A

Special Study Report

on

SPARK(A DISTRIBUTED FRAMEWORK)

Submitted in Partial Fulfillment of
the Requirements for the Third Year
of

Bachelor of Engineering

in

Computer Engineering

to

North Maharashtra University, Jalgaon

Submitted by

Ankita Kishor Wani

Under the Guidance of

Mr. Atul V. Dusane

DEPARTMENT OF COMPUTER ENGINEERING
SSBT's COLLEGE OF ENGINEERING AND TECHNOLOGY,
BAMBHORI, JALGAON - 425 001 (MS)
2015 - 2016

SSBT's COLLEGE OF ENGINEERING AND TECHNOLOGY, BAMBHORI, JALGAON - 425 001 (MS)

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the special study entitled *SPARK(A Distributed Framework)*, submitted by

Ankita Kishor Wani

in partial fulfillment of the Third Year of *Bachelor of Engineering* in *Computer Engineering* has been satisfactorily carried out under my guidance as per the requirement of North Maharashtra University, Jalgaon.

Date: October 15, 2015
Place: Jalgaon

Mr. Atul V. Dusane
Guide

Prof. Dr. Girish K. Patnaik
Head

Prof. Dr. K. S. Wani
Principal

Acknowledgement

I would like to express my gratitude and appreciation to all those who gave me the possibility to complete this report. A special thanks to our guide Mr. Atul V. Dusane , whose help, stimulating suggestions and encouragement, helped me to coordinate my seminar especially in writing this report.

I would also like to acknowledge with much appreciation the crucial role of the staff of Computer Department, who helped me a lot and gave the permission to use all required machinery and the necessary material to complete the seminar.

Last but not the least, many thanks go to the Head of the Department, Prof. Dr. G. K. Patnaik who have given his full effort in guiding me in achieving the goal as well as his encouragement to maintain my progress in track. I would like to appreciate the guidance given by other supervisors as well as the panels especially in our seminar presentation that has improved our presentation skills by their comments and tips.

Ankita Kishor Wani

Contents

List of Tables

List of Figures

Abstract

Distributed storage is fundamental to many of todays Big Data projects as it allows vast multi-petabyte datasets to be stored across an almost infinite number of everyday computer hard drives, rather than involving hugely costly custom machinery which would hold it all on one device. These systems are scalable, meaning that more drives can be added to the network as the dataset grows in size. Hadoop and Spark are both Big Data frameworksthey provide some of the most popular tools used to carry out common Big Data-related tasks. Hadoop, for many years, was the leading open source Big Data framework but recently the newer and more advanced Spark has become the more popular of the two Apache Software Foundation tools. However they do not perform exactly the same tasks, and they are not mutually exclusive, as they are able to work together. Although Spark is reported to work up to 100 times faster than Hadoop in certain circumstances, it does not provide its own distributed storage system.

Chapter 1

Introduction

This chapter contains overview of Spark and Hadoop framework. Some of the drawbacks of Hadoop and few features of Spark that overcome Hadoop's limitations.

Spark is a framework for performing general data analytics on distributed computing cluster like Hadoop.It provides in memory computations for increase speed and data process over mapreduce.It runs on top of existing hadoop cluster and access hadoop data store (HDFS), can also process structured data in Hive and Streaming data from HDFS,Flume,Kafka,Twitter.

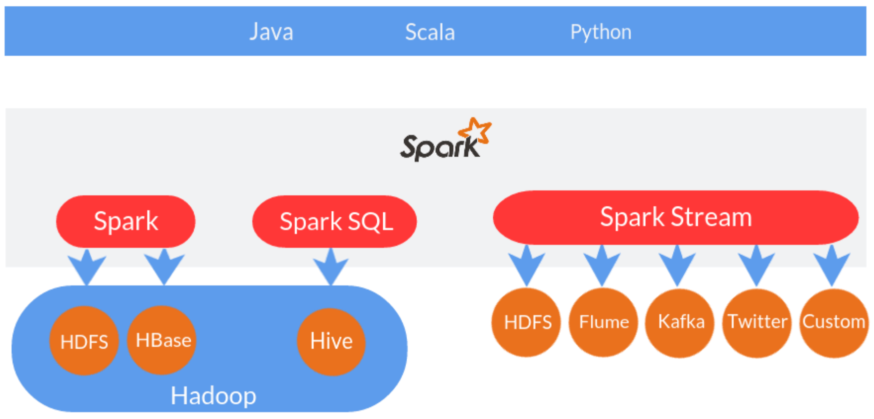

Figure 1.1: Echosystem of Spark

Hadoop is parallel data processing framework that has traditionally been used to run map/reduce jobs. These are long running jobs that take minutes or hours to complete. Spark has designed to run on top of Hadoop and it is an alternative to the traditional batch map/reduce model that can be used for real-time stream data processing and fast interactive queries that finish within seconds. So, Hadoop supports both traditional map/reduce and Spark.

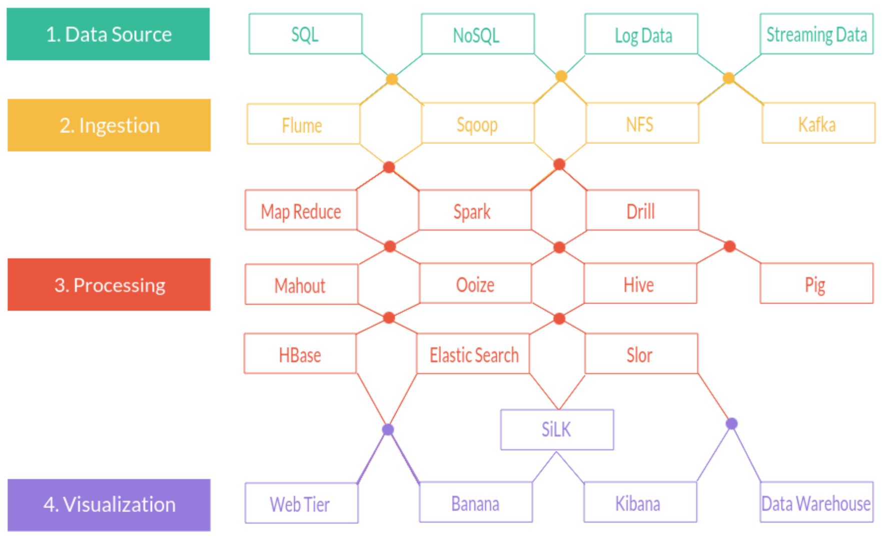

Figure 1.2: Echosystem of Hadoop

1.1 Limitations of Hadoop

A big data framework, Hadoop has some drawbacks related to its working, performance and use of available memory. Some limitations are listed below:

- Batch mode
  - Only batch layer in the Lamda pattern.
  - No real time
  - No repititive queries
- Iterative algorithms
- Interactive data queries
- Poor support for Disributed memeory

1.2 Why Apache Spark was developed?

Hadoop MapReduce that was envisioned at Google and successfully implemented and Apache Hadoop is an extremely famous and widely used execution engine. You will find several applications that are on familiar terms with how to decompose their work into a sequence of MapReduce jobs. All these real time applications will have to continue their operation without any change.

However the users have been consistently complaining about the high latency problem with Hadoop MapReduce stating that the batch mode response for all these real time applications is highly painful when it comes to processing and analyzing data.

Now this paved way for Hadoop Spark, a successor system that is more powerful and flexible than Hadoop MapReduce. Despite the fact that it might not be possible for all the future allocations or existing applications to completely abandon Hadoop MapReduce, but there is a scope for most of the future applications to make use of a general purpose execution engine such as Hadoop Spark that comes with many more innovative features, to accomplish much more than that is possible with MapReduce Hadoop.

1.3 Features of Spark

Spark data framework is used for big data processing in a distributed environment. It has some exception features over Hadoop. It can be viewed as a replacement to Hadoop on the basis of these plus points.

- APIs and libraries for Java, Scala, Python, R, and other languages.
- Scalability to over 8000 nodes in production.
- Ability to cache datasets in memory for interactive data analysis: extract a working set, cache it, query it repeatedly.
- Interactive command line interface (in Scala or Python) for low-latency, horizontally scalable, data exploration.
- Higher level library for stream processing, using Spark Streaming.
- Support for structured and relational query processing (SQL), through Spark SQL.
- Higher level libraries for machine learning and graph processing.

1.4 Summary

In this chapter, we have seen some part of Hadoop and Spark. Also some limitations of Hadoop and Features of Spark. In next chapter we will see the history of Spark.

Chapter 2

Literature Survey

Further to the introduction of Spark and Hadoop , its history and releases are described in given chapter.

2.1 Backgroud and History

Hadoop as a big data processing technology has been around for 10 years and has proven to be the solution of choice for processing large data sets. MapReduce is a great solution for one-pass computations, but not very efficient for use cases that require multi-pass computations and algorithms. Each step in the data processing workflow has one Map phase and one Reduce phase and you'll need to convert any use case into MapReduce pattern to leverage this solution.

The Job output data between each step has to be stored in the distributed file system before the next step can begin. Hence, this approach tends to be slow due to replication & disk storage. Also, Hadoop solutions typically include clusters that are hard to set up and manage. It also requires the integration of several tools for different big data use cases (like Mahout for Machine Learning and Storm for streaming data processing).

If you wanted to do something complicated, you would have to string together a series of MapReduce jobs and execute them in sequence. Each of those jobs was high-latency, and none could start until the previous job had finished completely.

2.2 Invention of Spark

Apache Spark is an open source big data processing framework built around speed, ease of use, and sophisticated analytics.

Apache Spark is an open source cluster computing framework originally developed the contributions of near about 250 developers from 50 companies in the AMPLab at University of California, Berkeley but was later donated to the Apache Software Foundation where it remains today.

In contrast to Hadoop's two-stage disk-based MapReduce paradigm, Spark's multi-stage in-memory primitives provides performance up to 100 times faster for certain applications. By allowing user programs to load data into a cluster's memory and query it repeatedly, Spark is well-suited to machine learning algorithms

2.3 Became Apache's Spark

Spark was initially started by *Matei Zaharia* at UC Berkeley AMPLab in **2009**, and open sourced in **2010** under a BSD license.

In 2013, the project was donated to the Apache Software Foundation and switched its license to Apache 2.0. In February 2014, Spark became a Top-Level Apache Project.

In November 2014, the engineering team at Databricks used Spark and set a new world record in large scale sorting.

2.4 How Apache Spark started?

The story started back in 2009 with mesos. It was a class project in UC Berkley. Idea was to build a cluster management framework, which can support different kind of cluster computing systems.Once mesos was built, then what we can build on top of mesos. Thats how spark was born.

At that time, Hadoop targeted batch processing, The requirement for machine learning came from the head of department at that time. They were trying to scale machine learning on top of Hadoop. Hadoop was slow for ML as it needed to exchange data between iterations through HDFS. The requirement for interactive querying came from the company, Conviva, which he founded that time.

As part of video analytical tool built by company, they supported adhoc querying. They were using Mysql at that time, but they knew its not good enough. Also there are many companies came out of UC,like Yahoo,Facebook were facing similar challenges. So they knew there was need for framework focused on interactive and iterative processing.

2.5 Release of Spark 1.0.0

Spark 1.0.0 is a major release marking the start of the 1.X line. This release brings both a variety of new features and strong API compatibility guarantees throughout the 1.X line. Spark 1.0 adds a new major component, Spark SQL, for loading and manipulating structured data in Spark. It includes major extensions to all of Sparks existing standard libraries (ML, Streaming, and GraphX) while also enhancing language support in Java and Python.

Finally, Spark 1.0 brings operational improvements including full support for the Hadoop/-YARN security model and a unified submission process for all supported cluster managers.

2.6 Summary

In this chapter , we have touched the background of Spark, it's born and releases. Further we will visit an architecture of Spark.

Chapter 3

Methodology

An important part i.e architecture of Spark must be discussed. So, the next part will elaborate internal design, components of Spark. the working from multiple layers too.

3.1 Architecture

In previous section , we have seen background of Spark. Now, we will look up into its architecture and working.

Although Spark has similarities to Hadoop, it represents a new cluster computing framework with useful differences. First, Spark was designed for a specific type of workload in cluster computingnamely, those that reuse a working set of data across parallel operations (such as machine learning algorithms). To optimize for these types of workloads, Spark introduces the concept of in-memory cluster computing, where datasets can be cached in memory to reduce their latency of access.

3.1.1 Components of Spark

The Spark project consists of multiple components.

1. **Spark Core and Resilient Distributed Datasets :**

   Spark Core is the foundation of the overall project. It provides distributed task dispatching, scheduling, and basic I/O functionalities. The fundamental programming abstraction is called Resilient Distributed Datasets (RDDs), a logical collection of data partitioned across machines. RDDs can be created by referencing datasets in external storage systems, or by applying coarse-grained transformations (e.g. map, filter, reduce, join) on existing RDDs.

   The RDD abstraction is exposed through a language-integrated API in Java, Python, Scala, and R similar to local, in-process collections. This simplifies programming complexity because the way applications manipulate RDDs is similar to manipulating local collections of data.

A Spark cluster is composed of one Driver JVM and one or many Executor JVMs.

2. **Spark SQL :**

   Spark SQL is a component on top of Spark Core that introduces a new data abstraction called DataFrames, which provides support for structured and semi-structured data. Spark SQL provides a domain-specific language to manipulate DataFrames in Scala, Java, or Python. It also provides SQL language support, with command-line interfaces and ODBC/JDBC server. Prior to version 1.3 of Spark, DataFrames were referred to as SchemaRDDs.

3. **Spark Streaming :**

   Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD transformations on those mini-batches of data. This design enables the same set of application code written for batch analytics to be used in streaming analytics, on a single engine.

3.2 Architecture of Spark

Spark also introduces an abstraction called resilient distributed datasets (RDDs). An RDD is a read-only collection of objects distributed across a set of nodes. These collections are resilient, because they can be rebuilt if a portion of the dataset is lost. The process of rebuilding a portion of the dataset relies on a fault-tolerance mechanism that maintains lineage (or information that allows the portion of the dataset to be re-created based on the process from which the data was derived).

An RDD is represented as a Scala object and can be created from a file; as a parallelized slice (spread across nodes); as a transformation of another RDD; and finally through changing the persistence of an existing RDD, such as requesting that it be cached in memory.

Applications in Spark are called drivers, and these drivers implement the operations performed either on a single node or in parallel across a set of nodes. Like Hadoop, Spark supports a single-node cluster or a multi-node cluster. For multi-node operation, Spark relies on the Mesos cluster manager. Mesos provides an efficient platform for resource sharing and isolation for distributed applications (see Figure 1). This setup allows Spark to coexist with Hadoop in a single shared pool of nodes.

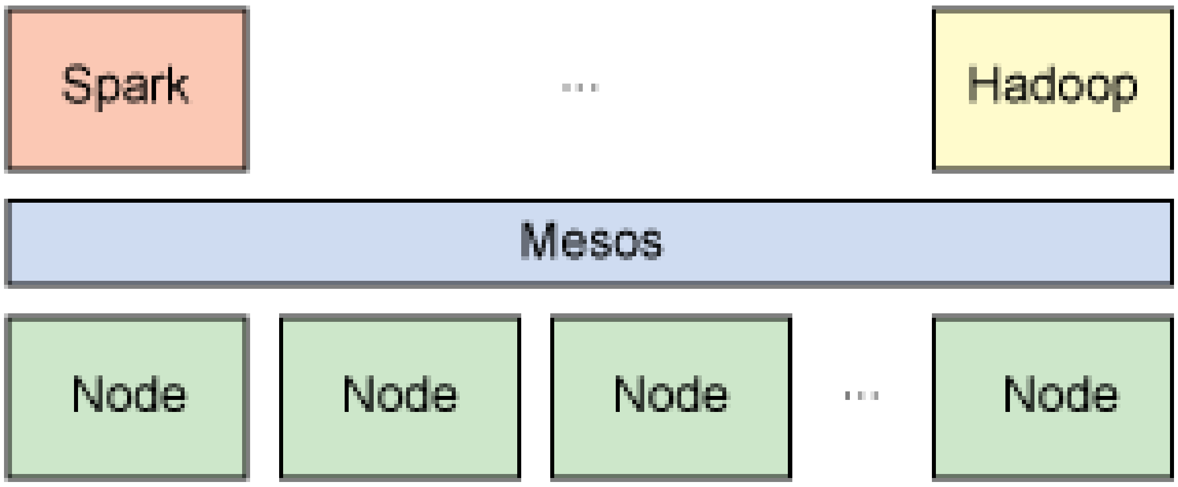

Figure 3.1: Spark relies on the Mesos cluster manager

3.2.1 Spark Programming Model

A driver can perform two types of operations on a dataset: an action and a transformation. An action performs a computation on a dataset and returns a value to the driver; a transformation creates a new dataset from an existing dataset. Examples of actions include performing a Reduce operation (using a function) and iterating a dataset (running a function on each element, similar to the Map operation). Examples of transformations include the Map operation and the Cache operation (which requests that the new dataset be stored in memory).

▪ *Map-Reduce*

Hadoop MapReduce is meant for data that does not fit in the memory whereas Apache Spark has a better performance for the data that fits in the memory, particularly on dedicated clusters.

Hadoop MapReduce can be an economical option because of Hadoop as a service offering(HaaS) and availability of more personnel. According to the benchmarks, Apache Spark is more cost effective but staffing would be expensive in case of Spark.

Spark and Hadoop MapReduce both have similar compatibility in terms of data types and data sources. Programming in Apache Spark is easier as it has an interactive mode whereas Hadoop MapReduce requires core java programming skills,however there are several utilities that make programming in Hadoop MapReduce easier.

- **Map Side of Hadoop Map Reduce :**

  Each Map task outputs the data in Key and Value pair. The output is stored in a CIRCULAR BUFFER instead of writing to disk. The size of the circular buffer is

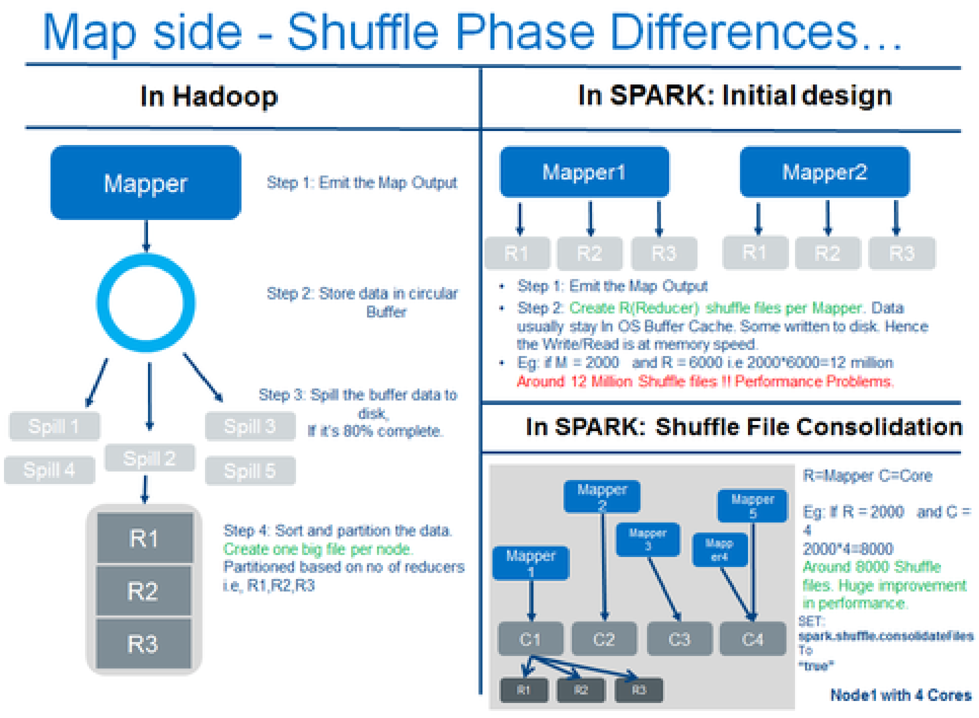
Map side - Shuffle Phase Differences…
In Hadoop
In SPARK: Initial design
Mapper
Step 1: Emit the Map Output
Step 2: Store data in circular Buffer
Step 3: Spill the buffer data to disk, If it's 80% complete.
Spill 1
Spill 2
Spill 3
Spill 4
Spill 5
R1
R2
R3
Step 4: Sort and partition the data. Create one big file per node. Partitioned based on no of reducers i.e, R1,R2,R3
Mapper1
Mapper2
R1
R2
R3
R1
R2
R3
Step 1: Emit the Map Output
Step 2: Create R(Reducer) shuffle files per Mapper. Data usually stay In OS Buffer Cache. Some written to disk. Hence the Write/Read is at memory speed.
Eg: if M = 2000 and R = 6000 i.e 2000*6000=12 million
Around 12 Million Shuffle files !! Performance Problems.
In SPARK: Shuffle File Consolidation
Mapper 1
Mapper 2
Mapper 3
Mapper4
Mapper 5
C1
C2
C3
C4
R1
R2
R3
R=Mapper C=Core
Eg: If R = 2000 and C = 4
2000*4=8000
Around 8000 Shuffle files. Huge improvement in performance.
SET: spark.shuffle.consolidatefiles To "true"
Node1 with 4 Cores

Figure 3.2: Map-Reduce in both framework

around 100 MB. If the circular buffer is 80% full by default, then the data will be spilled to disk, which are called shuffle spill files.

On a particular node, many map tasks are run as a result many spill files are created. Hadoop merges all the spill files, on a particular node, into one big file which is SORTED and PARTITIONED based on number of reducers.

- **Map side of Spark :**

  Initial Design:

  The output of map side is written to OS BUFFER CACHE. The operating system will decide if the data can stay in OS buffer cache or should it be spilled to DISK. Each map task creates as many shuffle spill files as number of reducers.

  SPARK doesn't merge and partition shuffle spill files into one big file, which is the case with Apache Hadoop. Example: If there are 6000 (R) reducers and 2000 (M) map tasks, there will be (M*R) 6000*2000=12 million shuffle files. This is because, in spark, each map task creates as many shuffle spill files as number of reducers. This caused performance degradation. This was the initial design of Apache Spark.

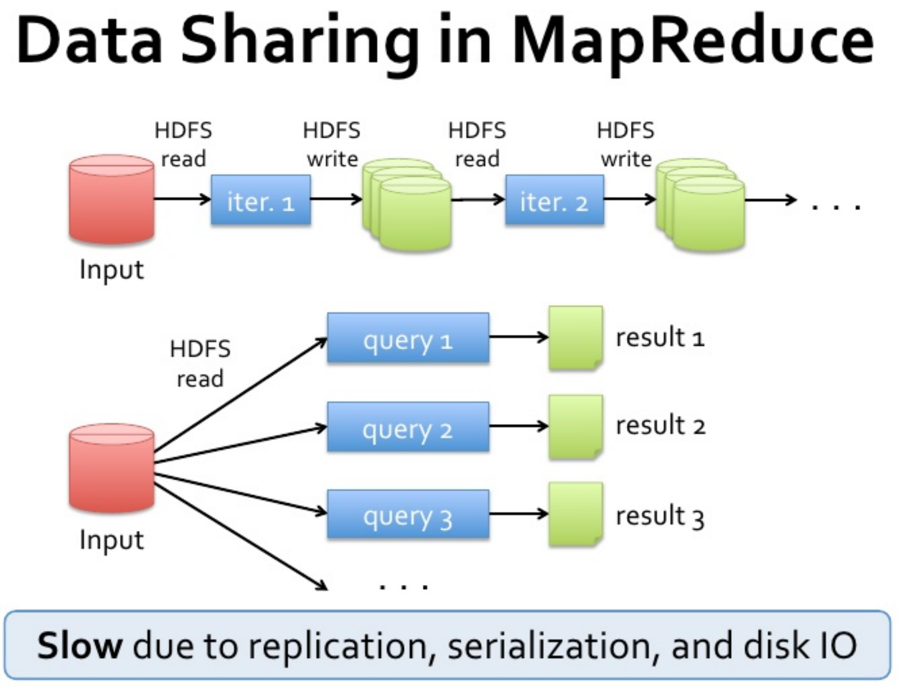

Figure 3.3: Data sharing in map-reduce

▮ *Resilient distributed datasets (RDDs)*

Resilient distributed datasets is a basic abstraction in Spark. It is an immutable, partitioned collection of elements that can be operated in parallel.There are three basic operations performed by the RDD - map, filter, persist. Custom RDD can also be implemented by overriding function. [4] RDD's characteristics :

- A list of partition
- A function of each split
- A list of dependencies on other RDD's

3.3 Working of Spark

3.4 Terms related to Spark

Job:- A piece of code which reads some input from HDFS or local, performs some computation on the data and writes some output data.

Stages:-Jobs are divided into stages. Stages are classified as a Map or reduce stages(Its easier to understand if you have worked on Hadoop and want to correlate). Stages are

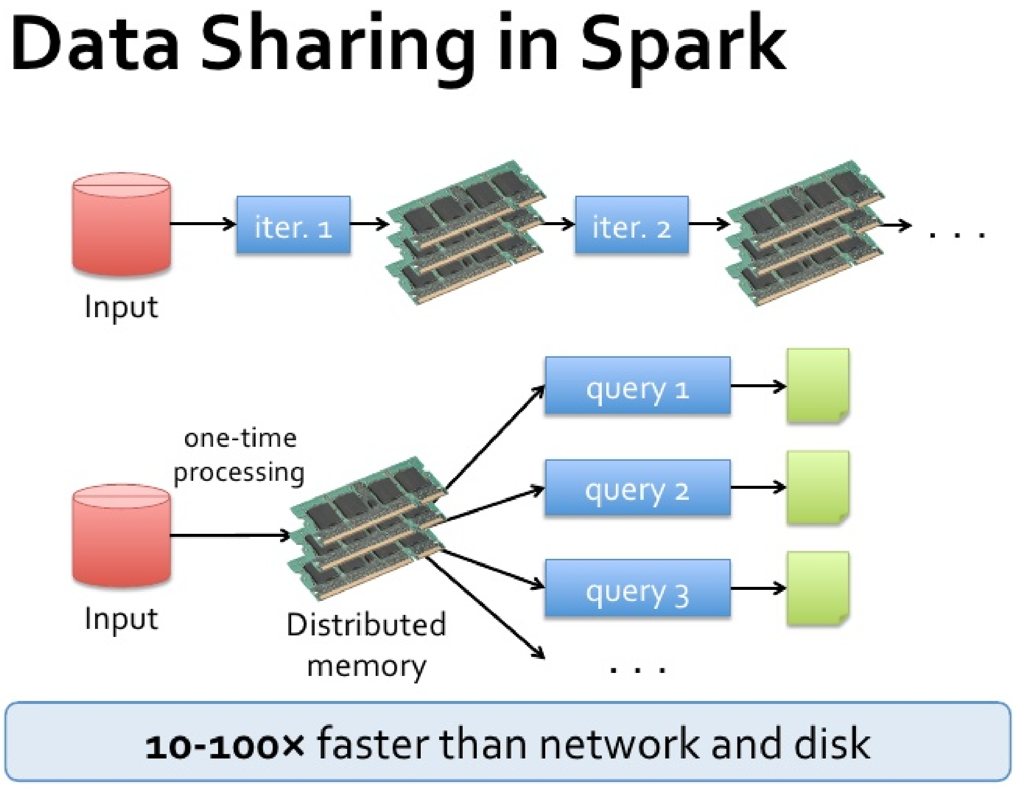

Figure 3.4: Data sharing in Spark

divided based on computational boundaries, all computations(operators) cannot be Updated in a single Stage. It happens over many stages.

Tasks:- Each stage has some tasks, one task per partition. One task is executed on one partition of data on one executor(machine).

DAG - DAG stands for Directed Acyclic Graph, in the present context its a DAG of operators.

Executor - The process responsible for executing a task.

Driver - The program or process responsible for running the Job over the Spark Engine

Master - The machine on which the Driver program runs

Slave - The machine on which the Executor program runs

3.4.1 Terminology

- *Spark context*

  It gives the connection between user's cluster and Spark's cluster.

- *Cluster Manager*

  It manages all resources required for clusters.

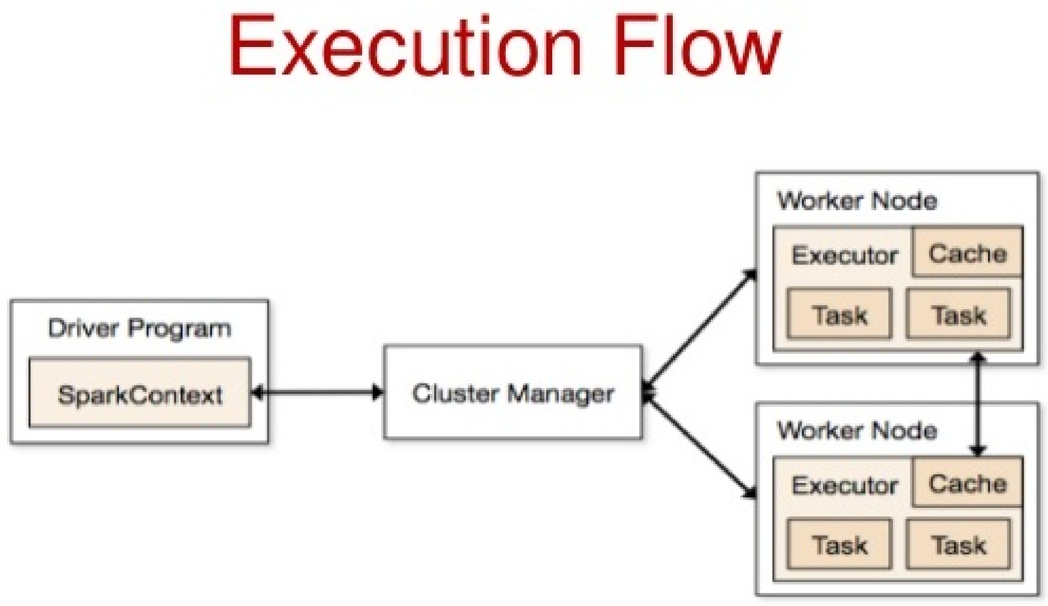

Figure 3.5: Execution Flow

- *Worker* It is simply an application program run on user's side.

[2] All jobs in spark comprise a series of operators and run on a set of data. All the operators in a job are used to construct a DAG (Directed Acyclic Graph). The DAG is optimized by rearranging and combining operators where possible. For instance lets assume that you have to submit a Spark job which contains a map operation followed by a filter operation. Spark DAG optimizer would rearrange the order of these operators, as filtering would reduce the number of records to undergo map operation.

Spark has a small code base and the system is divided in various layers. Each layer has some responsibilities. The layers are independent of each other.

The first layer is : the *interpreter*, Spark uses a Scala interpreter, with some modifications. As you enter your code in spark console(creating RDD's and applying operators), Spark creates a operator graph. When the user runs an action(like collect), the Graph is submitted to a *DAG Scheduler*. The DAG scheduler divides operator graph into (map and reduce) stages.

A *stage* is comprised of tasks based on partitions of the input data.
The DAG scheduler pipelines operators together to optimize the graph. For e.g. Many map operators can be scheduled in a single stage. This optimization is key to Sparks performance. The final result of a DAG scheduler is a set of stages. The stages are passed on to

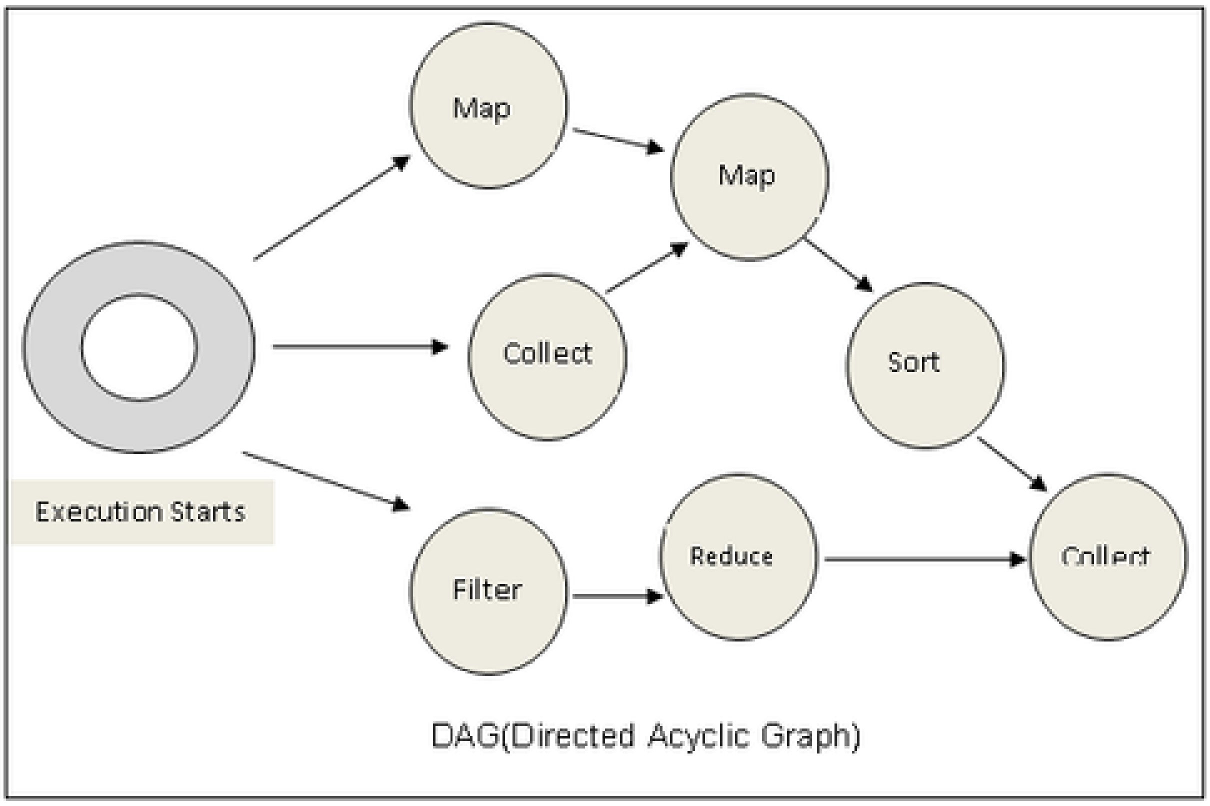

Figure 3.6: Directed Acyclic Graph

the *Task Scheduler*. The task scheduler launches tasks via cluster manager.(Spark Standalone/Yarn/Mesos). The task scheduler doesn't know about dependencies among stages.

The Worker executes the tasks on the Slave. A new JVM is started per JOB. The worker knows only about the code that is passed to it.

Spark caches the data to be processed, allowing it to me 100 times faster than hadoop. Spark uses Akka for Multithreading, managing executor state, scheduling tasks.

It uses Jetty to share files(Jars and other files), Http Broadcast, run Spark Web UI. Spark is highly configurable, and is capable of utilizing the existing components already existing in the Hadoop Eco-System. This has allowed spark to grow exponentially, and in a little time many organisations are already using it in production.

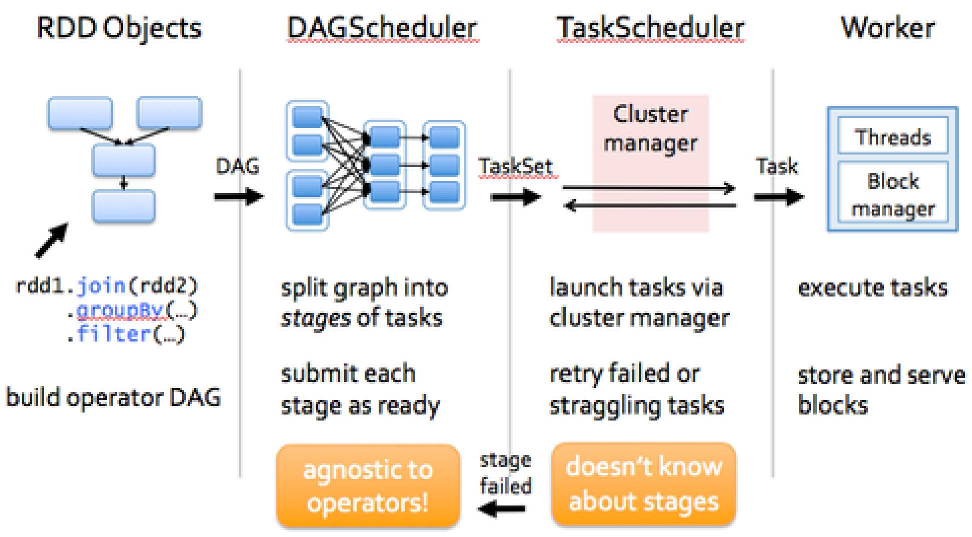

Figure 3.7: Layers in Spark

3.5 Summary

We have described the working of Spark in this chapter. Next we will visit some advantages, disadvantages of Spark.

Chapter 4

Key Points

According to its working, different components and multilayer functionality Spark brings some additional points in it. But with more work it has a little bit disadvantages. all these points along with applications are described here.

4.1 Difference between Hadoop and Spark

Differences between Hadoop and Spark on the basis of nodes involved, data size operated, rate of performance etc. [1]

Table 4.1: Differences of Hadoop and Spark

	Hadoop World Record	Spark 100 TB	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
Nodes	2100	206	190
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min

4.2 Advantages of Spark

- **Performance**:

  Spark performs better when all the data fits in the memory, especially on dedicated clusters; Hadoop MapReduce is designed for data that doesnt fit in the memory and it can run well alongside other services.

  The performance of Spark is better than Hadoop.

- **Speed**:

  Spark enables applications in Hadoop clusters to run up to 100x faster in memory, and 10x faster even when running on disk. Spark makes it possible by reducing number of read/write to disc. It stores this intermediate processing data in-memory. It uses the concept of an Resilient Distributed Dataset (RDD), which allows it to transparently store data on memory and persist it to disc only its needed. This helps to reduce most of the disc read and write the main time consuming factors of data processing.

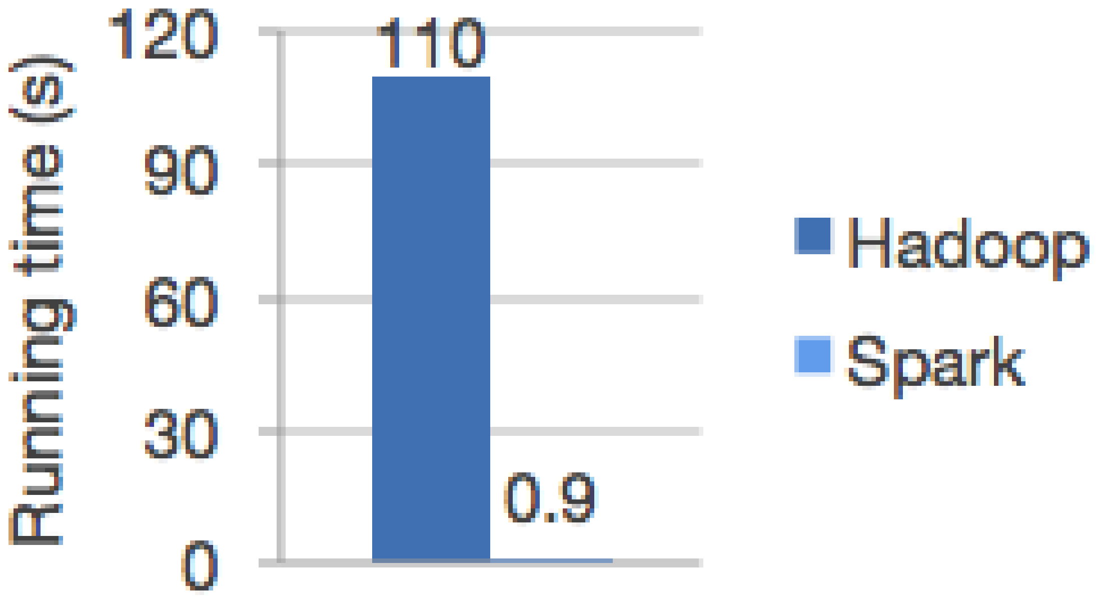

Figure 4.1: Speed of Spark

- **Runs Everywhere**:

  Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, S3.

- **Cost**:

  The memory in the Spark cluster should be at least as large as the amount of data you need to process, because the data has to fit into the memory for optimal performance. So, if you need to process really Big Data, Hadoop will definitely be the cheaper option since hard disk space comes at a much lower rate than memory space.

  It should be more cost-effective since less hardware can perform the same tasks much faster, especially on the cloud where compute power is paid per use. It depends on the hardware requirements of both the frameworks.

Table 4.2: Hardware Requirements

	Apache Spark (source)	Apache Hadoop workload slaves (source)
Cores	816	4
Memory	8 GB to hundreds of gigabytes	24 GB
Disks	48	46 one-TB disks

- **Faster**:

  Spark execute batch processing jobs , about 10 to 100 times faster than the Hadoop MapReduce framework by making the use of in-memory processing compared to persistence storage used by Hadoop. [3]

- **Caching**:

  One of the reason of Spark being extermely fast is by making use of caching and in-memory processing and Hadoop on the other hand is completely disk dependent.

- **Iterative computations**:

  One of the main aims for building Spark was iterative computations in use cases like machine learning when computation need to be performed multiple times on same set of data.

- **Recovery**:

  RDD is the main abstraction of spark. It allows recovery of failed nodes by re-computation of the DAG while also supporting a more similar recovery style to Hadoop by way of checkpointing, to reduce the dependencies of an RDD. Storing a spark job in a DAG allows for lazy computation of RDD's and can also allow spark's optimization engine to schedule the flow in ways that make a big difference in performance.

4.3 Disadvantages of Spark

- If a process crashes in the middle of execution, it could continue where it left off, whereas Spark will have to start processing from the beginning.
- It is more secure than hadoop but still RDD creates some problems.

4.4 User Applications

Spark is in use both at several Internet companies and in a number of machine learning research projects at Berkeley. Some of our users applications are discussed here :

1. **In-Memory Analytics on Hive Data**:

   Conviva Inc., an online video distribution company, used Spark to accelerate a number of analytics reports that previously ran over Hive, the SQL engine built on Hadoop. For example, a report on viewership across different geographical regions went from taking 24 hours with Hadoop to only 45 minutes with Spark (30 faster) by loading the subset of data of interest into memory and then sharing it across queries. [5]

Conviva is also using Spark to interactively search large collections of log files and troubleshoot problems, using both the Scala interface and a SQL interface called Shark.

2. **Interactive Queries on Data Streams**:

   Quantifind, a startup that specializes in predictive analytics over time series data, used Spark to build an interactive interface for exploring time series. The system periodically loads new data from external feeds (e.g., Twitter streams), runs an entity extraction algorithm to parse the data, and builds an in-memory table of mentions of each entity. Users can then query this table interactively through a Web application that runs Spark queries on the backend .

3. **Traffic Modeling**:

   Researchers in the Mobile Millennium project at Berkeley [3] parallelized a learning algorithm for inferring traffic conditions from crowd-sourced automobile GPS measurements. The source data were a 10,000 link road network for the San Francisco area, as well as 600,000 position reports for GPS-equipped automobiles (e.g., taxi cabs, or users running a mobile phone application) collected every minute. By applying an iterative expectation maximization (EM) algorithm to this data, the system can infer the time it takes to travel across individual road links.

4. **Twitter Spam Detection**:

   The Monarch project at Berkeley used Spark to identify link spam in Twitter posts. They implemented a logistic regression classifier on top of Spark, similar to the example in Other Examples, above. They applied it to over 80 GB of data containing 400,000 URLs posted on Twitter and 107 features/dimensions related to the network and content properties of the pages at each URL to develop a fast and accurate classifier for spammy links.

4.5 Summary

We have touched up some of the advantages, disadvantages of Spark. we have also seen few applications of Spark.

Conclusion

Conclusion:

We should look at Hadoop as a general purpose Framework that supports multiple models and We should look at Spark as an alternative to Hadoop MapReduce rather than a replacement to Hadoop.

Spark speeds up your interactive and complex analytics on Hadoop data.Big data analytics is evolving to include: More complex analytics (e.g. machine learning) More interactive ad-hoc queries More real-time stream processing.

Spark is a platform that unifies these models, enabling sophisticated apps Spark can be good alternative for Hadoop but it cannot be replacement to it.

Bibliography

[1] Spark the fastest open source engine, Reynold Xin, October 5, 2014.

[2] http//www.spark-project.org

[3] Fast and Interactive Analytics, MATEI ZAHARIA, MOSHARAF CHOWDHURY, TATHAGATA DAS, ANKUR DAVE, JUSTIN MA, MURPHY MCCAULEY, AUGUST 2012.

[4] Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, USENIX NSDI, 2012.

[5] D. Joseph, Using Spark and Hive to Process Big Data at Conviva: http://www.conviva.com/blog/engineering/using-spark-and-hive-to-process-bigdata -at-conviva.