# Review - *Spark*: Cluster Computing with Working Set

Name: Kunal Baweja
UNI: kb2896
COMS E6998 Cloud Computing & Big Data (Fall 2016) October 28, 2016

---

## Introduction

The *Spark*[1] cluster computing framework introduced in the paper provides a simple support for working data sets, which can be reused multiple times in applications across multiple parallel operations, unlike the traditional MapReduce based frameworks. Moreover, it still retains the scalability and fault tolerance equivalen to MapReduce by introducing an bastraction called *RDD (Resilient Distributed Dataset)* which is a read-only collection of objects partitioned across a set of machines.

## Programming Model

Developers using *spark*[1] are required to write a high-level control flow of their application, known as *driver program* which launches *parellel operations* that run on top of *resilient distributed datasets* which are described below:

**1. Resilient Distributed Datasets (RDD)**
RDD represents a read-only collectios of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like parallel operations.

Fault tolerance is achieved through a notion of *lineage* i.e if a partition is lost, the RDD reconstructs it from information retained in the system to derive it back from other partitions, hence re-building the partition.

Each RDD is represented by a Scala object (scala integrated with *spark*[1]) and allows programmers to construct RDDs in four ways:

1. From a *file* in shared file system.
2. By dividing a scala collection into a number of slices sent to multiple nodes (known as *paralleliztion*).
3. By transforming an existing RDD i.e. changing a dataset of elements of type A to dataset with elements of type B.
4. By changing the persistence of an RDD i.e specifying *cache* and/or *save* actions on it

**2. Parallel Operations**
Majorly three types of parallel operations can be performed on RDDs:

1. *reduce* operation combines dataset elements to produce result at the driver program.
2. *collect* operation sends all the elements of the dataset to the driver program
3. *foreach* operation passes each element through a user provided function.

**3. Shared Variables**
*Spark*[1] also provides restricted shared variables of two types:

1. *Broadcast variables:* These act as wrappers which ensure that each value is copied to all of the worker nodes, but only once.
2. *Accumulators*: Worker nodes can only add data to these variables which can be read only by the driver program.

## Working of Spark

*Spark*[1] runs on top of *Mesos* operating system for clusters which allows multiple parallel applications to share a cluster in a fine grained manner. When the driver program invokes a paralle operation, *spark*[1] creates *tasks* to process each partition of the dataset and sends these tasks to worker nodes via delayed scheduling. Different types of RDDs differ in their implementation of RDD interface, the most prominnent ones being *MappedDataset* and *CachedDataset*.

While shipping tasks to workers, *closures* are shipped to worker nodes. These tasks ultimately fetch the dataset from the workers and add them using the accumulators, which leads to the driver program reading off the combined data quickly from the accumulator, as described above.

## References

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud’10, (Berkeley, CA, USA), pp. 10–10, USENIX Association, 2010.