Big Data Analysis on Million Song Dataset (MSD)

ECE4710: Methods and Tools for Big Data

Yiding Chang Yifan Shen Kexuan Huang Qinhang Wu

July 28, 2022

Overview

- Milestone 0: HDF5 Data Pre-process
- Milestone 1: Drill Database Query
- Milestone 2: Advanced Data Analysis

Section 1

Milestone 0: HDF5 Data Pre-process

Goals

1. Compact small `hdf5` files into larger one
2. Read `hdf5` file and extract the information
3. Convert `hdf5` to `Avro` with `Apache Avro`

1. Compact small `hdf5` files into larger one

```
$ python3 create_aggregate_file.py <IN> <OUT>
```

- Input: a directory contains `hdf5` song files
- Output: an aggregate `hdf5` song file
- Example:

```
~/De/ECE4721J/p/p/m0  master !3  python3 src/create_aggregate_file.py MillionSo
ngSubset/ data/compact.h5

need pg module and MBrainzDB folder of Python source code if you
want to use musicbrainz related functions, e.g. fill_hdf5_from_musicbrainz
found 10000 H5 files.
Aggregate file created, we start filling it.
 17%|                                        | 1721/10000 [02:06<08:55, 15.45it/s]
```

Figure 1: Compact 10000 `hdf5` files into larger one

2. Read `hdf5` files and extract the information

```
$ python3 display_song.py [FLAGS] <HDF5> <idx> <field>
```

- Input: an `hdf5` song file
- Output: specified field content
- Example:

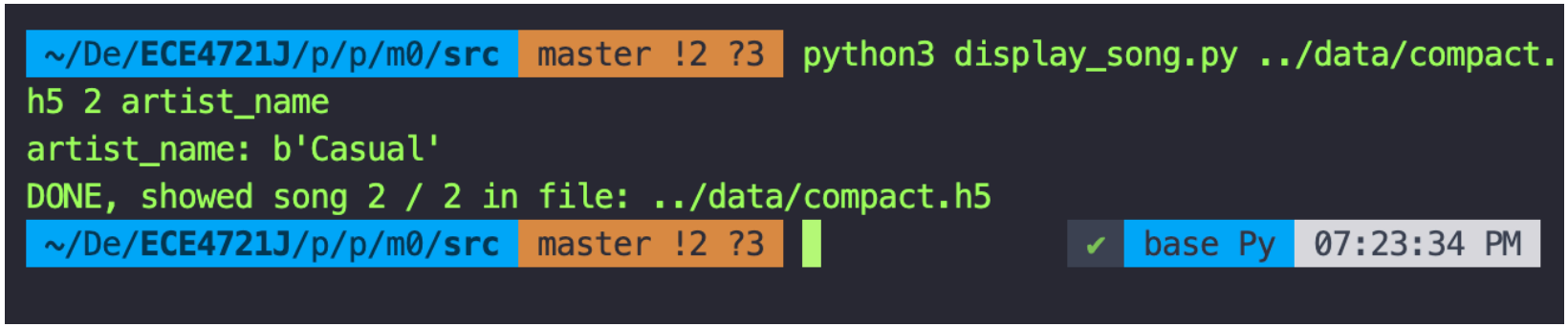


Figure 2: Get artist name of the second song in compacted `hdf5` file

3. Convert hdf5 to Avro with Apache Avro

```
$ hdf5_to_avro.py [-h] -s <SCHEMA> -i <HDF5> -o <AVRO>
```

- Input:
  - an Avro schema file
  - an hdf5 song file to be converted
- Output: an Avro song file

Sample schema file in `json` format:

```
{
  "namespace": "song.avro",
  "type": "record",
  "name": "Song",
  "fields": [
    {
      "name": "artist_name",
      "type": ["string", "null"]
    },
    {
      "name": "title",
      "type": ["string", "null"]
    }
  ]
}
```

```
root@hadoop-master:/home/s/pj1/m0# python3 src/hdf5_to_avro.py -s schema/songs
.avsc -i data/compact.h5 -o data/output.avro
21:18:10 [Info] Convert a song file from hdf5 to Avro...
21:18:10 [Info] Avro schema path: schema/songs.avsc
21:18:10 [Info] hdf5 input path: data/compact.h5
21:18:10 [Info] Avro output path: data/output.avro
21:18:10 [Info] Avro schema file and hdf5 file exist
21:18:10 [Warning] Avro output file data/output.avro already exists
21:18:10 [Info] Parsing the Avro schema file...
21:18:10 [Info] Get the following fields:
                artist_hotttnesss   ["float", "null"]
                artist_id           ["string", "null"]
                artist_name         ["string", "null"]
                duration            ["float", "null"]
                energy              ["float", "null"]
                release             ["string", "null"]
                song_hotttnesss     ["float", "null"]
                song_id             "string"
                tempo               ["float", "null"]
                title               ["string", "null"]
                track_id            ["string", "null"]
                year                ["int", "null"]
21:18:10 [Info] Found 10000 song(s)
21:18:10 [Info] Start converting hdf5 to Avro
21:18:10 Converting: 100%|          | 10000/10000 [00:22<00:00, 436.24it/s]
```

Figure 3: Convert compacted `hdf5` file to `Avro`

Reference

1. MSongsDB

   `https://github.com/tbertinmahieux/MSongsDB`

2. MSongsDB Field List

   `http://millionsongdataset.com/pages/field-list/`

3. Apache Avro Documentation

   `https://avro.apache.org/docs/current/index.html`

Section 2

Milestone 1: Drill Database Query

Goals

Query Million Song Dataset (MSD) with `Drill`:

1. Find the range of dates covered by the songs in the dataset
2. Find the hottest song that is the shortest and shows highest energy with lowest tempo
3. Find the name of the album with the most tracks
4. Find the name of the band who recorded the longest song

1. The range of dates covered by the songs

- SQL:

```
-- Age of the oldest songs
SELECT 2022 - MAX(year) AS Age
FROM hdfs.`/pj/m0/output.avro`;

-- Age of the youngest songs
SELECT 2022 - MIN(year) AS Age
FROM hdfs.`/pj/m0/output.avro`
WHERE year > 0;
```

1. The range of dates covered by the songs

- Results:

```
+--------+          +--------+
|  Age   |          |  Age   |
+--------+          +--------+
| 12     |          | 96     |
+--------+          +--------+
1 row selected      1 row selected
```

The oldest song's age is **96** and the youngest is **12**. As a result, the range of dates covered by the songs is **84** years.

2. The hottest song that is the shortest and shows highest energy with lowest tempo

- SQL:

```
SELECT title
FROM hdfs.`/pj/m0/output.avro`
WHERE song_hotttnesss <> 'NaN'
ORDER BY song_hotttnesss DESC,
    duration ASC,
    energy DESC,
    tempo ASC
LIMIT 10;
```

- Remarks: This query returns **5648** results, but we only display the first **10** records.

2. The hottest song that is the shortest and shows highest energy with lowest tempo

```
+-----------------------------------------------------+
|                        title                        |
+-----------------------------------------------------+
| b'Immigrant Song (Album Version)'                   |
| b"Nothin' On You [feat. Bruno Mars] (Album Version)" |
| b'This Christmas (LP Version)'                      |
| b'If Today Was Your Last Day (Album Version)'       |
| b'Harder To Breathe'                                |
| b'Blue Orchid'                                      |
| b'Just Say Yes'                                     |
| b'They Reminisce Over You (Single Version)'         |
| b'Exogenesis: Symphony Part 1 [Overture]'           |
| b'Inertiatic Esp'                                   |
+-----------------------------------------------------+
10 rows selected (0.471 seconds)
```

3. The name of the album with the most tracks

- SQL:

```
SELECT release, COUNT(release) AS NumTrack
FROM hdfs.`/pj/m0/output.avro`
GROUP BY release
ORDER BY NumTrack desc
LIMIT 1;
```

- Results:

```
+------------------+----------+
|     release      | NumTrack |
+------------------+----------+
| b'Greatest Hits' | 21       |
+------------------+----------+
1 row selected (0.695 seconds)
```

4. The name of the band who recorded the longest song

- SQL:

```
SELECT artist_name, duration
FROM hdfs.`/pj/m0/output.avro`
ORDER BY duration DESC
LIMIT 1;
```

- Results:

```
+-------------+-----------+
| artist_name | duration  |
+-------------+-----------+
| b'UFO'      | 1819.7677 |
+-------------+-----------+
1 row selected (0.27 seconds)
```

Section 3

Milestone 2: Advanced Data Analysis

Goals

1. Determine distance between artists with adjacency matrix, using parallelized BFS
2. Propose similar songs with distance and "provide more diverse recommendations"
3. Do it in both MapReduce and Spark to compare the time
4. Prepare a set of slides with Beamer and a poster with Beamerposter for presentations

BFS with MapReduce

Runs BFS on the MillionSongDataset to search for similar artists to a certain artist within distance 3 with MapReduce:

BFS with MapReduce - A Simple Example

Let's say we want to find artists similar to **A** with distance **3**, and we have following relationships (each edge has distance **1**):

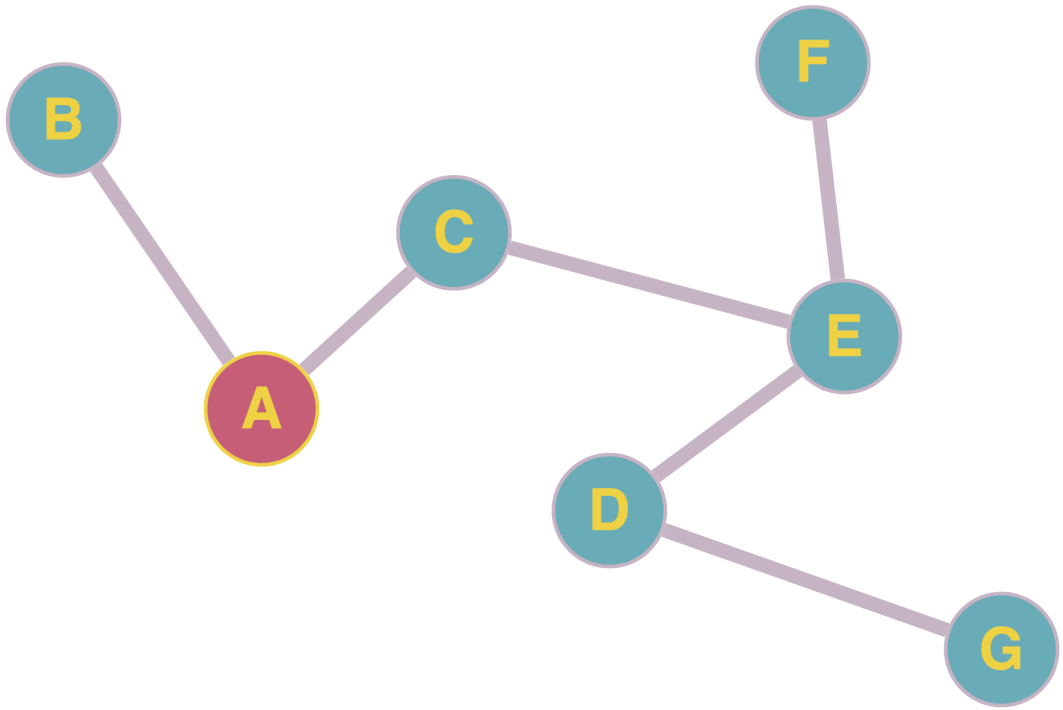
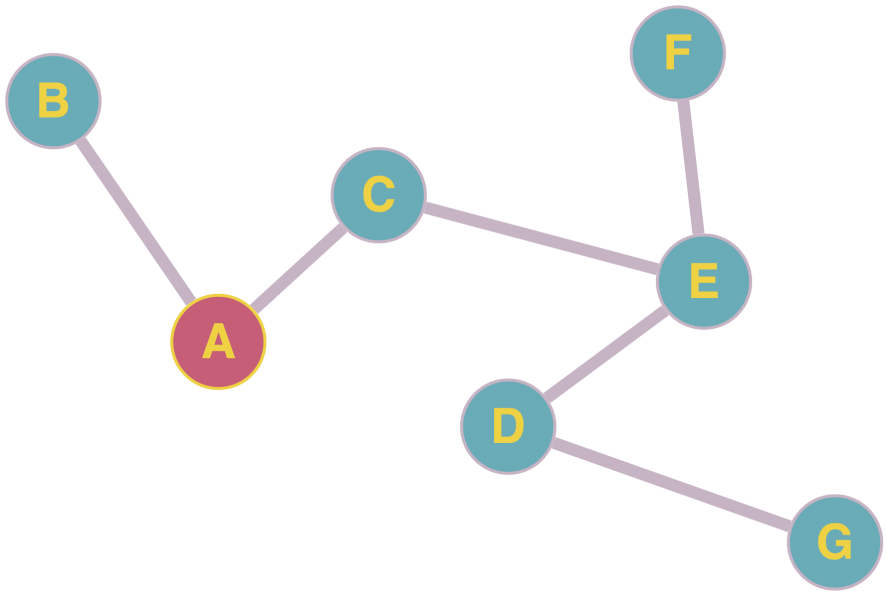


Figure 4: Similarity Graph

Step 1: Initialize Graph File with Target Artist

Format: each line contains **Node** | **Distance** | **Neighbours**




```
A | 0   | B,C
B | inf | A
C | inf | A,E
D | inf | E,G
E | inf | C,D,F
F | inf | E
G | inf | D
```

Figure 5: Initialize Graph File

Step 2: Generate Distance Pair in Mapper

Format: each line contains **Neighbours, Node, Distance+1**

```
A | 0   | B,C
B | inf | A
C | inf | A,E
D | inf | E,G
E | inf | C,D,F
F | inf | E
G | inf | D
```

Mapper →

```
B, A, 1
C, A, 1
A, B, inf
A, C, inf
E, C, inf
E, D, inf
G, D, inf
C, E, inf
D, E, inf
F, E, inf
E, F, inf
D, G, inf
```

Sort →

```
A, B, inf
A, C, inf
B, A, 1
C, A, 1
C, E, inf
D, E, inf
D, G, inf
E, C, inf
E, D, inf
E, F, inf
F, E, inf
G, D, inf
```