# Systems for Data Science - HW6

Spark - Paper Review

Shubham Shetty

February 25 2021

# 1 Summary

Spark provides a language-oriented programming interface to *Resilient Distributed Datasets* (RDDs), written in the Scala language. Basically the Spark system acts as an additional layer of abstraction over RDDs, which is the main focus of the given paper.

Resilient Distributed Datasets are distributed memory abstractions which allow for in-memory computations on large clusters in a fault tolerant manner. Technically speaking, RDDs are parallel distributed data structures which allow users to programatically control their partitioning and manipulate them using a rich set of operators. In general, distributed systems such as Hadoop require data to be present on disk which results in large overhead and network bandwidth bottlenecks due to data transfer. In contrast, RDDs provide coarse-grain operations which are performed on the various partitions parallely. These operations are performed on the dataset (like map, transform, reduce, etc.) which are then stored in a built-up dataset called lineage. Lineage is just a list of transforms performed on the RDD, which are persisted to memory instead of actual data. Spark lazily evaluates these functions only when a action is performed (transformations are stored in lineage). Hence if there is a fault, the data state can be recreated just by following transformation steps present in the lineage.

RDDs are abstracted as software via Spark, which treats each dataset as an object and allows users to perform transformations and actions on these objects via method calls. Programmers can perform functions such as map, reduce, aggregate functions, persist, etc. Programmers use a driver program that connects to a cluster of workers. This is analogous to the master-slave architecture, where the driver acts as master and carries out several similar critical tasks, such as invoking actions on RDDs, tracking lineage etc.

Advantages of RDDs can be observed when comparing them against distributed shared memory architecture. RDDs can be written through coarse grain writes whereas DSMs support both coarse- and fine-grained writes. This ensures that RDDs are more fault tolerant at the expense of bulk writes. RDDs are immutable and hence consistent in nature, however DSMs' consistency depends on the application. Also, while straggler mitigation is difficult in DSM architecture, for RDD it resolves this by using backup tasks. Due to all these reasons, RDDs are best for data processing applications which require a batch processing, i.e. where same operation is executed on all elements of the datasets. RDDs are not suitable for applications where multiple fine grain changes/operations are executed, such as streaming or real time OLTP applications.

## 2 Strengths

The strengths of the Spark framework are -

- High Performance - Spark processes data at a very high speed, 100x faster than Hadoop for large scale processing. As Spark uses in-memory processing, latency for data access is lesser as compared to Hadoop which stores all data on local disc, leading to higher processing speeds.
- Easy to Use - Spark is packaged with easy-to-use APIs which help in reducing the syntax learning curve and makes it easier to build big data processing apps. Spark can also be programmed using Scala, Python, and Java, and also supports SQL via SparkSQL.
- RDDs are fault tolerant by design. Transformations are lazily evaluated and stored in memory as lineage, which can then be recreated and executed in case a server running the RDD program fails.

## 3 Unclear Aspects

The unclear aspects with respect to Spark are -

- How does Spark perform for smaller data? Is it optimal only for big data use cases or can it be used for normal data sizes as well?
- Can Spark handle the kinds of algorithms which are not typically supported by MapReduce?
- How are the RDDs implemented on the underlying data store?

## 4 Limitations/Areas of Improvement

The limitations of RDDs and Spark are -

- Spark processing is memory intensive - it requires a lot of RAM to be able to effectively process big data. Hence Spark processing can have higher costs.
- Spark requires manual optimization. Optimizing and fine-tuning Spark for performance requires a deep knowledge of the Spark system and architecture, and is more difficult to learn. It would be better if Spark could introduce automated tuning/optimization.
- Spark is still better for batch processing than for real-time stream processing.