Group 10 Lab1 report

Implementation

1. **RDD-based Implementation**
   - First we filter out the entries in rdd which have columns less than 23(allNames Column) or having 23rd column empty.

```
-   val transform1=grdd.map(x=>x.split("\t")).filter(col=>col.size>23 && col(23)!="")
```

   - Then we build the date from first column using substring() function and split allNames column using delimiter semicolon.

```
-   val rdd2= transform1.map(col=>(col(1).substring(0, 4)+"-"+col(1).substring(4, 6) +"-"+
    col(1).substring(6, 8),col(23).split(";")))
```

   - We split the tuples formed in allNames using delimiter "," and select only the topic (discard the number) and mapping it with 1. Then we use reduceByKey() on date to get a mapping of date => all topics. There are two implementation in the code : one with "distinct" operation on topic in row and one without the distinct operation.

```
-   .mapValues(t=>t.map(s=>(s.split(",")(0),1)).distinct)
-   .reduceByKey((x,y)=>x ++ y)
```

   - Lastly we apply groupBy() operation on (topic,1) and aggregate sum of the matching topic. Then we apply sort in descending order after converting to array and then print the RDD.

```
-   .map(x =>
    {(x._1,x._2.groupBy(_._1).mapValues(_.map(_._2).sum).toArray.sortBy(t=>t._2).reverse.take(10))})
-   .collect().foreach(col=>println(col._1,col._2.mkString(" ")))
```

2. **Dataset approach**

   - Before any action is performed on the dataset, we needed to filter data in AllNames column such that only an array of terms remain, i.e we removed commas and numbers from each element of AllNames column.
     For this, we used a user defined function that replaces commas and numbers with an empty string.

```
val urlCleaner = (s:String) => {
     if (s == null) null else s.replaceAll(","," ").replaceAll("\\d+(?:[.,]\\d+)*\\s*", "")}
```

   - We then split the string at each row of the AllNames column on semicolon to obtain an Array(String). This array was then exploded using the explode function.

```
val a = filter2.withColumn("AllNames",split(col("AllNames"),";").cast("array<String>"))
val b = a.withColumn("AllNames",explode($"AllNames"))
```

   - Gdate was also cast to DateType to extract dates form the timestamp.

```
val newDS = c.withColumn("Gdate", c("Gdate").cast(DateType))
```

   - Now, we have final filtered dataset with <Gdate: timestamp,AllNames: String > as a row. We call this dataset newDS
   - Finally, we groupyBy "Gdate" and "AllNames" with aggregation as count on "AllNames". We the rank the outcome to obtain top 10 counts and terms of each date present in the dataset.

```
val counts = newDS.groupBy($"Gdate",$"AllNames").agg(count($"AllNames").as("count"))

counts.withColumn("rank",rank().over(Window.partitionBy("Gdate").orderBy($"count".desc)))
      .filter($"rank" <= 10)
      .drop("rank")
      .show()
```

Questions

1. In typical use, what kind of operation would be more expensive, a narrow dependency or a wide dependency? Why?

- Operations involving wide dependencies are more expensive.
- This is because there exists a one-to-many relationship between the partitions in parent RDD to the partitions in child RDD when considering wide dependency.
- Therefore, transformations involving wide dependencies are slower as they require the data to shuffled over the network.

2. What is the shuffle operation and why is it such an important topic in Spark optimization?

- Data is distributed over the network. This data has to be moved from one node to another whenever a “group” operation (or mostly operations with wide dependency) is performed.
  This process is called shuffling.
- For example, groupByKey() method involves all the distributed key-value pairs with the same key to be grouped together on a same machine. This causes shuffling.
- If we try to prevent shuffling from happening until it’s totally necessary, we can significantly reduce the running time of our program. Hence, shuffling is an important topic in spark optimization.

3. In what way can Dataframes and Datasets improve performance both in compute, but also in the distributing of data compared to RDDs? Under what conditions will Dataframes perform better than RDDs?

- In Dataframes and Datasets, optimizations are done using catalyst optimizer, which provides both rule based as well as cost based optimizations. This increases the performance of queries that developers write. On the contrary, no such optimizations are done with a RDD implementation.
- When it comes to distribution of data, spark dataframe uses off heap memory for serialization. byte code is generated dynamically and there is no need for deserialization when considering small operations. On the other hand, while distributing data into the network, RDD uses Java serialization. This serialization of Java and scala objects incurs an overhead as both structure and data is shuffled between the nodes.
- Dataframes will perform better than RDDs when we are dealing with structured data and when our processing requires filters, maps, aggregations, attribute access on semi-structured data.

4. Consider the following scenario. You are running a Spark program on a big data cluster with 10 worker nodes and a single master node. One of the worker nodes fails. In what way does Spark’s programming model help you recover the lost work?

- Spark maintains a DAG that keeps track of all the transformations that have been done on the database.
- As it is a lazy evaluation, only transformations have been saved in DAG. Therefore, whenever a node fails, the DAG is traversed till the previous node to the node which has failed is reached. The next state is then re-computed based on the execution plan of previous node.

5. We might distinguish the following five conceptual parallel programming

- **models:**
- **1. farmer/worker**
- **2. divide and conquer**
- **3. data parallelism**
- **4. function parallelism**
- **5. bulk-synchronous**

Pick one of these models and explain why it does or does not fit Spark's programming model. (max. 100 words)

Data parallelism

- Data is placed into different partitions within the cluster across same set of machines.
- A driver program is present within the Spark cluster where application logic is stored.
- This driver program directs worker nodes to start processing on the corresponding partition of data.
- All this while, each node processes data in parallel and transformations will keep happening in the same partitions before there is an actual need of shuffling.
- For example, when a reduceByKey() operation is performed on a rdd with <String,Int> pairs, summation on Integer values of the same keys present on the same partition is performed first. Only after this the data is shuffled and is brought to a same node and final grouping is done.

Implementation analysis questions.

1. **Measure the execution times for 10, 20, 50, 100 and 150 segments. Do you observe a difference in execution time between your Dataframe and RDD implementations? Is this what you expect and why? (max. 50words)**

⇨

Segments	RDD Implementation (in sec)	DF/DS Implementation (in sec)
10	13.110	11.896
20	21.620	17.481
50	37.872	21.938
100	97.659	101.010
150	143.762	138.117

As we can see from the observations above, the RDD implementation and Dataframe Implementation don't have much difference in terms of execution time for small number of segments. We expect DF implementation to be faster since it is compiled into execution plan and then executed. Spark can optimize the execution plan to reduce execution time.

2. **How will your application scale when increasing the amount of analyzed segments? What do you expect the progression in execution time will be for, 100,1000,10000 segments? (max. 50words)**

⇨ The increment in segments would lead to increment in execution time. From the table in answer of question 1 we can observe the progression from 10 segments to 100 segments and comment that the progression is exponential. We can expect a similar exponential progression in execution time for segments 100,1000,10000.

3. **If you extrapolate the scaling behavior on your machine, using your results from question1, to the entire dataset, how much time will it take to process the entire dataset? Is this extrapolation reasonable for a single machine? (max. 50words)**

⇨ The GDELT dataset has 157378 segments (Sept 2019). For a reasonable extrapolation we apply linear regression to estimate how much time it would take to process the entire dataset on my laptop. The approximate time we got after applying linear regression is 150,000 seconds ~ 41.66 hours.

4. **Now suppose you had a cluster of identical machines with that you performed the analysis on. How many machines do you think you would need to process the entire dataset in under an hour? Do you think this is a valid extrapolation? (max. 50words)**

⇨ Based upon the approximate calculation done in question 3, and considering overhead of data sharing and collecting, It would take approximately 45 machines in a cluster to perform analysis of entire dataset. I think this gives a fair estimate on how the analysis would perform on cluster.

5. **Suppose you would run this analysis for a company. What do you think would be an appropriate way to measure the performance? Would it be the time it takes to execute? The amount of money it takes to perform the analysis on the cluster? A combination of these two, or something else? Pick something you think would be an interesting metric, as this is the metric you will be optimizing in the 2nd lab! (max. 100words)**

⇨ This would highly depend on the business requirement of the company/project. Many companies would value lower latency over money spent on adding more clusters whereas for other companies execution time could be a trivial thing but managing finances and allocating just as much resources as required could be a priority.

A better alternative to measure performance should be a combination of both of these for example execution time per certain number of machines. Such a metric could give an idea to introspect if there's a need to optimize the code or increase hardware capabilities.