Lab 3

Section C

C.1 What is an RDD

What is RDD (Explain the Resilient and Distributed)? How does Spark work with RDD?

An RDD is a Resilient Distributed Dataset. It describes how Spark stores and operates on data. Instead of holding a single large object in the filesystem or in memory, the data is split into many partitions, each one held in memory on at least one machine across a potentially large cluster of connected machines. Operations are still performed on the data as a single object, from the view of the user, but internally the operations are distributed across these partitions.

Operations on an RDD are also lazy. When an operation on an RDD is specified, this operations is not necessarily immediately executed, instead, it is used to build an execution graph in the background. Only certain operations will actually start execution of this graph, like .collect(). This specification is used to make RDD's resilient, even though they can be distributed across many machines: when a partition becomes unavailable for any reason, the execution graph still exists, and so the operations that have been executed to generate the contents of this partition can be performed again, regenerating the missing data.

C.2 Your colleague is trying to understand her Spark code

Your colleague is trying to understand her Spark code by adding a print statement inside her split_line(..) function, as shown in this code snippet:

```
def split_line(line):
  print('splitting line...')
  return line.split(' ')

lines = spark_context.textFile("hdfs://host:9000/i-have-a-dream.txt")
print(lines.flatMap(split_line).take(10))
```

When she runs this code in her notebook, she sees the following output: ['I', 'am', 'happy', 'to', 'join', 'with', 'you', 'today', 'in', 'what'] But, she doesn't see the "splitting line..." output in her notebook. Why not?

Operations on an RDD are performed by a worker (or multiple workers) in the Spark cluster. The print function writes something to stdout on those machines, which is not specified to be stored as part of an RDD, so the results of this print statement are not visible on the client.

C.3 Running out of memory by calling .collect()

Calling .collect() on a large dataset may cause driver application to run out of memory. Explain why.

The .collect() function gathers the contents from all partitions of an RDD into a single object. Partitions are usually around 128MB in size, which (by itself) easily fits on most machines, but an RDD may consist of an arbitrary number of these partitions. So each machine in this cluster has enough resources to store a fraction of an RDD, but maybe not enough to store the whole RDD (also applies to the client), which is one of the reasons why a cluster like this would be used in the first place.

C.4 Partitions mutable or immutable

Are partitions mutable or immutable? Why is this advantageous?

A partition is immutable. This is an advantage on at least two fronts: scalability (1) and resilience (2).

1) All partitions are immutable, so there is no need to differentiate between operations that could/should operate on an existing partition vs. those that might require a copy to leave the original data intact, which greatly simplifies scheduling operations across a large cluster. This also means that operations, data and workers are not strictly tied together, data and operations they can be moved across the cluster relatively freely. 2) Mutable partitions would be quite bad for resilience. A mutable partition may, at any point, be in a state where it had a copy of some original data, and then performed a very large number of transformations on this data. If the machine retaining this partition is then lost, restoring this partition requires running all those transformations again, and to copy the original data as well, which itself may have been modified/overwritten, propagating this requirement to recalculate. Since an RDD is immutable though, the data exists in the cluster until it truly is not needed anymore, e.g. when the final result is calculated. When any partition is lost, the data used to generate it directly is still present in the cluster, so there is only little computation to replicate (combined with the lazy evaluation, a lost partition may not even require replication if its result has already been used for its intended purpose).

C.5 Difference between DataFrame and RDD

What is the difference of DataFrame and RDD? Explain their advantages/disadvantages

The difference between Spark Dataframe's and Spark RDD's is rather slim, since Spark Dataframes are built on top of Spark RDD's. The main difference is about the kind of data they can store, where a dataframe is very structured and an RDD can be arbitrarily structured (structured, semi-structured, unstructured).

Dataframes have a well defined schema, so when an operation on a dataframe is scheduled, the spark framework knows about the layout of the data inside the RDD, enabling better performance. Even if a blank RDD contains structured data, spark has no fundamental knowledge of this structure and cannot perform the same optimizations it can when operating on a spark dataframe. These optimizations come with the disadvantage that there is specific list of legal operations on a dataframe, while a blank RDD can be operated on by any function written in some driver language (e.g. python or scala).

Additionally, any structured vs unstructured data argument also applies here. Dataframes may allow additional optimizations by the spark framework that are not possible with an arbitrary RDD, but not all data fits a dataframe naturally. Highly heterogenous data or data with dynamic structure may not fit a dataframe easily, so even it could be made to fit into a dataframe, the performance on this dataset might end up effectively worse than on a regular RDD.

Section D

"A colleague has mentioned her Spark application has poor performance, what is your advice?" Provide at least 4 clear recommendations, answer in full sentences. 3 or 4 sentences are suggested for each recommendation. Marks will be awarded based on the correct use of Spark terminology. Do not submit code for this question (but you might want to mention API methods, for example). You do not need to include references, but if you feel this supports your answer, please include a short, well-formatted bibliography.

Without any additional information about the application or the code, here is some general advice:

1) *Suboptimal implementation.* The application may not perform well because the code performs redundant operations frequently. In the driver code, operations may calculate results implicitely multiple times, so that they cannot be cached and have to be recomputed frequently. Data formats may also not be chosen well, like operating on strings instead of dedicated data types.

2) *The driver code relies heavily on client side operations.* A spark application performs operations on an RDD by distributing it across a cluster of machines, each worker performing the operation on a small fraction of the RDD (a small number of partitions). A spark application with poor performance may .collect() data into the driver application where operations are run locally instead of on the cluster. This wastes performance potential by moving all data across the network (instead of operating locally on a worker), and by not utilizing the compute power of the distributed cluster.

3) *Bad driver configuration.* Low spark application performance may be caused by improper configuration of the driver application, like the following: The number of executors may be much lower than the number of workers in the cluster, underutilizing resources. The number of executor cores may be too low (wasting resources), or too high (making the jobs compete for resources on the same machine, also wasting resources). The executor memory may also be set too high, degrading performance (caused by moving memory in and out of swap space).

4) *Overreliance on file system operations.* Operating on a very large dataset may imply that data needs to be stored on disk because it does not fit into the memory of the cluster. Because the data is stored across a large cluster it could fit into the memory of the cluster without any issues though. Storing data to disk, and restoring it at later points, has substantially worse performance than keeping it in memory and operating on it directly, or even just moving the data across the network between memory of different workers.