Why Apache Spark Architecture if we have Hadoop?

The Hadoop Distributed File System (HDFS), which stores files in a Hadoop-native format and parallelizes them across a cluster, and applies MapReduce the algorithm that actually processes the data in parallel. The catch here is Data Nodes are stored on disk and processing has to happen in Memory. Thus we need to do lot of I/O operations to process and also Network transfer operations happen to transfer data across the data nodes. These operations in all may be a hindrance for faster processing of data.

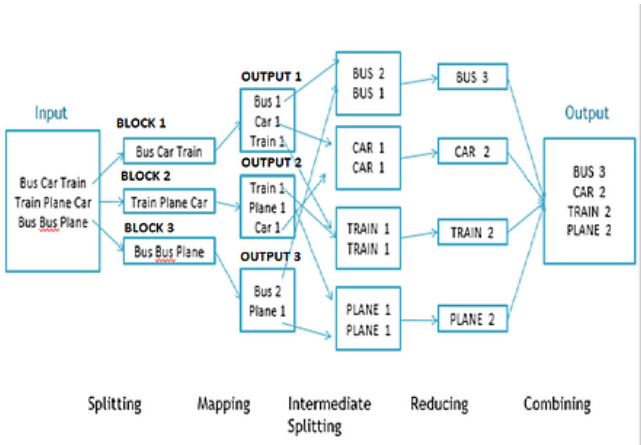
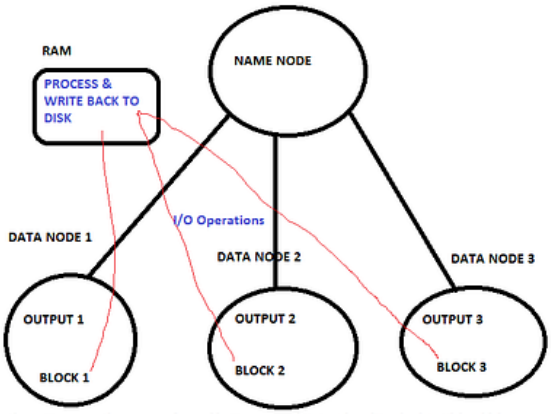


Fig. WorkFlow of MapReducing



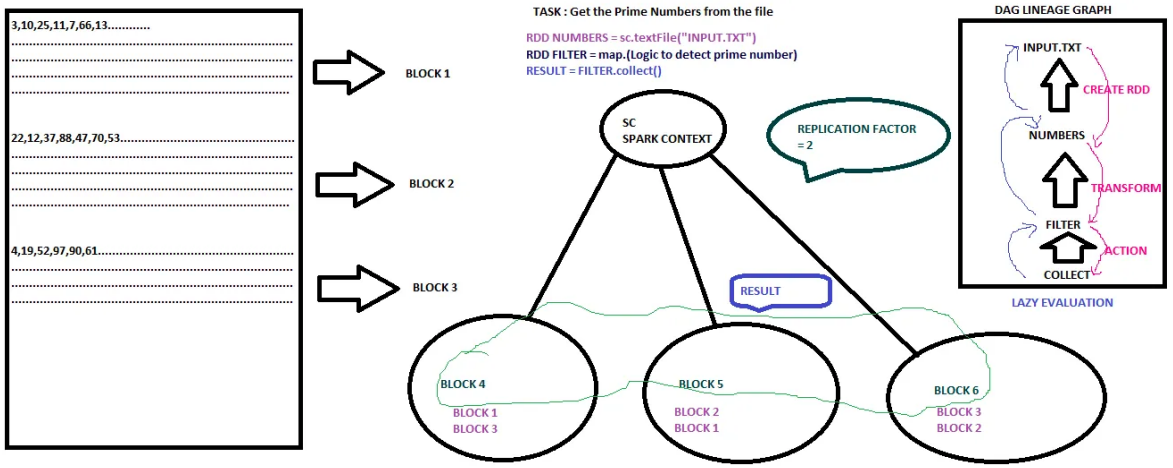
These Map operations are performed in Memory and Output is written back to Disk, which requires I/O operations.

Above Image describes, blocks are stored on data notes which reside on disk and for Map operation or other processing has to happen in RAM. This requires to and fro I/O Operation which causes a delay in overall result.

Apache Spark: Official website describes it as : “Apache Spark is a **fast** and **general-purpose** cluster computing system”.

Fast: Apache spark is fast because computations are carried out in memory and stored there. Thus there is no picture of I/O operations as discussed in Hadoop architecture.

General-Purpose: It is an optimized engine that supports general execution graphs. It also supports a rich SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming for live data processing.



Entry point to Spark is Spark Context which handles the executors nodes. The main abstraction data structure of Spark is Resilient Distributed Dataset (RDD), which represents an **immutable** collection of elements that can be operated on in parallel.

Lets discuss the above example to understand better: A file consists of numbers, task is find the prime numbers from this huge chunk of numbers. If we divide them into three blocks B1,B2,B3. These blocks are immutable are stored in Memory by spark. Here the replication factor=2, thus we can see that a copy of other node is stored in corresponding other partitions. This makes it to have a fault-tolerant architecture.

Step 1 : Create RDD using Spark Context
Step 2 : **Tranformation**: When a **map()** operation is applied on these RDD, new blocks i.e B4, B5, B6 get created as new RDD’s which are immutable

again. This all operations happen in Memory. Note: B1,B2,B3 still exist as original.
Step 3 : **Action**: When **collect(),** this when the actual results are collected and returned.

LAZY EVALUATION: Spark does not evaluate each transformation right away, but instead batch them together and evaluate all at once. At its core, it optimizes the query execution by **planning out the sequence of computation and skipping potentially unnecessary steps**. Main **Advantages** : Increases Manageability**,** Saves Computation and increases Speed, Reduces Complexities, Optimization.

How it works ? When it we execute the code to create Spark Context, then create RDD using sc, then perform tranformation using map to create new RDD. In actual these operations are not executed in backend, rather a **Directed Acyclic Graph(DAG) Lineage** is created. Only when the **action** is performed i.e. to fetch results, example : **collect()** operation is called then it refers to DAG and climbs up to get the results, refer the figure, as climbing up it sees that filter RDD is not yet created, it climbs up to get upper results and finally reverse calculates to get the exact results.

RDD — **Resilient :** i.e. fault-tolerant with the help of RDD lineage graph. RDD's are a deterministic function of their input. This plus immutability also means the RDD's parts can be recreated at any time. This makes caching, sharing and replication easy.

Distributed : Data resides on multiple nodes.