title**: For display purposes
9. **year**: For displaying and for clustering
10. **release**: Corresponds to album name and can be used for recommendations

INSTALLING APACHE SPARK

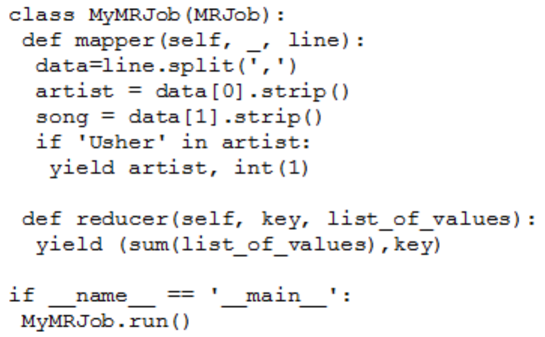


```
15/04/09 00:00:26 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 833 ms on localhost (1/2)
15/04/09 00:00:26 INFO python.PythonRDD: Times: total = 713, boot = 625, init = 6, finish = 82
15/04/09 00:00:26 INFO executor.Executor: Finished task 1.0 in stage 0.0 (TID 1). 692 bytes result sent to driver
15/04/09 00:00:26 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 832 ms on localhost (2/2)
15/04/09 00:00:26 INFO scheduler.DAGScheduler: Stage 0 (reduce at /home/ubuntu/spark-1.2.0.2.2.0.0-82-bin-2.6.0.2.2.0.0-2041/examples/src/main/python/pi.py:38) finished in 0.850
 s
15/04/09 00:00:26 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
15/04/09 00:00:26 INFO scheduler.DAGScheduler: Job 0 finished: reduce at /home/ubuntu/spark-1.2.0.2.2.0.0-82-bin-2.6.0.2.2.0.0-2041/examples/src/main/python/pi.py:38, took 1.044
968 s
Pi is roughly 3.141560
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage/kill,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/static,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/threadDump/json,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/threadDump,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors/json,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/executors,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/environment/json,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/environment,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/rdd/json,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/rdd,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage/json,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/storage,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/pool/json,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/pool,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage/json,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/stage,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages/json,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/stages,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/job/json,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/job,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs/json,null}
15/04/09 00:00:26 INFO handler.ContextHandler: stopped o.e.j.s.ServletContextHandler{/jobs,null}
15/04/09 00:00:26 INFO ui.SparkUI: Stopped Spark web UI at http://ip-172-31-8-143.us-west-2.compute.internal:4040
15/04/09 00:00:26 INFO scheduler.DAGScheduler: Stopping DAGScheduler
15/04/09 00:00:27 INFO spark.MapOutputTrackerMasterActor: MapOutputTrackerActor stopped!
15/04/09 00:00:27 INFO storage.MemoryStore: MemoryStore cleared
15/04/09 00:00:27 INFO storage.BlockManager: BlockManager stopped
15/04/09 00:00:27 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
15/04/09 00:00:27 INFO spark.SparkContext: Successfully stopped SparkContext
15/04/09 00:00:27 INFO remote.RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
15/04/09 00:00:27 INFO remote.RemoteActorRefProvider$RemotingTerminator: Remote daemon shut down; proceeding with flushing remote transports.
15/04/09 00:00:27 INFO remote.RemoteActorRefProvider$RemotingTerminator: Remoting shut down.
ubuntu@ip-172-31-8-143:~/spark-1.2.0.2.2.0.0-82-bin-2.6.0.2.2.0.0-2041$
```

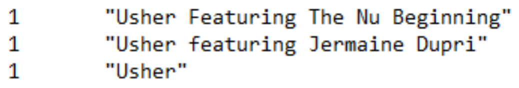
MAPREDUCE

- The MapReduce was used to compute the following things based on the Million Song Subset:

1. The artist with most number of songs
2. The list of songs produced by artist individually and in Collaboration

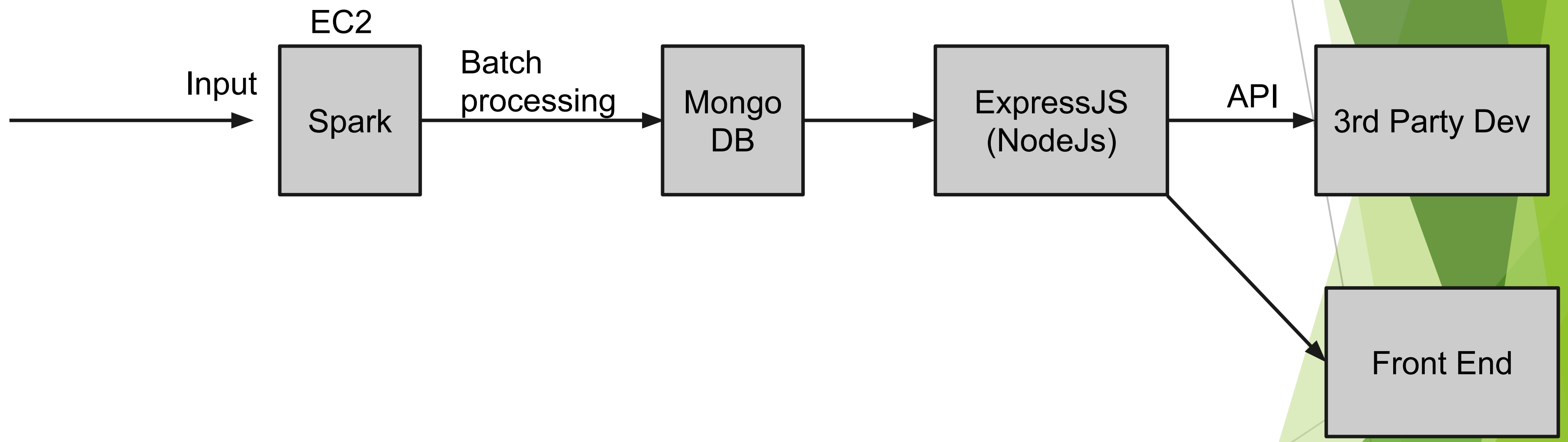


```
class MyMRJob(MRJob):
 def mapper(self, _, line):
  data=line.split(',')
  artist = data[0].strip()
  song = data[1].strip()
  if 'Usher' in artist:
   yield artist, int(1)

 def reducer(self, key, list_of_values):
  yield (sum(list_of_values),key)

if __name__ == '__main__':
 MyMRJob.run()
```



```
1       "Usher Featuring The Nu Beginning"
1       "Usher featuring Jermaine Dupri"
1       "Usher"
```

ARCHITECTURE



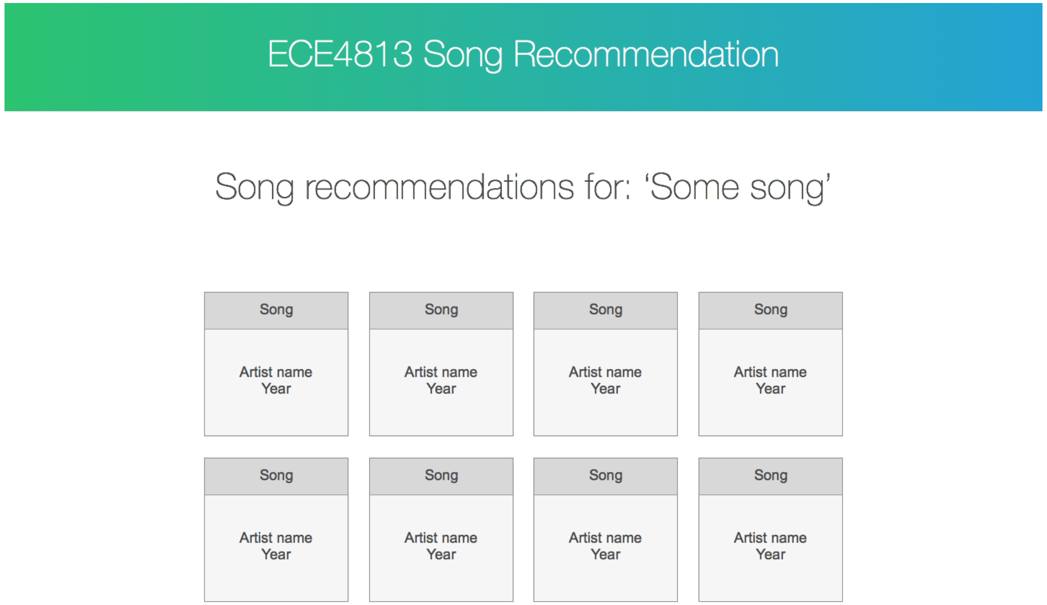
RESTFUL API

Examples: GET api/songs/:song_id, GET api/artists/:artist_id, GET api/charts?type={artist|year}, GET api/songs?name='song_name'

```
song
{
        id: (primary key)
        similar: [{
                song_id: 12,
                name: 'Some song'
        }],
        artist_id: 625,
        name: 'Primary song name',
        year: 2014
}
```

```
artist
{
        id: (primary key)
        similar: [{
                artist_id: 12,
                name: 'Artist
name''
        }],
        songs: [{
                song_id: 53,
                name: 'Song name'
        }]
}
```

FRONTEND GUI



ECE4813 Song Recommendation

Song recommendations for: 'Some song'

Song Artist name Year	Song Artist name Year	Song Artist name Year	Song Artist name Year
Song Artist name Year	Song Artist name Year	Song Artist name Year	Song Artist name Year

ISSUES AND CHALLENGES

- Challenge: implementing the database as dynamic
- Issue: multiple users accessing the same account (e.g. multiple family members)

INDIVIDUAL CONTRIBUTION

- Monodeep Kar: Spark and Dataset
- Vinish Chamrani : Spark and Database
- Akshay Phadke : Spark and Machine Learning
- Dylan Slack: API and Front-End
- Zeheng Chen: Database and API
- Daniel Morton: Database