Big Data Technologies - CSP 554
Assignment 6
Prabhu Avula | A20522815
Illinois Institute of Technology, Chicago

Summary of the Article

The paper "A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench" by Ahmed et al. presents an in-depth empirical study comparing the performance of Hadoop MapReduce and Apache Spark frameworks when handling large-scale data sets. This analysis is crucial for system administrators and researchers to understand the behavior of these systems under various configurations and workloads, particularly as big data applications continue to grow in complexity and scale.

The study begins with an overview of related work, noting that while previous studies have compared Hadoop and Spark, they often focused on small data sets or limited workloads. The authors highlight the need for a more comprehensive analysis that includes a variety of parameters and larger data sets. To address this gap, they conduct experiments using two well-known workloads, WordCount and TeraSort, on data sets up to 600 GB.

The paper explains the fundamental differences between Hadoop and Spark. Hadoop MapReduce processes data in batches, reading and writing to disk between each map and reducing stage, which can lead to slower performance. In contrast, Spark processes data in memory, which can significantly speed up data processing tasks. The authors emphasize that understanding these differences is crucial for optimizing performance based on specific use cases.

The experimental setup section details the hardware and software configurations used for the study. The experiments were conducted on a 10-node cluster, each node equipped with an Intel Xeon CPU, 32 GB of RAM, and 6 TB of local storage. The authors used HiBench, a big data benchmark suite, to evaluate the performance of Hadoop and Spark. They meticulously configured various parameters for each framework to ensure a fair comparison.

The study's key focus is the tuning parameters that impact the performance of Hadoop and Spark. For Hadoop, parameters such as the size of the input split, the number of map and reduce tasks, and shuffle configurations were varied. Similarly, parameters, including the number of executors, executor memory, and shuffle configurations, were tuned for Spark. The authors aimed to identify which parameters had the most significant impact on performance.

The results and discussion section provides a detailed analysis of the experimental results. The authors found that Spark consistently outperformed Hadoop in terms of execution time due to its in-memory processing capabilities. For the WordCount workload, Spark was up to 5 times faster than Hadoop, while for TeraSort, the performance improvement was around 3 times. The study also measured system scalability, showing Spark performed better as the data set size increased.

Another important finding is the impact of tuning on performance. Properly configured, both Hadoop and Spark showed significant performance improvements. For instance, increasing the number of executors and executor memory led to substantial reductions in execution time. The authors highlight that understanding and tuning these parameters is essential for achieving optimal performance.

In conclusion, the paper demonstrates that while both Hadoop and Spark can handle large-scale data sets, Spark's in-memory processing provides a significant performance advantage. However, achieving optimal performance requires careful tuning of system parameters. The authors suggest that future research should explore additional workloads and configurations to refine these findings further and provide more comprehensive guidance for practitioners.

This comprehensive analysis provides valuable insights for anyone involved in big data processing. It offers a clear comparison of Hadoop and Spark and practical advice on system tuning to enhance performance.

Spark Exercises

1.

```
scala> val filePath = "hdfs:///user/hadoop/foodratings/foodratings20398.txt"
filePath: String = hdfs:///user/hadoop/foodratings/foodratings20398.txt

scala> val ex1RDD = sc.textFile(filePath)
ex1RDD: org.apache.spark.rdd.RDD[String] = hdfs:///user/hadoop/foodratings/foodratings20398.txt MapPartitionsRDD[9] at textFile a
t <console>:24

scala> ex1RDD.take(5).foreach(println)
Joy,29,49,16,24,5
Mel,30,42,39,12,5
Joy,26,20,10,23,3
Mel,1,1,25,20,5
Joe,13,28,40,25,4
```

2.

```
scala> val filePath = "hdfs:///user/hadoop/foodratings/foodratings20398.txt"
filePath: String = hdfs:///user/hadoop/foodratings/foodratings20398.txt

scala> val ex1RDD = sc.textFile(filePath)
ex1RDD: org.apache.spark.rdd.RDD[String] = hdfs:///user/hadoop/foodratings/foodratings20398.txt MapPartitionsRDD[11] at textFile
at <console>:24

scala> val ex2RDD = ex1RDD.map(line => line.split(","))
ex2RDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[12] at map at <console>:23

scala> val firstFiveRecords = ex2RDD.take(5)
firstFiveRecords: Array[Array[String]] = Array(Array(Joy, 29, 49, 16, 24, 5), Array(Mel, 30, 42, 39, 12, 5), Array(Joy, 26, 20, 1
0, 23, 3), Array(Mel, 1, 1, 25, 20, 5), Array(Joe, 13, 28, 40, 25, 4))

scala> firstFiveRecords.foreach(record => println(record.mkString(", ")))
Joy, 29, 49, 16, 24, 5
Mel, 30, 42, 39, 12, 5
Joy, 26, 20, 10, 23, 3
Mel, 1, 1, 25, 20, 5
Joe, 13, 28, 40, 25, 4
```

3.

```
scala> val filePath = "hdfs:///user/hadoop/foodratings/foodratings20398.txt"
filePath: String = hdfs:///user/hadoop/foodratings/foodratings20398.txt

scala> val ex1RDD = sc.textFile(filePath)
ex1RDD: org.apache.spark.rdd.RDD[String] = hdfs:///user/hadoop/foodratings/foodratings20398.txt MapPartitionsRDD[14] at textFile
at <console>:24

scala>

scala> val ex2RDD = ex1RDD.map(line => line.split(","))
ex2RDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[15] at map at <console>:23

scala>

scala> val ex3RDD = ex2RDD.map(line => Array(line(0), line(1), line(2).toInt, line(3), line(4), line(5)))
ex3RDD: org.apache.spark.rdd.RDD[Array[Any]] = MapPartitionsRDD[16] at map at <console>:23

scala>

scala> val firstFiveRecords = ex3RDD.take(5)
firstFiveRecords: Array[Array[Any]] = Array(Array(Joy, 29, 49, 16, 24, 5), Array(Mel, 30, 42, 39, 12, 5), Array(Joy, 26, 20, 10,
23, 3), Array(Mel, 1, 1, 25, 20, 5), Array(Joe, 13, 28, 40, 25, 4))

scala> firstFiveRecords.foreach(record => println(record.mkString(", ")))
Joy, 29, 49, 16, 24, 5
Mel, 30, 42, 39, 12, 5
Joy, 26, 20, 10, 23, 3
Mel, 1, 1, 25, 20, 5
Joe, 13, 28, 40, 25, 4
```

4.

```
scala> val filePath = "hdfs:///user/hadoop/foodratings/foodratings20398.txt"
filePath: String = hdfs:///user/hadoop/foodratings/foodratings20398.txt

scala> val ex1RDD = sc.textFile(filePath)
ex1RDD: org.apache.spark.rdd.RDD[String] = hdfs:///user/hadoop/foodratings/foodratings20398.txt MapPartitionsRDD[18] at textFile
at <console>:24

scala>

scala> val ex2RDD = ex1RDD.map(line => line.split(","))
ex2RDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[19] at map at <console>:23

scala>

scala> val ex3RDD = ex2RDD.map(line => Array(line(0), line(1), line(2).toInt, line(3), line(4), line(5)))
ex3RDD: org.apache.spark.rdd.RDD[Array[Any]] = MapPartitionsRDD[20] at map at <console>:23

scala>

scala> val ex4RDD = ex3RDD.filter(record => record(2).asInstanceOf[Int] < 25)
ex4RDD: org.apache.spark.rdd.RDD[Array[Any]] = MapPartitionsRDD[21] at filter at <console>:23

scala>

scala> val firstFiveRecords = ex4RDD.take(5)
firstFiveRecords: Array[Array[Any]] = Array(Array(Joy, 26, 20, 10, 23, 3), Array(Mel, 1, 1, 25, 20, 5), Array(Joy, 43, 12, 31, 26
, 2), Array(Joe, 27, 8, 22, 4, 1), Array(Sam, 26, 16, 47, 43, 1))

scala> firstFiveRecords.foreach(record => println(record.mkString(", ")))
Joy, 26, 20, 10, 23, 3
Mel, 1, 1, 25, 20, 5
Joy, 43, 12, 31, 26, 2
Joe, 27, 8, 22, 4, 1
Sam, 26, 16, 47, 43, 1
```

5.

```
scala> val ex5RDD = ex4RDD.map(record => (record(0), record))
ex5RDD: org.apache.spark.rdd.RDD[(Any, Array[Any])] = MapPartitionsRDD[22] at map at <console>:23

scala>

scala> val firstFiveRecords = ex5RDD.take(5)
firstFiveRecords: Array[(Any, Array[Any])] = Array((Joy,Array(Joy, 26, 20, 10, 23, 3)), (Mel,Array(Mel, 1, 1, 25, 20, 5)), (Joy,A
rray(Joy, 43, 12, 31, 26, 2)), (Joe,Array(Joe, 27, 8, 22, 4, 1)), (Sam,Array(Sam, 26, 16, 47, 43, 1)))

scala> firstFiveRecords.foreach(record => println(record._1 + ", " + record._2.mkString(", ")))
Joy, Joy, 26, 20, 10, 23, 3
Mel, Mel, 1, 1, 25, 20, 5
Joy, Joy, 43, 12, 31, 26, 2
Joe, Joe, 27, 8, 22, 4, 1
Sam, Sam, 26, 16, 47, 43, 1
```

6.

```
scala> import org.apache.spark.SparkContext._
import org.apache.spark.SparkContext._

scala> import org.apache.spark.rdd.RDD
import org.apache.spark.rdd.RDD

scala>

scala> val filePath = "hdfs:///user/hadoop/foodratings/foodratings20398.txt"
filePath: String = hdfs:///user/hadoop/foodratings/foodratings20398.txt

scala> val ex1RDD = sc.textFile(filePath)
ex1RDD: org.apache.spark.rdd.RDD[String] = hdfs:///user/hadoop/foodratings/foodratings20398.txt MapPartitionsRDD[24] at textFile at <console>:28

scala>

scala> val ex2RDD = ex1RDD.map(line => line.split(","))
ex2RDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[25] at map at <console>:27

scala>

scala> val ex3RDD = ex2RDD.map(line => Array(line(0), line(1), line(2).toInt, line(3), line(4), line(5)))
ex3RDD: org.apache.spark.rdd.RDD[Array[Any]] = MapPartitionsRDD[26] at map at <console>:27

scala>

scala> val ex4RDD = ex3RDD.filter(record => record(2).asInstanceOf[Int] < 25)
ex4RDD: org.apache.spark.rdd.RDD[Array[Any]] = MapPartitionsRDD[27] at filter at <console>:27

scala>

scala> val ex5RDD: RDD[(String, Array[Any])] = ex4RDD.map(record => (record(0).toString, record))
ex5RDD: org.apache.spark.rdd.RDD[(String, Array[Any])] = MapPartitionsRDD[28] at map at <console>:27

scala>

scala> val ex6RDD = ex5RDD.sortByKey()
ex6RDD: org.apache.spark.rdd.RDD[(String, Array[Any])] = ShuffledRDD[31] at sortByKey at <console>:27

scala>

scala> val firstFiveRecords = ex6RDD.take(5)
firstFiveRecords: Array[(String, Array[Any])] = Array((Jill,Array(Jill, 15, 9, 18, 26, 5)), (Jill,Array(Jill, 10, 24, 28, 30, 5)), (Jill,Array(Jill, 18, 19, 35, 39, 5)), (Jill,Array(Jill, 11, 5, 15, 32, 4
)), (Jill,Array(Jill, 49, 6, 39, 2, 1)))

scala> firstFiveRecords.foreach(record => println(record._1 + ", " + record._2.mkString(", ")))
Jill, Jill, 15, 9, 18, 26, 5
Jill, Jill, 10, 24, 28, 30, 5
Jill, Jill, 18, 19, 35, 39, 5
Jill, Jill, 11, 5, 15, 32, 4
Jill, Jill, 49, 6, 39, 2, 1
```