UPPSALA
UNIVERSITET

Large Datasets for Scientific Applications

Assignment 2

Hamza Imran Saeed

May 17, 2019

Section C - Concepts in Apache Spark and Distributed Computing

D.1 Why do we use the Map-Reduce paradigm?

We use the Map Reduce Paradigm because it gives us the the ability to parallelize computations and gives us a solution to process large amounts of data in a distributed environment making it highly scalable. Using Map Reduce, large volumes and varieties of data can be mapped to key value pairs and reduced into smaller sets of data through aggregation operations. Frameworks such as Spark give users the ability to use Map Reduce to run models over large distributed sets of data and different ML and statistical techniques to find patterns, make prediction and do other complex computations.

D.2 Anna Exampleson is trying to understand her Spark code by adding a print statement inside her split-line(..) function. When she runs this code, she doesn't see the "splitting line..." output in her notebook. Why not?

She doesn't see the print statement that she has inside the split function because the split function is actually executed on each of the worker nodes and her command is running on the master node. When a spark job is submitted, the cluster manager then launches executors on the worker nodes on behalf of the driver (Master node). This means that the output she gets is on the master node which is the compiled result from the worker nodes but the split function would print just on the worker node and she is not able to see that.

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

When you run .collect(), the master/driver tells the worker nodes to send all the RDD data that they have to the master node and stores it in the master node memory. This can be a problem if the data is big because all the data from all the RDDs may not fit in the memory of the master node and the driver application can run out of memory causing a problem on the cluster.

D.4 Are partitions mutable? Why is this advantageous?

Yes, partitions in spark are mutable. The advantage of this mutability is that at times the programmers would like to change the partitioning scheme by changing the size of the partitions and number of partitions based on the requirements of the application and the available data and this mutability makes that possible. Another advantage is that choosing appropriate number of partitions can significantly improve the performance of an application as the number of partitions define the level of parallelism, and the ability to re-partition can help the users improve performance.

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

The RDDs are resilient in the sense that they can recompute missing or damaged partitions from history in case of node failures. RDDs achieve this by keeping a lineage graph, which is a graph of all the parent RDDs of a RDD. If some data is lost, it is possible to use the RDD Lineage to recreate the lost data.

Section D - Essay Questions

"A colleague has mentioned her Spark application has poor performance, what is your advice?"

To improve the performance of her Spark application my advice to her would be the following:

- **Use Data frames instead of RDDs**: The Spark DataFrame API (introduced in version 1.3) provides a table-like abstraction for storing data in-memory and provides performance optimizations by using Spark SQL's optimized execution engine [1]. The garbage collection overhead for Data Frames is lower compared to RDDs. RDD API is slower to perform simple grouping and aggregation operations as compared to Data Frames. In addition to this, Data Frames have better efficiency and memory usage ast he use of off-heap memory for serialization reduces the overhead and there is no need for deserialization for small operations [2].

- **Avoid UDFs where possible**: When we consider user defined functions in python, they are slow because they have to be executed in a Python process, rather than a JVM-based Spark Executor and there are additional serialization and desirializations costs.

- **Use caching**: Caching intermediate results can dramatically improve performance. Spark provides its own native caching mechanisms, which can be used through different methods such as .persist() and .cache() [2].

- **Customize cluster configuration**: Depending on the Spark cluster workload, a custom Spark configuration can result in more optimized Spark job execution. Configurations such as `num-executors`, `executor-memory` and `executor-cores` can be set manually [2].

- **Optimize data serialization**: Spark jobs are distributed, so performance can be improved by using appropriate data serialization e.g Kyro serialization [2].

- **Use Bucketing**: Bucketing is similar to data partitioning and is an optimization technique used with Spark SQL. Bucketing improves application performance by shuffling and sorting data prior use of operations such as table joins [3].

References

[1] A. Spark. Spark sql, dataframes and datasets guide. Apache. [Online]. Available: https://spark.apache.org/docs/latest/sql-programming-guide.html

[2] M. Azure. Optimize apache spark jobs. Microsoft. [Online]. Available: https://docs.microsoft.com/en-us/azure/hdinsight/spark/apache-spark-perf

[3] Databricks. How to improve performance with bucketing. Databricks. [Online]. Available: https://docs.databricks.com/user-guide/faq/bucketing.html