UPPSALA
UNIVERSITET

Assignment 3: Spark

Junjie Chu

Part C

C.1

When dealing with massive data on a single machine, we cannot complete the task effectively because of the limitation of hardware resources.

Once the stand-alone program is extended to cluster for distributed operation, the complexity and development difficulty of the program will be greatly increased.

With the introduction of MapReduce programming models, developers can focus most of their work on the development of programming logic, while the complexity of distributed computing is handled by the framework.

C.2

When running the application, the ‘cluster manager’ assigns it to some of the ‘Worker’ nodes and the RDDs will be processed on different ‘Worker’ nodes. The ‘print’ operations of RDDs and their outputs are also on different “Worker” nodes, only the return values will be transmitted to her notebook on the ‘Driver’, so she cannot see ‘splitting line…’ in her notebook. If she wants to see them, she could use ‘collect()’ to download the data from the clusters to the driver and then run ‘print’ or ‘split_line’.

C.3

The ‘collect()’ will convert ‘RDD’ into an array and collect all data from the remote cluster to the driver. The data are stored in arrays, which occupy the memory of the ‘Driver’. When the dataset is very large, the ‘Driver’ will run out of memory.

C.4

Partitions are immutable. In distributed systems, immutable partitions can be shared safely across various processes and threads. They allow us to easily recreate the RDDs and we can enhance the computation process by caching them.

UPPSALA
UNIVERSITET

C.5

RDDs are resilient because RDDs are immutable (can't be modified once created) and fault tolerant.
Spark RDDs are fault tolerant as they track data lineage information to rebuild lost data automatically on failure. They rebuild lost data on failure using lineage, each RDD remembers how it was created from other datasets (by transformations like a map, join or groupBy) to recreate itself.

Part D

Recommendation 1:

Serialization plays an important role in the performance for any distributed application. By default, Spark uses Java serializer. Spark can also use another serializer called 'Kryo' serializer for better performance. Kryo serializer is in compact binary format and offers processing 10x faster than Java serializer.

Recommendation 2:

Spark introduced three types of API to work upon – RDD, DataFrame, DataSet. RDD is used for low level operation with less optimization. DataFrame is best choice in most cases due to its catalyst optimizer and low garbage collection (GC) overhead. Dataset is highly type safe and use encoders. It uses Tungsten for serialization in binary format.

Recommendation 3:

Broadcasting plays an important role while tuning your spark job. Broadcast variable will make your small data set available on each node, and that node and data will be treated locally for the process. Suppose you have a situation where one data set is very small and another data set is quite large, and you want to perform the join operation between these two. In that case, we should go for the broadcast join so that the small data set can fit into your broadcast variable.

Recommendation 4:

Spark provides its own caching mechanism like Persist and Caching. Persist and Cache mechanisms will store the data set into the memory whenever there is requirement, where you have a small data set and that data set is being used multiple times in your program.

Recommendation 5:

ByKey operations generate lot of shuffle. Shuffles are heavy operation because they consume a lot of memory. Instead of groupBy, a user

UPPSALA
UNIVERSITET

should go for the reduceByKey because groupByKey creates a lot of shuffling which hampers the performance, while reduceByKey does not shuffle the data as much. Whenever any ByKey operation is used, the user should partition the data correctly.

Recommendation 6:

Spark comes with many file formats like CSV, JSON, XML, PARQUET, ORC, AVRO and more. A Spark job can be optimized by choosing the parquet file with snappy compression.

Recommendation 7:

JVM garbage collection can be a problem when you have large collection of unused objects.The first step in GC tuning is to collect statistics by choosing – verbose while submitting spark jobs. In an ideal situation we try to keep GC overheads < 10% of heap memory.

Recommendation 8:

Parallelism plays a very important role while tuning spark jobs. Every partition or task requires a single core for processing. There are two ways to maintain the parallelism: ‘Repartition’ and ‘Coalesce’. It is not advisable to go for Repartition when you want to lash all the data. Coalesce will generally reduce the number of partitions and creates less shuffling of data.