I: Implementation outline

Task:

Output a structure that maps dates to a list of 10 tuples containing the most mentioned topics and the amount of times they were mentioned.

Implementation:

DataFrame-Based Implementation:

1. Create a class that could be mapped to the date and AllNames columns of the schema
2. Read CSV files, using tabs as a delimiter and implicitly mapping to the class defined
3. Create a temporary View on which to perform SQL operations
4. Write SQL queries to do the following:
   a. Select dates and AllNames which are formatted using user defined functions which parse each date into yyyy-MM-dd format and convert each AllNames string into separate topics (for example, `"Amsterdam,123;Deft,456"` gets converted to `[Amsterdam, Delft])` respectively. These are then split into individual rows using the built-in `explode` function.
   b. Count the number of times each topic appears, and group by date, filtering out topics which do not contain “ParentCategory” as it is a false positive.
   c. Use a partitioned function, `dense_rank`, to assign ranking numbers to each of the counts within their date grouping. We can then filter out all rank numbers over 10 to get the top 10 most common results.
   d. Group the count and topic columns together into a tuple and use `collect_list` to pivot the top 10 results into a single column so that they can be displayed under the same date key in the final object.
5. Convert the results to a JSON object and display

RDD-Based Implementation:

1. Read all segments
2. Split into columns on tabs
3. Filter rows which don’t have an AllNames value
4. Extract the date column formatted as yyyyMMdd and split topics in the AllNames column into individual topics
5. Count the number of occurrences of each topic, split data into tuples of (date, (topics, count)) and group by distinct date
6. Filter out topics containing the phrase “ParentCategory”, sort by maximum mentions of a topic, and display the top 10 topics with the highest frequency.

II: Questions

1. **In typical use, what kind of operation would be more expensive, a narrow dependency or a wide dependency? Why? (max. 50 words)**

   Narrow dependencies only operate on single rows at a time (such as mapping), as such they can be done within a single partition. Wide dependencies (such as grouping) require data to be shuffled around, which is expensive and requires communication between partitions. This makes wide dependencies slower and more expensive.

2. **What is the shuffle operation and why is it such an important topic in Spark optimization? (max. 100 words)**

Shuffle operations move data around, such as to group data together (e.g. grouping operations). Doing this often requires communication between different partitions, which can be on processes or even on different machines altogether. This means that there can be significant slow down due to communication and network latency. Optimal performance in spark would be achieved without having to move any data around during computation.

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? (max. 100 words)**

Dataframes may store data column-by-column instead of row-by-row. Many computations require reading of only a select number of columns, these can now be stored more closely together which can reduce the amount of communication between partitions required to access the relevant data, as well as reducing the number of different sectors that need to be read to access the data. Therefore dataframes may perform better when working with large amounts of data of which only a select number of fields is relevant to the current computation.

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? (Think about the execution plan or DAG) (max. 50 words)**

If any data on a worker node is lost, the data can be recovered be simply computing the operations from the original dataset using the lineage of operations. Since Spark RDDs are immutable, every dependency between the RDDs is documented in the lineage graph created in the DAG.

5. **We might distinguish the following five conceptual parallel programming models:**
    i. **farmer/worker**
    ii. **divide and conquer**
    iii. **data parallelism**
    iv. **function parallelism**
    v. **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 consists of one machine with shared memory. The dataset is split into subsets, and a "worker" is assigned to each dataset to perform operations on it. There are multiple independent workers processing their respective datasets parallelly, and the individual results are combined to obtain a final product.

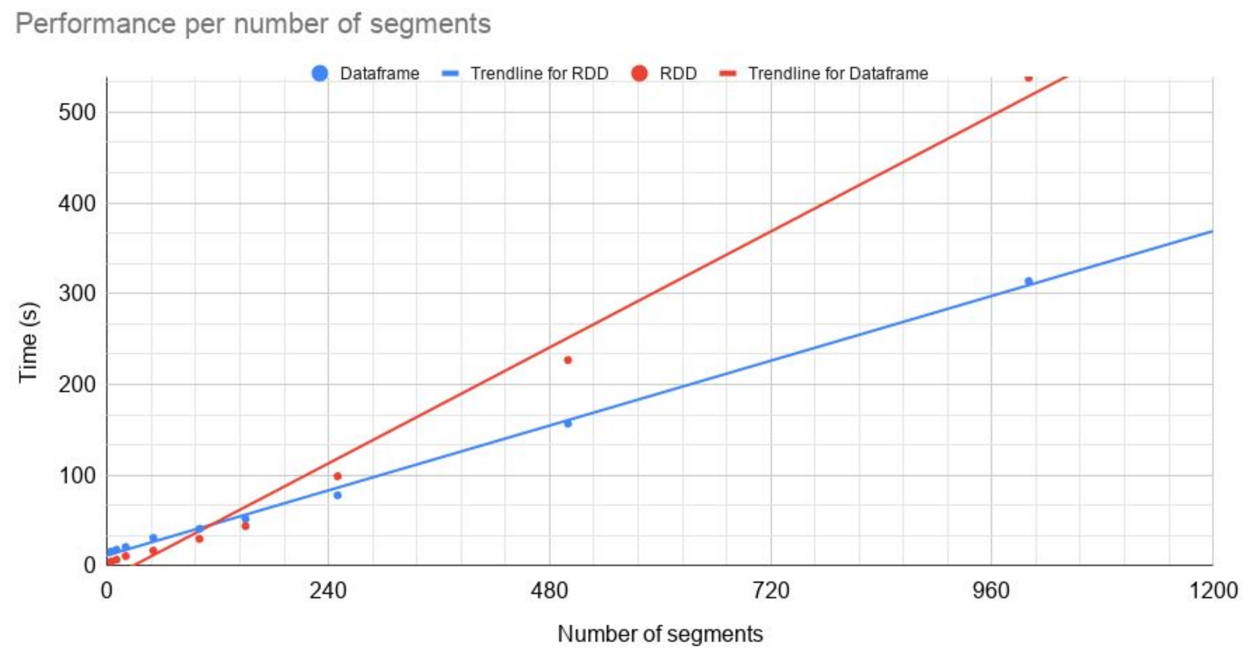
Spark uses distributed data parallelism, ie, data is distributed over multiple independent nodes, instead of a single machine. Each individual node works on the data subsets in parallel, and integrates the results when done. The disadvantage here is there is a latency if the nodes have to communicate with each other.

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. 50 words)**

It seems that RDDs are a lot faster on small amounts of data. However, the dataframe implementation scales a lot better and surpasses the RDD implementation around 200 segments. This is a greater difference than we expected, as we thought there would be only minor performance differences. Results:

Segments	Dataframe	RDD
4	16s	5s
10	18s	7s
20	21s	11s
50	31s	17s
100	41s	30s
150	52s	44s
250	78s	99s
500	157s	227s
1000	314s	538s



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. 50 words)**

We expect that the dataframes will do increasingly well in comparison to the RDDs. As the dataset grows larger than the available RAM, the greater data locality of the dataframes will provide increasing performance benefits. Expected execution time for 10000 segments: 50 minutes for dataframes, 90 minutes for RDDs.

3. **If you extrapolate the scaling behavior on your machine, using your results from question 1, to the entire dataset, how much time will it take to process the entire dataset? Is this extrapolation reasonable for a single machine? (max. 50 words)**

The expected time to process the entire dataset will be 11 hours (20 hours for RDDs)*. However, it is unlikely one machine could manage this performance as a single column won't fit in the RAM, so many much slower disk reads and writes will have to be performed.

* Based on 4.2 TB / 32 MB = apx. 130 000 segments

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. 50 words)**

Twelve machines would be required. Adding machines adds additional overhead in coordination and communication (especially during shuffles), so the required required number will be a bit higher. This extrapolation may be inaccurate as we only tested a single machine on a small bit of data, so real-world performance may differ.

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. 100 words)**

While execution time is important, it often won't matter whether it would take seconds or hours to run the computation. The duration will also be heavily dependant on the resources assigned to the job, meaning execution time depends on the monetary budget. The most useful metric will be the total cost of computation, as it will directly influence how often such a computation could be run in comparison to the value created by the result of the computation. e.g. If potential profits gained from the result are X, a total running cost of X/2 may be justifiable.