Name: Pankaj Bhootra
Matriculation No.: A0144919W
NUSNET ID: E0009011

CS4225 Assignment 2 Report

Task 1

Comparisons on programming with Hadoop and Spark:

While Hadoop and Spark do many of the same tasks, there are also few noticeable differences. The first thing I noticed was that Spark did not have its own file management system, so it still relies on Hadoop's Distributed File System (HDFS) or other similar solutions. Hence, I think that the pairing of Spark's optimal data processing engine with Hadoop's DFS is a smart, efficient and powerful solution for big data applications.

However, Spark is very different from Hadoop's MapReduce as it has a much faster real-time data processing capability as opposed to MapReduce that is more disk-bound and batch-oriented, giving Spark solutions a faster execution time of 10-100 times compared to the Hadoop counterparts.

I especially noticed this while solving Task 1 of CommonWords example, as we wrote both Hadoop and Spark version for this problem. I found some noticeable conveniences in working with Spark instead, as mentioned below:

1. Much less verbosity: The code written using Spark is a lot less verbose compared to Hadoop, as in case of Hadoop we have to define Mapper and Reducer classes for each sub-part of the problem that is a MapReduce job in itself. In comparison, Spark provides a simpler RDD data type interface, on which functions like map and reduce can be called just like normal methods. Hence, a MapReduce operation on an input dataset can be performed in just one line of code in Spark using RDDs and Functional Programming, whereas for Hadoop it would take many lines of multiple Mapper/Reducer classes.
2. Expressive API/ease of use: Spark comes with many predefined MapReduce protos such as groupBy, aggregateByKey, sortByKey, flatMap, mapValues, reduce, reduceByKey, etc. In case of my CommonWords solution for Hadoop, I had to write multiple MapReduce classes for even the simple tasks such as sorting the final output by word count. However, in case of Spark, the sortByKey library provided allows me to achieve the same objective in a much simpler manner (also in just one line!). Hence, this allows the programmer to focus more on solving the actual problem rather than spending time figuring out how to solve the very simple parts of the problem.

The CommonWords task took around 200 lines of code in Hadoop as opposed to 40 lines approximately in Spark, making the latter much more convenient and programmer-friendly.

Comparisons on runtime execution with Hadoop and Spark:

Spark not only provides greater convenience and flexibility in terms of programming, but as mentioned before, it provides a much greater boost in performance thanks to its data processing engine. While MapReduce jobs are slow and batch-oriented, Spark jobs run in real-time and can easily transfer results from one operation (such as map, reduce or sort) to the other without any intermediate I/O operations. Eliminating the intermediate I/O operations which involve reading from and writing to disk saves a lot of time in case of Spark as the result of one operation is directly fed to the next one.

In case of the CommonWords implementation in both Hadoop and Spark, the Spark job took only around 5-6 seconds whereas the Hadoop solution, which had multiple MapReduce jobs in it, took around 1 minute, giving a 10x boost in time efficiency.

This makes sense as Spark processes all the different steps for CommonWords in memory. This leads me to believe that Spark's in-memory processing can deliver near real-time analytics for many large datasets as compared to MapReduce's batch-processing mechanism. MapReduce was never really built to provide blinding speed, because its original goal was to just continuously crawl and index information from websites (during its development at Google) and there were no "real-time" requirements for this data.

Other differences:

Besides its ease of use and performance, another advantage of Spark over Hadoop is that it comes with user-friendly APIs across multiple programming languages such as Scala (its native language), Java, Python and even SQL. In fact, Spark SQL is so conveniently abstracted that on its surface, it appears identical to SQL 92, so there is no additional learning curve required to learn how to use it.

In terms of costs, Spark requires a lot of RAM to compute processes in-memory, whereas Hadoop MapReduce requires faster disks since its processing is disk-based and batch-oriented. Also, Spark's technology reduces the number of systems required to solve the same task vs. Hadoop's MapReduce, so it runs with significantly fewer systems (though, with additional RAM).

I also want to share a quote that highlights all the above benefits of Spark: "Spark has been shown to work well up to petabytes. It has been used to sort 100 TB of data 3x times faster than Hadoop MapReduce on one-tenth of the machines. This feat won Spark the 2014 Daytona GraySort Benchmark."

Hence, Spark is not only efficient in terms of programming, but also more time efficient in runtime execution, learning and costs. It provides a significant improvement in data processing and can be used in tandem with Hadoop's Distributed File System to solve many big data applications.

Task 2

Analysis of Result:

TODO

Analysis of the parameters in K-Means:

TODO

Further discussion on the system performance:

TODO