Review of A8-Xia

Bishwajeet Dey

November 17, 2017

Compilation issues

1. This report reviews commit - 9e881a5b2a4f369cb3741acf9924d91233d1e953
2. `make gunzip` had some errors while unzipping but it unzipped successfully.

```
gunzip input/MillionSongSubset/*; gunzip input/MillionSongSubset/*
gzip: input/MillionSongSubset/artist_terms.csv: unknown suffix -- ignored
gzip: input/MillionSongSubset/similar_artists.csv: unknown suffix -- ignored
gzip: input/MillionSongSubset/song_info.csv: unknown suffix -- ignored
Makefile:30: recipe for target 'gunzip' failed
make: [gunzip] Error 2 (ignored)
```

3. However, the program compiled and ran successfully. It took 9 minutes to process the one million song dataset.

Report & Code issues

- The number of distinct albums does not match with the expected total of `221751`.

  `Number of distinct albums: 149275`

  - This can be traced to the following lines in `millionsong.scala`.

```
private def getDistinctAlbum(x: RDD[SongInfo]) = {
  val artistAlbumTuple = x.map(line => (line.getArtId(), line.getAlbum())).distinct()
  val numbleOfAlbum = artistAlbumTuple.countByKey().foldLeft(0l)(_ + _._2)
  println("Number of Albums is: ", numbleOfAlbum)
}
```

  - From the above snippet, the author is trying to count distinct `<artistId, albumId>` pairs. One artist may have more than one albums. So such albums were not counted. Also, instead of using `countByKey` followed by `foldLeft`, the author could have used `count`. It can be easily fixed by removing artist id from the tuple.

- Consider the following snippet of scala code in spark shell with its output:

```
val rdd = sc.parallelize(List(("ABC", 1.5), ("DEF", 2.1), ("AAC", 1.8)))

//snippet 1 - WRONG result
scala> rdd.sortBy(_._2).top(2)
  res0: Array[(String, Double)] = Array((DEF,2.1), (ABC,1.5))

//snippet 2 - Correct result
scala> rdd.sortBy(_._2, false).take(2)
  res2: Array[(String, Double)] = Array((DEF,2.1), (AAC,1.8))
```

`top` implicitly uses the first field instead of second field as intended. Now, let's look at author's code for all the functions which use `top`:

```
getTop5(records, "loudness")
getTop5(records, "length")
getTop5(records, "tempo")

val songLoudnessTuple = x.map(..).distinct()
val top5LoudestSongs = songLoudnessTuple.sortBy(_._2, false).top(5).toList

val artistFamiliarityTuple = x.map(..).distinct()
val top5FamiliarArtist = artistFamiliarityTuple.sortBy(_._2, false).top(5).toList

val artistHotnessTuple = x.filter(..).distinct().persist()
val top5HottestArtist = artistHotnessTuple.sortBy(_._2, false).top(5).toList

val songHotnessTuple = x.filter(..).distinct()
val top5HottestSongs = songHotnessTuple.sortBy(_._2, false).top(5).toList
```

All the above instances produce the wrong result. The `top5LoudestSongs` gets called from `getTop5` function. Thus, top5 - `song loudness, song length, song tempo, familiar artists, hottest artist, hottest songs` produce the wrong output. The files written in the output directory are different from the ones which are presented in the report. All above `top` instances should be replaced by `take`.

- Instead of reading from the output files produced by the program, the author has directly pasted results in plain text which the program does not produce. Consider the following snippet from the report:

```
  #### Top 5 loudest songs:
(Assumption: a larger loudness implies a louder song)
Song Id                          | Song Name                           | Loudness Score
---------------------------------|-------------------------------------|---------------------------
SOCMYZF12AB0186FF4               | Modifications                       | 4.318
SONHDXQ12AB018C1F1               | Track 02                            | 4.3
SOZCIHV12AC46894E3               | Hey You Fuxxx!                      | 4.231
SOEJMJF12AC90715D1               | War Memorial Exit                   | 4.166
SOBEMPS12A8C13BD46               | Meta Abuse                          | 4.15
```

The actual results from `output/top5loudness` results are:

```
(SOZZZZM12AB018B095,Be Anything),-12.62
(SOZZZZH12A8C140F37,Climate Control),-8.852
(SOZZZYE12A81C22900,Down The Road),-9.519
(SOZZZWN12AF72A1E29,Peter Pan),-8.418
(SOZZZWG12A8C14184C,Ik Nazar Dekh Le Khawaja),-18.507
```

Similarly for top 5 fastest songs. Following is the text in the report:

```
#### Top 5 fastest songs:
(Assumption: a larger  tempo implies a faster song)

Song Id                          | Song Name                         | Tempo Score
---------------------------------|-----------------------------------|-----------------------------
SOVVTEZ12AB0184AAB               | Beep Beep                         | 302.3
SOMSJWX12AB017DB99               | Late Nite Lounge: WVIP            | 296.469
SOUTBKH12A8C136286               | A Place Called Hope               |285.157
SOEVQJB12AC960DA2C               | Bellas Lullaby - Perrier Citron |284.208
SOTUXOB12AB0188C3A               | Troubled Times                    |282.573
```

However, the output from `output/top5tempo` is:

```
(SOZZZZM12AB018B095,Be Anything),65.877
(SOZZZZH12A8C140F37,Climate Control),124.004
(SOZZZYE12A81C22900,Down The Road),156.746
(SOZZZWN12AF72A1E29,Peter Pan),88.181
(SOZZZWG12A8C14184C,Ik Nazar Dekh Le Khawaja),175.179
```

This difference in output also exists for other files - `output/top5length`, `output/top5HottestSongs`, `output/top5HottestArtist`, `output/top5FamiliarArtist`. All these files have incorrect results. However, the report with text fragments for all these queries has correct results.

- The top 5 prolific artist in the report is more wrong than the one which exists in the `output/top5ProlificArtist`. From the report:

```
Top 5 Prolific artists:
Artist Id Artist Name Number of songs by the artist
AR6681Y1187FB39B02 Ike & Tina Turner 208
ARXPPEY1187FB51DF4 Michael Jackson 204
ARH861H1187B9B799E Johnny Cash 201
AR8L6W21187B9AD317 Diana Ross & The Supremes 196
ARLHO5Z1187FB4C861 Beastie Boys 194
```

The actual output file contains:

```
(ARH861H1187B9B799E,Johnny Cash),189
(ARLHO5Z1187FB4C861,Beastie Boys),184
(AR60DO1187FB4D9AB,Joan Baez),181
(ARXPPEY1187FB51DF4,Michael Jackson),179
(AR40GSU1187FB3AA01,Neil Diamond),176
```

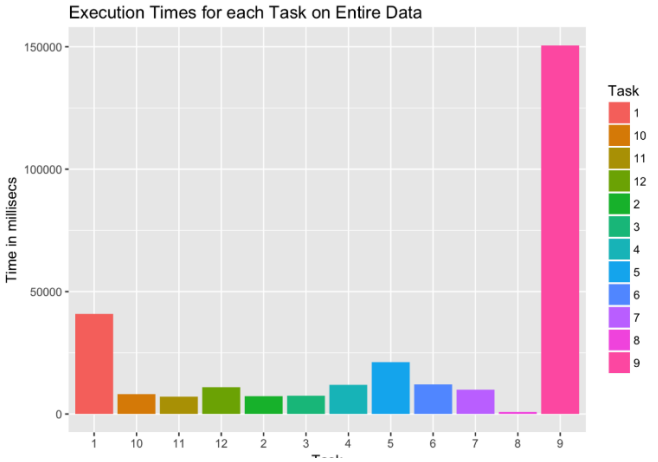
- The top 5 keys in the file: `output/topPopularKeys` contain the right keys. However, the author has presented a different text in the report:

```
Top 5 most popular keys (must have confidence > 0.7):
Key Ocurrances
  7 53144
  9 46855
  2 46431
  4 35099
  11 35078
```

The output file contains:

```
7,30420
0,28333
2,25845
9,21283
4,15214
```

```
##    Task          Query performed
## 1     1        Find distinct songs
## 2     2      Find distinct artists
## 3     3       Find distinct albums
## 4     4   Find top 5 loudest songs
## 5     5  Find top 5 longest songs
## 6     6  Find top 5 fastest songs
## 7     7  Find top 5 hottest songs
## 8     8 Find top 5 familiar artists
## 9     9  Find top 5 hottest artists
## 10   10  Find top 5 hottest Genres
## 11   11 Find top 5 most popular keys
## 12   12 Find top 5 prolific artists
## 13   13    Find top 5 common words
```



- The author's plot legends and the graph are not in sync. `Task 9` in pink from the legends is `9 Find top 5 hottest artists`. However, the author talks about computing `hottest genres` which takes the maximum amount of time which is marked as `Task 10` in the legend. it would be nice to have time in minutes rather than milliseconds. It is hard to see that the total execution time is sum of all queries. It would be nice to see the bar plot/stacked chart of the time to get the hottest genres vs the total time.
- The author is trying to compare persist vs no-persist computation time of queries which are computed only once. The query intermediate steps are small enough to fit in memory. So, with persist, it takes slightly more time. It would be nice to see graphs side by side. The author comes to an incorrect conclusion for the difference - `This is because we are running our code in a standalone mode rather than in a distributed mode.`
- The program does not have any snippet which measures time for all queries. Also, the files which measure time per query were not generated when I ran the program.
- Many code snippets use `toSeq` and then sort the data. This defeats the purpose of using spark since sorting is now done at the driver's memory.

```
//Snippet from millionsong.scala
99:  val top5PopularKey = keyCountTuple.countByKey().toSeq.sortWith(_._2 > _._2).take(5).toList
109: val top5ProlificArtists = artistSongTuple.countByKey().toSeq.sortWith(_._2 > _._2)
                                                    .take(5).toList
164: val top5Words = wordsCount.countByKey().toSeq.sortWith(_._2 > _._2).take(5).toList
```

- Scala naming conventions should be used for naming classes. Class `millionSong` could have been named better.
- The report contains many typos.