UPPSALA UNIVERSITET

Data Engineering I - Assignment 3

5 mars 2021

Section C - Concepts in Apache Spark and Distributed Computing

C.1 Why do we use the Map-Reduce programming model?

Generally the MapReduce programming model works well for large-scale data-intensive applications in clusters. This is because it has certain nice properties, e.g. parallel operations, enables programs to be scaled upwards, can handle very large data sets and provides fault tolerance. Spark uses MapReduce as its foundation but with some changes to increase performance and ease of use.

C.2 Anna Exampleson is trying to understand her Spark code by adding a print statement inside her split_line(..) function, as shown in this code snippet. But, she doesn't see the "splitting line..." output in her notebook. Why not?

The code is running on the cluster meaning that workers are running this function when it is called. The print statement printing out "splitting line..." is therefore printed on the stdout on the workers and not in the driver.

C.3 "Calling .collect() on a large dataset can cause my driver application to run out of memory" Explain why.

When performing the .collect() action on a RDD all the elements on the RDD are fetched to the driver node (the node hosting the user's Spark application and not part of the cluster). This can result in the driver running out of memory if the data set is large enough

C.4 Are partitions mutable or immutable? Why is this advantageous?

Partitions in Spark are immutable, meaning read-only. This design choice is advantageous because it provides efficient data reuse and enables Spark's abilities of caching- sharing- and replicating data without risk of any problems.

C.5 In what sense are RDDs 'resilient'? How is this achieved?

This refers to the property that RDD's are fault-tolerant. This is possible because RDD's consist of multiple smaller partitions. Since the RDD's are immutable and all potential transformations are logged, a lost partition can simply be recomputed.

Section D - Essay Questions

"A colleague has mentioned her Spark application has poor performance, what is your advice?" List 4 clear recommendations, answer in full sentences.

Recommendation 1 - Memory Management in Driver: If my colleague is performing any rdd.collect() operations, Spark will try to move all data in the RDD to the driver machine which risks slowing things down or even to crash the application. Instead I would advice to use the take()-function to just look at a small part of the data.

Recommendation 2 - Incorrect configuration: Make sure Spark is configured to meet the applications needs. Review the defined number of max executor cores and the memory per executor.

Recommendation 3 - Dataframes: If my colleague is performing simple operations on RDD's I would advise to instead work with DataFrames. RDD's are slower to perform simple grouping and aggregation operations compared to DataFrames.

Recommendation 4 - groupByKey: If my colleague is grouping to perform an aggregation (e.g. sum or average) over each key and using groupByKey, using reduceByKey/aggregateByKey will give faster results. This is because groupByKey will shuffle more data than reduceByKey/aggregateByKey and shuffle is an expensive operation.