BI Assignment 3

0427561 - Bernhard Müller
1226913 - Marian Amann

February 10, 2017

1 MapReduce product review analysis in Java

In this section three datasets are analysed with the help of `MapReduce`. The datasets contain product reviews and metadata of articles from Amazon. The datasets are available at http://jmcauley.ucsd.edu/data/amazon/. The three chosen datasets were the `Digital Music`, `Patio, Lawn and Garden` and `Grocery and Gourmet Food` datasets (both `5-core` and `ratings only`).

In the first subsection the average ratings of the given datasets will be calculated and in the second subsection the overall sentiment for articles will be analysed.

The programs were tested and developed in the recommended Cloudera Quickstart VM.

1.1 Average Ratings

Program Usage The program excepts two Arguments: An Input File, which contains the articles and an Output Destination. The program assumes all parameters are correct/valid.

After the Program is executed the resulting Averages will be found in the specified output path.

Thus the Synopsis has the following form (given the jar exists; can be executed with hadoop/-java):
`AverageReviewScoreAmazon.jar inputFile outputPath`.

Implementation The program was implemented analogous to the Wordcount program (as discussed in the first Part of the Big Data Lecture on Slides 44-47). Therefore three classes were implemented: a Driver (i.e `ReviewScoreAverageDriver`), a Mapper (i.e `ReviewScoreMapper`) and a Reducer (i.e `AverageReducer`). Each of these classes only have one method of interest (i.e the main function, the map method and the reduce method respectively).

The Driver sets up/configures the job and starts it. The Mapper then puts for every line in the input file the article and the respective rating out, which means the Output Key-Value type is `<Text,DoubleWritable>`.

The Reducer then sums the rating for every article together and divides it by the number of instances, which means the Output type is the same as in the Mapper.

Listings for the map/reduce methods are included to demonstrate the approach to the problem.

Listing 1: Excerpt from the map method (Average Review) Code

```
String line = value.toString();
StringTokenizer itr = new StringTokenizer(line,",");
String keyS;
itr.nextToken();
keyS = itr.nextToken();
String word = itr.nextToken();
double rating = Double.valueOf(word);
output.collect(new Text(keyS), new DoubleWritable(rating));
```

Listing 2: Excerpt from the reduce method (Average Review) Code

```
double count = 0;
double sum = 0;
while(values.hasNext()) {
        sum += values.next().get();
        count++;
}
output.collect(key, new DoubleWritable(sum/count));
```

Scaling Scaling out in the case of these three Datasets will likely not be of too much of a benefit (there might be some slight positive effect, because of the large amount of records more nodes could be beneficial), since every file is under 64 MB (typical Block size for HDFS). It's also possible that more nodes would result in more (unnecessary) overhead or that it wouldn't really optimize the disk I/O. With bigger/more datasets/files more nodes would probably be appropriate and the program should scale in an acceptable (given there are enough nodes) way (i.e. graceful decline). The scaling with bigger/more files should have a good (linear) scaling (as it was with these small subsets). Vertical scaling is most likely also not much of a benefit because of the aforementioned small sizes of the datasets and the calculations aren't too complex.

Excerpt from the resulting averaged ratings In the following the first 15 results of the average ratings (for every dataset) will be listed. The article column contains the `Amazon Standard Identification Number`.

Table 1: Average Ratings for the Digital Music Dataset

Article	Average Rating
5555991584	4.64
6308051551	5.0
7901622466	4.82
B0000000ZW	4.33
B00000016T	4.72
B00000016W	4.43
B00000017R	4.77
B0000001BA	4.8
B0000001BO	4.67
B0000001O0	5.0
B0000001P4	4.63
B0000001PS	4.86
B0000001Q8	4.67
B0000001RE	5.0
B0000001SH	4.4

Table 2: Average Ratings for the Grocery and Gourmet Food Dataset

Article	Average Rating
0657745316	5.0
0700026444	5.0
1403796890	2.0
141278509X	5.0
1453060375	1.0
1453060464	3.0
1453060782	4.0
1603112251	3.0
1613170416	4.75
1837994021	5.0
3295000018	4.84
3301261876	4.5
3621813330	3.0
5628754218	4.67
5901002482	3.0

Table 3: Average Ratings for the Patio, Lawn and Garden Dataset

Article	Average Rating
0981850006	5.0
144072007X	3.65
1554701503	5.0
1579822932	4.75
1754164498	5.0
1880241064	2.5
1885010753	4.0
1938146824	4.5
398501938X	5.0
6035000029	4.33
6035000037	1.0
6035000045	4.5
6303146775	5.0
6304429150	4.51
8386244453	5.0

1.2 Sentiment Analysis

The Sentiment Analysis try to measure the overall sentiment (positive or negative?) in a review. For this a list of positive words and negative words is needed. The sentiment is then calculated like this: (positive-negative) / (positive+negative)

Program Usage The program excepts four Arguments: An Input File, which contains the articles, an Output Destination, a positive wordlist and a negative wordlist. The program assumes all parameters are correct/valid.

After the Program is executed the resulting Sentiments will be found in the specified output path.

Thus the Synopsis has the following form (given the jar exists; can be executed with hadoop/-java):

```
java -jar AvgReviewScore.jar inputFile outputPath positive negative.
```

Implementation The program was implemented analogous to the Sentiment example given in the assignment.[1]

[1] https://www.cloudera.com/documentation/other/tutorial/CDH5/topics/ht_example_4_sentiment_analysis.html

As before three classes were implemented: a Driver (i.e `SentimentAmazonDriver`), a Mapper (i.e `SentimentAmazonMapper`) and a Reducer (i.e `SentimentAmazonReducer`). The Driver and Reducer basically only have one method of interest again (the run method and the reduce method respectively). The Mapper in this program has an setup method (which is called once at the start of the task) and two parse methods for the wordlists in addition to the map method.

The Output Key-Value type is `<Text,DoubleWritable>` for Mapper and Reducer again. The Mapper checks for every word in a review if its either a good or a bad word. If its a negative word he puts for the article (the key) -1 as the value out. If its a positive word he puts for the article (the key) 1 as the value out.

The Reducer can then basically sum up the instances of negative words (if the input value is negative) and positive words (if the input value is positive) for a given article and calculate the sentiment.

Listings for the map/reduce methods are included to demonstrate the approach to the problem.

Listing 3: Excerpt from the map method (Sentiment) Code

```
for (String word : WORD_BOUNDARY.split(review)) {
        if (word.isEmpty()) {
                continue;
        }
        if (goodWords.contains(word)) {
                context.write(keyArt,one);
        }
        if (badWords.contains(word)) {
                context.write(keyArt,minus);
        }
}
```

Listing 4: Excerpt from the reduce method (Sentiment) Code

```
for (DoubleWritable instance : instances) {
        if(instance.get() > 0) {
                pos++;
        } else if(instance.get() < 0) {
                neg++;
        }
}
if(pos == neg) {
        context.write(word,new DoubleWritable(0));
} else {
        double a = pos + neg;
        double b = pos - neg;
        double sentiment = b / a;
        context.write(word,new DoubleWritable(sentiment));
}
```

Scaling In the case of these three datasets some sort of (small) scale out would be likely beneficial, since two of these datasets are above 64 MB, which means they are above the typical Block size. In the case of the Patio, Lawn and Garden Dataset a scale out will be not of a real benefit since it is only 14 MB big. Below we have also the Map/Reduce Invocations for the respective Datasets:

Table 4: Invocations of Map and Reduce Functions

Dataset	Map Invocations	Reduce Invocations
Grocery and Gourmet Food	151254	8713
Patio, Lawn and Garden	13272	962
Digital Music	64706	3568

(As expected) for every review there is a Map Invocation. In the Case of the Grocery and Gourmet Food Dataset there are a lot of Invocations (which means more I/O), therefore we can

expect a significant amount of speedup if we scale out properly. Overall there is one Map Task and one Reduce Task per Dataset. Below we also have the Time spent during the Map and Reduce phases and a line chart that demonstrates the relation between input size and time spent (for the map method only).

Table 5: Time (ms) spent in Map and Reduce Functions

Dataset	Map Time	Reduce Time
Grocery and Gourmet Food	102972	24785
Patio, Lawn and Garden	23287	18156
Digital Music	47479	27490

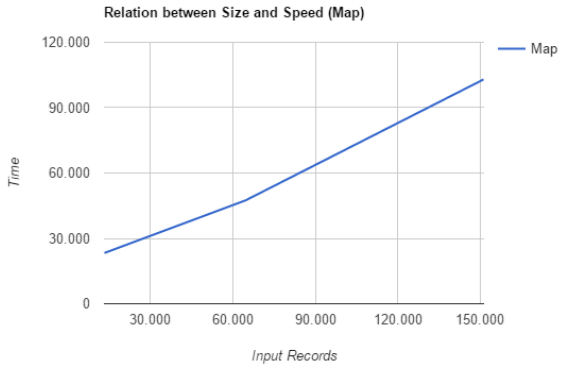

Figure 1: Relation between Input and Time

As expected bigger datasets result in a longer time spent. The relation between input size and time is more or less linear with the used datasets. The time spent in reduce is also apparently dependent on how many different reduce input groups exist.

Excerpt from the resulting sentiments In the following the first 10 results of the average sentiment (for every dataset) will be listed. The article column contains the `Amazon Standard Identification Number`. To get the percentages (negative/positive) the respective Sentiments have to be multiplied with 100.

Table 6: Sentiment for the Digital Music Dataset

Article	Sentiment
5555991584	0.52
B0000000ZW	0.56
B00000016T	0.57
B00000016W	0.55
B00000017R	0.57
B0000001P4	0.76
B0000002HZ	0.62
B0000002J9	0.69
B0000002JR	0.67
B0000002ME	0.74

Table 7: Sentiments for the Grocery and Gourmet Food Dataset

Article	Sentiment
616719923X	0.57
9742356831	0.61
B00004S1C5	0.44
B0000531B7	0.36
B00005344V	0.44
B0000537AF	0.51
B00005C2M2	0.28
B00006IUTN	0.49
B0000CCZYY	0.49
B0000CD06J	0.82

Table 8: Sentiments for the Patio, Lawn and Garden Dataset

Article	Sentiment
B00002N674	0.53
B00002N67P	-0.097
B00002N67Q	0.20
B00002N68C	0.29
B00002N6AN	0.27
B00002N8K3	0.13
B00002N8NR	0.59
B00004DTNH	0.54
B00004R9TK	0.43
B00004R9TL	0.41

2 MovieLens dataset analysis with Hive

Each subsection presents the query and resulting rows. Additionally the discussion regarding mapreduce tasks for each query is added. Finally we give a short discussion on the possible scale up on a larger cluster [2]

2.1 Query 1 - How many movies are there in total in the dataset?

```
SELECT COUNT(*)
FROM movies;
```

Result: 27278

As the movies data set is very small (2MB) only one mapper is used to select the data. The reducer counts all rows an finishes.

2.2 Query 2 - How many movies in the dataset belong to the "Film-Noir" genre?

```
SELECT COUNT(*)
FROM movies
WHERE array_contains(split(genres,'\\|'), 'Film-Noir');
```

Result: 330

This query is similar to the first one with an extra filter for the data rows. This filter can be implemented in the mapper and therefore again, only one mapper is used to select the data. The reducer counts all rows an finishes.

2.3 Query 3 - Which are the 10 most frequently assigned tags (by users, i.e., from the tags table)?

[2] This assignment was executed on a Cloudera Virtualbox VM with a single node.

```
SELECT tag_ranked
FROM
  (SELECT tag AS tag_ranked,
          count(tag) AS tag_count
   FROM tags
   GROUP BY tag
   ORDER BY tag_count DESC limit 10) r;
```

Result:

Tag
sci-fi
based on a book
atmospheric
comedy
action
surreal
BD-R
twist ending
funny
dystopia

This query has to be split into two tasks. The first mapper solves the inner SELECT statement, except the LIMIT statement. The limiting to 10 rows and selection of the single output column can be done in the second job.

2.4 Query 4 - Which 10 movies were the most controversial in 2015 (i.e., had the highest variance in ratings between 2015/01/01 and 2015/12/31)?

```
SELECT title
FROM
  (SELECT title,
          variance(rating) AS rating_var
   FROM movies
   LEFT JOIN ratings
       ON (movies.movieId = ratings.movieId)
   WHERE year(from_unixtime(ratings.timestamp)) = 2015
   GROUP BY title
   ORDER BY rating_var DESC LIMIT 10) t;
```

Result:

Title
Rent-a-Kid (1995)
Harder They Come, The (1973)
Docks of New York, The (1928)
A Cinderella Story: Once Upon a Song (2011)
Born in Flames (1983)
Elle: A Modern Cinderella Tale (2011)
Conspirators of Pleasure (Spiklenci slasti) (1996)
The War at Home (1979)
Jesse Stone: Sea Change (2007)
Cry_Wolf (a.k.a. Cry Wolf) (2005)

This query will be split into 3 tasks. At first the tables movies and ratings will be joined and filtered for ratings from 2015. This data set gets aggregated by title and variance of rating calculated. The last task is again sorting the data descending, limiting the output to 10 rows and selecting the title column exclusively.

2.5 Query 5 - Which movies (titles) are the 10 most frequently tagged and how often have they been tagged?

```
SELECT title,
       count(tag) AS tag_count
FROM movies
LEFT JOIN tags
    ON (movies.movieId = tags.movieId)
GROUP BY title
ORDER BY tag_count DESC LIMIT 10;
```

Result:

Title	Tag count
Pulp Fiction (1994)	1994
Fight Club (1999)	1779
Inception (2010)	1552
Matrix, The (1999)	1430
Shawshank Redemption, The (1994)	1339
Eternal Sunshine of the Spotless Mind (2004)	1240
Donnie Darko (2001)	1177
Memento (2000)	1168
Silence of the Lambs, The (1991)	1100
Avatar (2009)	995

The tables have to be joined in a first job. Afterwards the data can be grouped and the count field aggregated. After ordering the last job is to limit the output to 10 rows.

By discovering the GUI we have found the EXPLAIN option. According to this output the JOIN operation is performed by a local task. We assume this is because of the small size of the tags and the movies table - only few megabyte. This gave a little bit more insight on the CLI output.

2.6 Query 6 - Which 15 movies (titles) have been most frequently tagged with the label "mars"?

```
SELECT title
FROM
  (SELECT title,
          count(tag) AS tag_count
   FROM movies
   LEFT JOIN tags
       ON (movies.movieId = tags.movieId)
   WHERE tag = 'mars'
   GROUP BY title
   ORDER BY tag_count DESC LIMIT 15) t;
```

Result:

Title
Mars Attacks! (1996)
War of the Worlds, The (1953)
Total Recall (2012)
Total Recall (1990)
Capricorn One (1978)
Martian Child (2007)
It Came from Outer Space (1953)
Day the Earth Stood Still, The (1951)
Mission to Mars (2000)
6th Day, The (2000)
RocketMan (a.k.a. Rocket Man) (1997)
Destination Moon (1950)
Red Planet (2000)
Impostor (2002)
Doom (2005)

This query is very similar to query 5, except with an additional WHERE clause. This filter can be added to the reduction job of the JOIN operation, so no additional job is needed.

2.7 Query 7 - Which are the 10 best-rated movies (on average; list titles) with more than 1000 ratings?

```
SELECT title
FROM
   (SELECT title,
           t.rating_count
   FROM
      (SELECT title,
              avg(rating) AS rating_av,
              count(rating) AS rating_count
      FROM movies
      LEFT JOIN ratings
          ON (movies.movieId = ratings.movieId)
      GROUP BY title
      ORDER BY rating_av DESC) t
      ORDER BY t.rating_count desc) f LIMIT 10;
```

Result:

Title
Pulp Fiction (1994)
Forrest Gump (1994)
Shawshank Redemption, The (1994)
Silence of the Lambs, The (1991)
Jurassic Park (1993)
Star Wars: Episode IV - A New Hope (1977)
Braveheart (1995)
Terminator 2: Judgment Day (1991)
Matrix, The (1999)
Schindler's List (1993)

The ratings table is rather large (around 500MB), therefore the JOIN operation can be split over multiple mappers and reducers.

2.8 Query 8 - Which are the highest-rated "Film-Noir" movies with more than 10 ratings (average rating; movies with genre "Film-Noir", max. 10)?

```
SELECT title
FROM
   (SELECT title,
           avg(rating) AS rating_av,
           count(rating) AS rating_count
   FROM movies
   LEFT JOIN ratings
       ON (movies.movieId = ratings.movieId)
   WHERE array_contains(split(genres,'\\|'), 'Film-Noir')
   GROUP BY title
   ORDER BY rating_av DESC) t
WHERE rating_count > 10 LIMIT 10;
```

Result:

Title
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)
Third Man, The (1949)
Double Indemnity (1944)
Big Sleep, The (1946)
Chinatown (1974)
Notorious (1946)
M (1931)
Maltese Falcon, The (1941)
Touch of Evil (1958)
Strangers on a Train (1951)

The first task will be again the JOIN operation and filter for the genre 'Film-Noir'. The second task is the aggregation of average and count. Additionally the data will be filtered on the calculated count.Finally the output column title is selected and rows are limited.

2.9 Query 9 - What are the 15 most relevant genome tags for the movie "Toy Story (1995)" (movieId=1)?

```
SELECT tag
FROM
   (SELECT tag,
           g.relevance
  FROM
     (SELECT tagID,
             relevance
     FROM genome_scores
     WHERE movieId = 1) g
     LEFT JOIN genome_tags
          ON (genome_tags.tagId = g.tagId)
     ORDER BY  g.relevance DESC) f LIMIT 15;
```

Result:

Genome Tag
toys
computer animation
pixar animation
kids and family
animation
kids
pixar
children
cartoon
imdb top 250
animated
childhood
great movie
disney animated feature
friendship

By filtering the genome_scores table first in a subquery and join it afterwards, we could reduce the overall execution time by 25%. Hive reduces this query to a single job. But uses to mappers because of the large table genome_scores.

2.10 Query 10 - Which are the 10 most relevant movies for Vienna (i.e., with the highest genome tag relevance rating for the tag "vienna")?

```
SELECT title
FROM
   (SELECT movieId,
```

```
        g.tag,
        relevance
FROM
   (SELECT tagId,
        tag
    FROM genome_tags
    WHERE tag = 'vienna') g
    LEFT JOIN genome_scores
        ON (genome_scores.tagId = g.tagId)
    ORDER BY  relevance DESC) t
 LEFT JOIN movies
    ON (t.movieId = movies.movieId) LIMIT 15;
```

Result:

Title
Third Man, The (1949)
Johnny Guitar (1954)
Before Sunrise (1995)
Before Sunset (2004)
Before Midnight (2013)
Night Porter, The (Portiere di notte, Il) (1974)
Illusionist, The (2006)
Amadeus (1984)
Foreign Affair, A (1948)
Love in the Afternoon (1957)
Odessa File, The (1974)
Bad Timing: A Sensual Obsession (1980)
Best Offer, The (Migliore offerta, La) (2013)
Odd Man Out (1947)
Stranger, The (1946)

For this final query a join task with filtering for the tag 'vienna' will be generated first. The resulting rows are joint with the movies table. Finally in the third task the output column is selected an limited to 15 rows.

2.11 Scale Up

The effects of scale up should not be very significant because of the relatively small size of the data sets. Queries containing the genome-scores or ratings table can finish in the half time. But all queries in this assignment have jobs with only a single mapper or reducer, which would not benefit from a larger cluster.

Even for HDFS with a default chunk size (Cloudera) of only 128 MB is over sized for this tasks.

But Hive is intended for much larger datasets starting at the level of multiple gigabytes. Therefore the small numbers of mappers in this examples are comprehensible. A to fast increase of mappers for small dataset would result in too much mappers for large data sets producing a large overhead and poor performance of the whole system.