Uppsala University

Assignment 3

Author:
Jinglin Gao

February 23, 2023

1 Introduction

For questions in section C and section D, the answers are discussed here.

2 Section C

2.a

Question c.1 What is RDD? How does Spark work with RDD?

RDD is the abbreviation of Resilient Distributed Dataset, it is the fundamental data structure of spark. They are a distributed collection of objects, which are stored in memory or on disks of different machines of a cluster. A single RDD can be divided into multiple logical partitions so that these partitions can be stored and processed on different machines of a cluster.
Spark exposes RDDs through a functional programming API in Scala, Java, Python, and R. First users create an RDD, RDDs can be created from a variety of data sources, including HDFS, local file systems, and so on. Second, transform an RDD. Once one has an RDD, one can transform it by applying one or more operations to its elements. Then persisting an RDD, RDDs are lazily evaluated, which means that Spark will only compute the elements of an RDD when necessary. If one wants to reuse an RDD multiple times or save it to disk for later use, one can persist it in memory or on disk. Last, to perform an action, an action is an operation that triggers the computation of an RDD and returns a result to the driver program or writes it to an external storage system.

2.b

Question c.2

The reason is that the 'print' statement inside the 'split_line' function is executed on the worker nodes in the Spark cluster, not on the driver node where the notebook is running. The worker nodes execute the code in parallel on the data partitions, and the output is not returned to the driver node until the 'flatMap' operation is complete and the 'take' action is called.

2.c

Question c.3

Because the collect action will try to move all data in RDD/DataFrame to the machine with the driver and where it may run out of memory and crash.

2.d

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

In Spark the partitions are immutable. It is advantageous because it enables Spark to perform distributed processing in a fault-tolerant manner. Since partitions are immutable, Spark can easily recover from node failures or job crashes by simply recomputing the lost partitions. Another advantage is that it enables Spark to perform lazy evaluation of transformations since it just needs to compute the partitions that are required to fulfill the requested output.

2.e

Question c.5 What is the difference between DataFrame and RDD? Explain their advantages/shortages.

The data in RDD is unstructured, the models are a set of values, or set of key/value pairs while the data in DataFrames are structured, so the models are tables. For example, the data in RDD usually are text, XML, and HTML and the data in DataFrames usually are CSV or any record-based file format. Also, RDD uses SparkContext as API while DataFrame use SparkSession as API.
With RDD it can easily and efficiently process data that is structured as well as unstructured, but it does not infer the schema of the ingested data and requires the user to specify it. However DataFrame works only on structured and semi-structured data, it organizes the data in the named column. DataFrames allow Spark to manage the schema.
Another advantage of RDD is that it contains a collection of records that are partitioned. Each partition is one logical division of data that is immutable and created through some transformation on existing partitions. Immutability helps to achieve consistency in computations.

3 Section D

There are some suggestions to improve the Spark application performance.

Optimize data partitioning: If the data is not partitioned optimally, it could cause significant performance issues. It is crucial to ensure that the data is partitioned correctly across the cluster. One way to optimize data partitioning is by using the repartition() method to ensure that the data is partitioned uniformly and increases the level of parallelism in the application. Additionally, using coalesce() can be used to reduce the number of partitions without shuffling.
Do not use collect() method: When your RDD is large and a collect operation is issued on this RDD, the dataset is copied to the driver, and a memory exception will be thrown if the dataset is too large to fit in memory.
Use efficient transformations and actions: Choosing the right transformations and actions can have a significant impact on the performance of a Spark application. It is essential to use transformations that are suitable for the use case and to minimize the use of expensive operations.
Tune the Spark configuration: Spark provides a wide range of configuration parameters that can be tuned to optimize the application's performance. Some critical configuration parameters to consider include the number of executors, the amount of memory allocated to the driver and executors, and the number of cores per executor. Tuning these parameters can significantly improve the application's performance.
In summary, to improve the performance of a Spark application, it is important to optimize data partitioning, avoid memory exceptions, choose efficient transformations and actions, and tune the Spark configuration.