**C.1 What is RDD (Explain the Resilient and Distributed)? How does Spark work with RDD?**

RDD (Resilient Distributed Datasets) is an abstract concept of distributed memory. RDD provides a highly restricted shared memory model, that is, RDD is a set of read-only record partitions, which can only be performed by performing deterministic transformations in other RDDs. **The resilience** is shown in following ways: memory resilience (automatic switching between memory and disk), fault-tolerant resilience (data loss can be automatically recovered), computational resilience (calculation error retry mechanism), sharding resilience (resharding according to need). **The distribution** is shown in storage and computation, the data is distributed on multiple nodes by partition, so that it can be calculated in parallel on multiple nodes, and then merge the calculation results.

RDD generally represents the data stored on file system such HDFS or Hive. Spark will read these files and divide them into different RDD, then let different nodes apply computation on them.

**C.2**

The print() works in this split\_line function, but the outcome goes to the stdout /stderr on the computer that is running a spark executor. Therefore, we can only see this when we find the logs. Or we also can choose to use YARN to print this out.

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

The collect operation in spark is to transmit remote data to the local through the network. It will collect the data of the RDD and return it to the Driver in the form of an array. The default memory size on the Driver side is 1G. If the data volume of an RDD exceeds the default 1G memory on the Driver side, and the collect operation is called on the RDD, there will be an out-of-memory (OOM) on the Driver side, so all this collect operation has certain risks.

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

Partitions are immutable. Because RDD is immutable once it is created, and RDD is consist of different partitions.

Immutability rules out a lot of potential problems due to updates (read and write) from multiple threads at the same time. Immutable data is safe to share across processes. Immutability also makes the recreation of RDD at any time it’s needed becomes possible. Immutable data is easy to be stored in memory that the operations on these data can be moved from disk to memory, which relieve stress of limited resources.

**C.5 What is the difference of DataFrame and RDD? Explain their advantages and disadvantages**

**Difference:** RDD can store any single-machine type of data, but it does not know the specific structure of the stored-data, and it does not support Spark SQL operation. While DataFrame provides detailed structural information, so that Spark SQL can clearly know which columns are included in the data set, and what is the name and type of each column. RDD is a distributed collection of Java objects. A DataFrame is a distributed collection of Row objects. In addition to providing richer operations than RDDs, DataFrame has more important features to improve execution efficiency, reduce data reading, and optimize execution plans, such as filter pushdown and clipping. In short, the DataFrame related interface is an extension of RDD, which allows RDD to understand which columns the stored data contains and can perform operations on the columns.

**Advantages and Disadvantages:** The RDD API is functional, emphasizing immutability, and tends to create new objects rather than modify old ones in most scenarios. Although this feature brings a clean and tidy API, it also makes Spark applications tend to create many temporary objects during runtime, which puts pressure on the GC. Based on the existing RDD API, we can use the mapPartitions method to overload the data creation method in a single RDD shard and use the method of reusing variable objects to reduce the overhead of object allocation and GC, but this sacrifices the readability of the code. On the other hand, Spark SQL has tried to reuse objects as much as possible within the framework. Although this will break the immutability internally, when the data is returned to the user, it will be converted into immutable data again. Develop with the DataFrame API and utilize these optimizations.

**D “A colleague has mentioned her Spark application has poor performance, what is your advice?”List 4 clear recommendations, answer in full sentences. I suggest 2 or 3 sentences for each recommendation.**

**optimal resource allocation:** It is to allocate more resources for tasks. Within a certain range, the allocation of increased resources is proportional to the improvement of performance, and the optimal allocation of resources is achieved. For Spark Standalone mode, before submitting a task, we can know the resources we can use. When writing the submit script, resources are allocated according to the available resources. For Spark Yarn mode, since Yarn uses resource queues for resource allocation and scheduling. When writing submit scripts, resources are allocated according to the resource queues to which Spark jobs are submitted.

**RDD optimization:** When performing operators on RDDs, it is necessary to avoid repeated calculations on RDDs under the same operator and calculation logic. In Spark, when the operator is performed on the same RDD multiple times, the RDD will be recalculated with the previous parent RDD each time. This situation must be avoided. It is a huge waste of resources. Therefore, it is necessary to persist RDDs that are used many times and cache the data of public RDDs in memory/disk through persistence. After that, the calculation of public RDDs will directly obtain RDD data from memory/disk.

**Parallelism adjustment:** The degree of parallelism in a Spark job refers to the number of tasks in each stage. If the parallelism setting is unreasonable and the parallelism is too low, it will lead to a great waste of resources. The ideal parallelism setting should match the parallelism with the resources. In short, the parallelism should be set as large as possible under the premise of resources permitting, so that the cluster resources can be fully utilized.

**Adjust the localization wait time:** During the running process of the Spark job, the Driver will assign the task of each stage, but the task may not be assigned to the node where the data it processes is located. Data transfer occurs when the data to be processed by the task is not on the node where the task is located. A large number of network transmissions will seriously affect performance. Therefore, we hope to adjust the localized waiting time. If the target node processes part of the task during the waiting time, the current task will have the opportunity to be executed. Improve overall performance of Spark jobs.