**Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing**

**Raghupriya Nekkalapu**

**Problem**

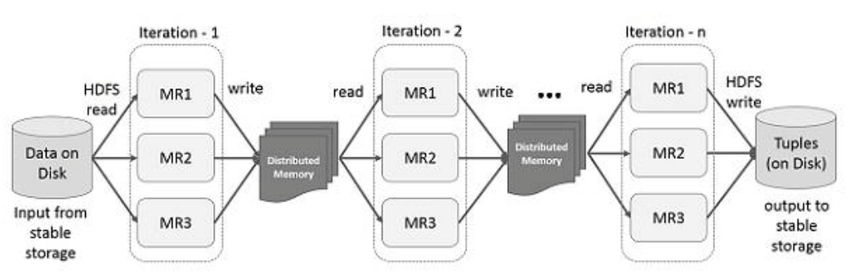
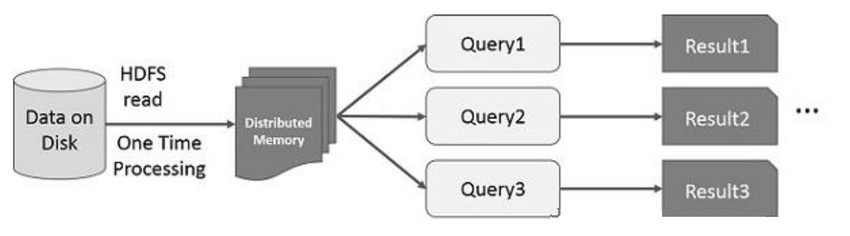
Many developing applications like K-means clustering, Machine Learning, PageRank, and the iterative algorithms behind them, as well as interactive data analysis/mining, where users run ad-hoc queries on the data. Across multiple computations they rely on the use of intermediate results. Nevertheless, almost all existing cluster computing frameworks like MapReduce, Dryad etc., adopted for data intensive analytics are lacking/inefficient in this area. Their course of action to reuse depending on data on writing the results to external stable storage system, may contain the vast amount of enumerate execution, which is due to data duplicity/replication, disk I/O, network bandwidth. Many programming models which are specialized related for this concern are not general purpose, distributed memory, abstraction and fault tolerant. This is the problem that RDD are designed to solve by addressing computing the needs that were met previously by introducing new specialized models (Map Reduce).

**Solution RDD**

To accommodate the problems mentioned a solution is RDD, explicitly present as read-only, immutable portioned collection records. RDD provides an interface in spark programming based on coarse grained transformations, from which a dataset called lineage, RDD keep enough information about their sources, is built rather than the actual data. In the stable storage the actual data is stored by RDD. Thus, lineage ensures that lost partitions are reconstructed in case of any failures. RDD, in general two features, partitioning and persistence which enable users to reuse intermediate results in memory and optimize many operations using data parallelization and precisely handle them by a range of operations like map and filter (Narrow dependencies). On real world applications it shows good performance for those naturally perform same operations to multiple data items (wide dependencies).

**Novelty**

Traditionally Transformation or functions applied to all data, in-memory data storage abstractions for cluster computing, for instance distributed shared memory depends on fine grained updates to shared state, involving data replication and logging updates to build the lineage graph across machines, which can be time consuming and resource costly for computation in large scale. In such cases RDD provides a novel solution build on coarse-grained transformation on data items, realizes an efficient, fault tolerant data storage abstraction, and a specific property guaranteed that an RDD that a program cannot reference an RDD that it cannot construct after a failure. To differentiate the RDD from the other in-memory based systems such as DSM and Piccolo use checkpointing and roll back, expensive than lineage mechanism, in which the interface only reads and updates changeable states of dataset, moreover RDD provide a high level programming interface enabling to manipulate data by users. Also, MapReduce and Dryad use lineage-based recovery within a computation, but this information is lost after a job ends. In contrast, RDDs persists lineage information across computations which is very quicker and cheaper partition recovery. RDD’s are created by using the transformations from existing one, but actions is required to work on the dataset.

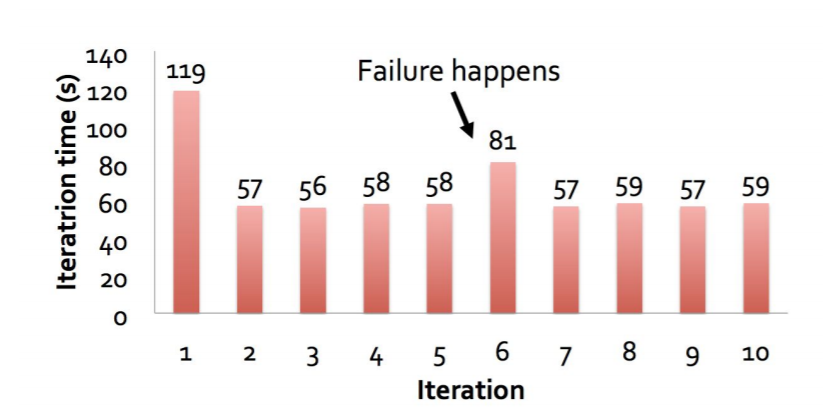
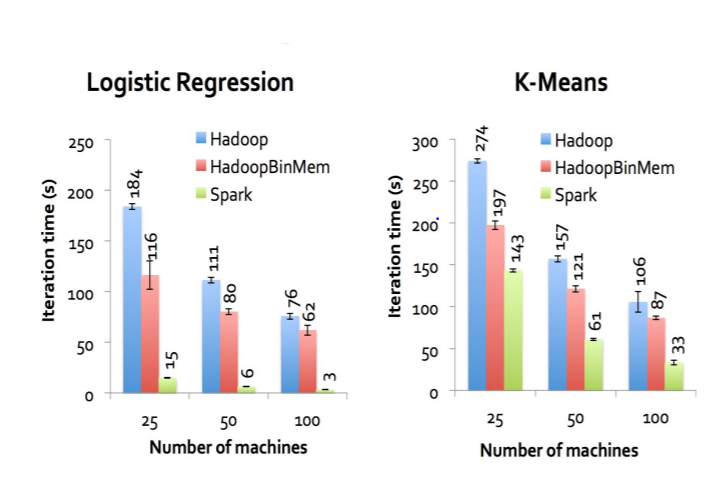
 

(a) Iterative Operations on Spark RDD (b) Interactive Operations on Spark RDD

**Evaluation**

In the system spark and RDD are implemented and are evaluated on the basis of range of experiments Amazon EC2, which includes the benchmarks in opposition to Hadoop with respect to Machine Learning, also PageRank, fault recovery, behavior with inadequate memory, User application measurements and interactive datamining. Strengths of RDD and spark contemplate on iterative algorithms (in machine learning Logistic regression and K-means) and data intensive analytics/mining. Spark surpassed Hadoop in terms of speeds through data storing in memory which makes avoidance of replication and I/O traffic, as well, require less RAM. Based on the lineage the fault recovery mechanism reconstructs the lost partition and the extension cost time is subtle. It can query the large amount of data interactively. Spark RDD are suitable for broad range of applications and express new applications capture by them not only the specialized programming models for iterative computations. One weakness that RDD that to ahigh level it depends on its memory space to store the data operations. The RDD stand out only at the applications which are parallel that perform the same operations to multiple elements of dataset, which is due to coarse-grained transformation and read-only property.

Hadoop is outperformed by Spark 20 times in iterative machine learning and graph-based applications. Spark can quickly recover by reconstructing only at the lost RDD partitions when the node fails.



Spark RDD and its core principle quiver the place of Hadoop and MapReduce in the field of open source distributed computing frameworks and impoverished their limitations on interactive and iterative operations. In addition, it is consistent with HDFS, therefore fits in the environment of Hadoop.

**Limitations**

RDDs are most appropriate for cluster applications that apply a similar activity or operations to all dataset elements, but not suites for asynchronous fine-grained applications that make updates to shared state. For the sake of caching, sparks load the process into memory. The major performance degradations come when the data is too big to fit entirely into the memory. RDD has Storage, performance limitation and fails to handle structured data.

**Solution**: Data Frames of spark solve all the limitations of RDD. When Data Frame and Dataset emerged, the system become more friendly and plays with vast amount of data. To overcome the issue, we increase the size of disk and RAM. Then this might have question raise about what if we don’t have enough memory? When data doesn’t fit in the memory, Spark operators spill the data in disk allowing it to run well on any sized data. Whatever Hadoop provides Spark can also provide all the capabilities. For instance, in Hadoop, stages in MapReduce which starts with HDFS and end at HDFS, which sustains the overheads because of disk I/O, data replication, and serialization dominates execution times of applications. Action result stored in driver program or may be written to the external storage because of RDD stored in memory. Which leads to the actual computation or evaluation.

**Conclusion**

We can conclude that RDD’s are partitioned collection of records, immutable, which can only be built through the coarse-grained operations like Transformations (map, filter join, etc.,). We can achieve efficient fault tolerant recovery using lineage. Spark programming interface provides the RDD’s and perform operation like transformation (build new RDD’s) and actions (compute an output the result). Spark Controls the RDD’s by using partitioning (layout across nodes) and persistence (storage in RAM, on disk, etc.,). Finally, we can say that Spark RDD is a fault tolerant abstraction for in-memory cluster computing which recovers data using lineage instead of replication and performs much better on iterative computations and interactive data queries