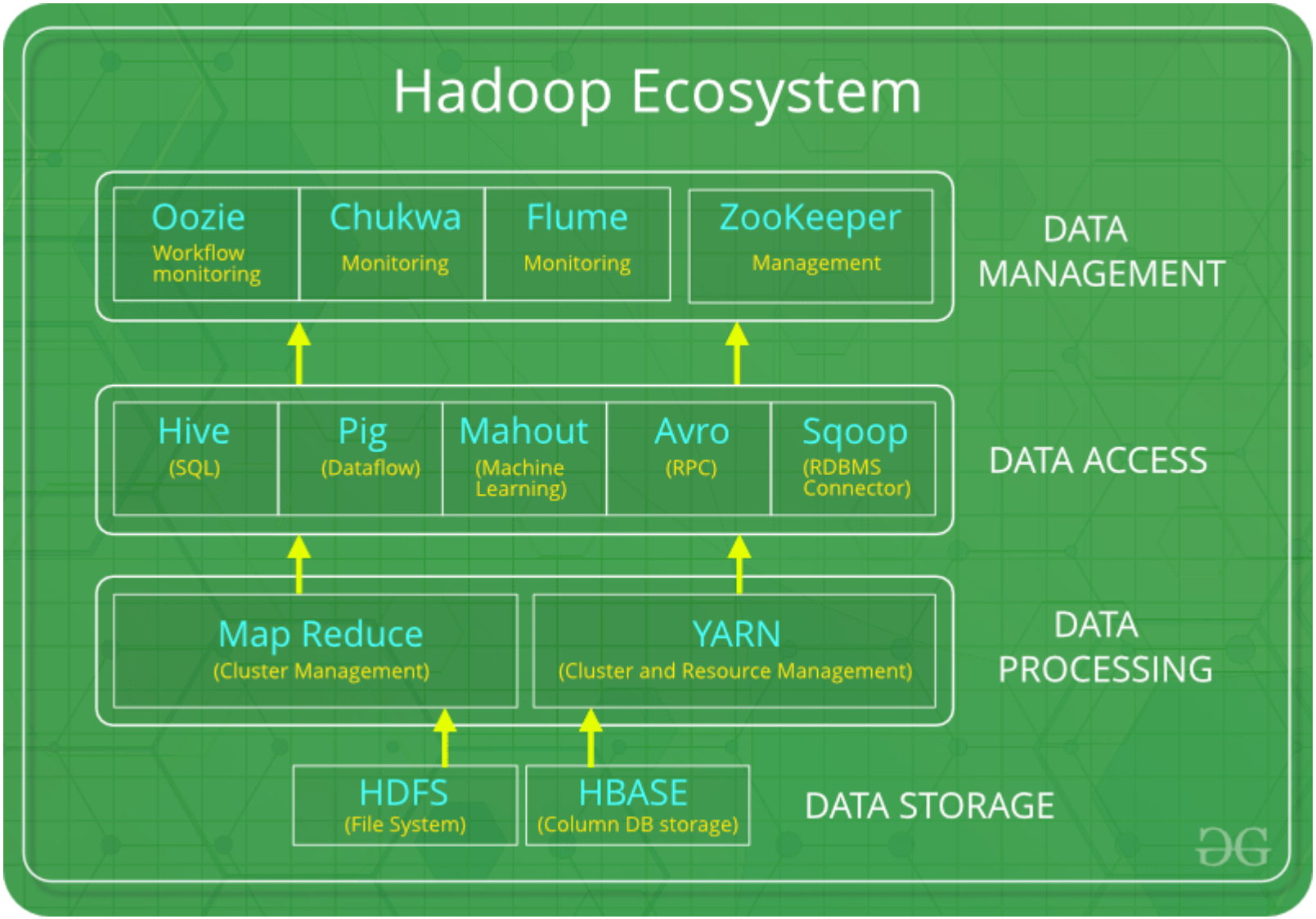
What is Hadoop?

Apache Hadoop is an open source software framework used to develop data processing applications which are executed in a distributed computing environment.

Applications built using HADOOP are run on large data sets distributed across clusters of commodity computers. Commodity computers are cheap and widely available. These are mainly useful for achieving greater computational power at low cost.

Similar to data residing in a local file system of a personal computer system, in Hadoop, data resides in a distributed file system which is called as a **Hadoop Distributed File system**. The processing model is based on **'Data Locality'** concept wherein computational logic is sent to cluster nodes (server) containing data. This computational logic is nothing, but a compiled version of a program written in a high-level language such as Java. Such a program, processes data stored in Hadoop HDFS.

Hadoop Ecosystem is a platform or a suite which provides various services to solve the big data problems.


Hadoop Ecosystem
Oozie
Workflow monitoring
Chukwa
Monitoring
Flume
Monitoring
ZooKeeper
Management
DATA MANAGEMENT
Hive
(SQL)
Pig
(Dataflow)
Mahout
(Machine Learning)
Avro
(RPC)
Sqoop
(RDBMS Connector)
DATA ACCESS
Map Reduce
(Cluster Management)
YARN
(Cluster and Resource Management)
DATA PROCESSING
HDFS
(File System)
HBASE
(Column DB storage)
DATA STORAGE

What is HDFS?

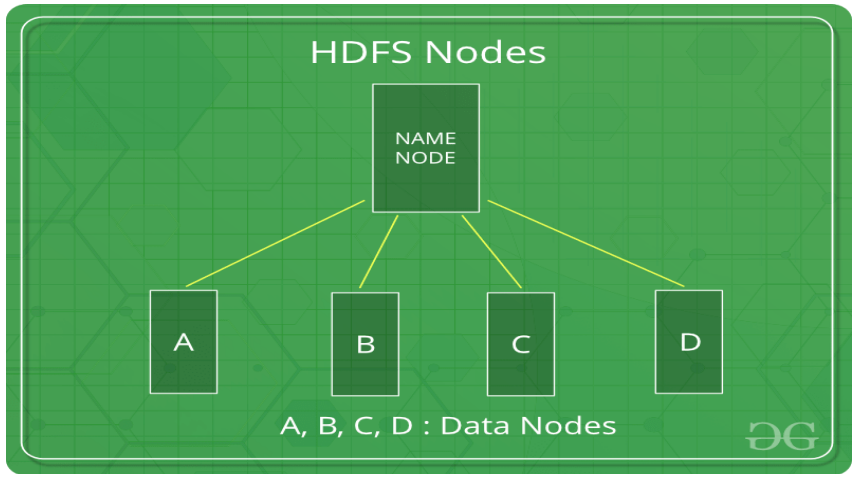
HDFS is a distributed file system for storing very large data files, running on clusters of commodity hardware. It is fault tolerant, scalable, and extremely simple to expand. Hadoop comes bundled with **HDFS** (**Hadoop Distributed File Systems**).

HDFS:

- HDFS is the primary or major component of Hadoop ecosystem and is responsible **for storing large data sets** of structured or unstructured data across various nodes and thereby maintaining the metadata in the form of log files.

  When **data exceeds the capacity** of storage on a single physical machine, it becomes essential to divide it across a number of separate machines. A file system that manages storage specific operations across a network of machines is called a distributed file system. HDFS is one such software.

- HDFS consists of two core components i.e.
  1. Name node
  2. Data Node



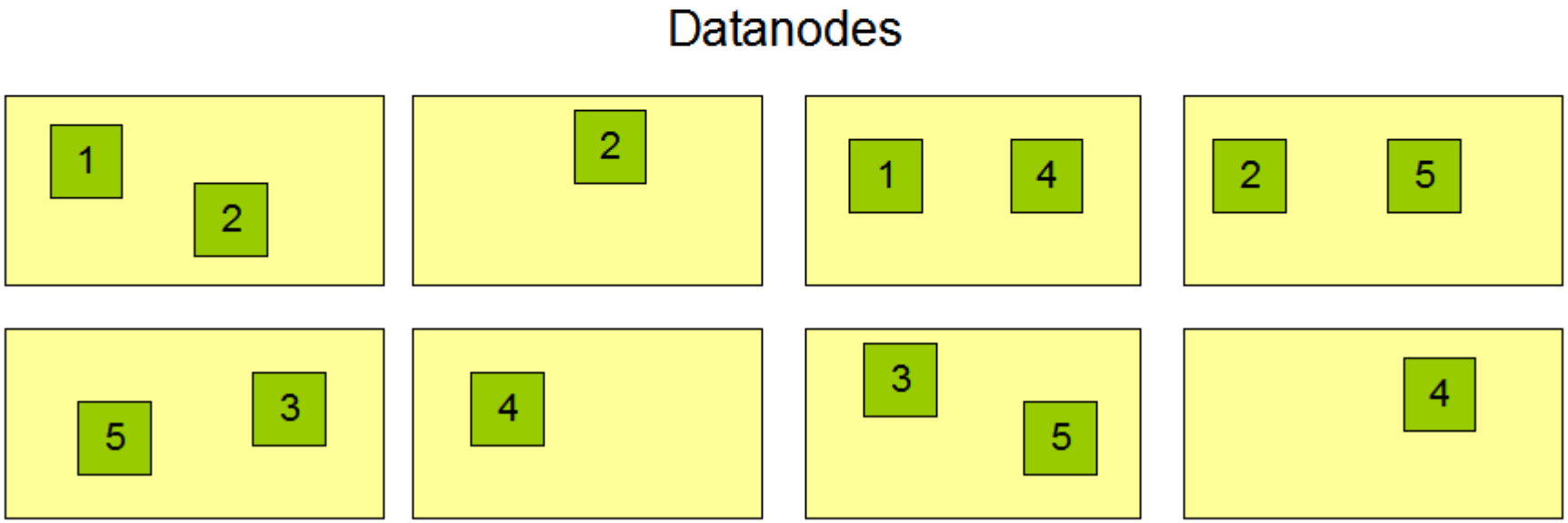
- Name Node is the prime node which contains metadata (data about data) requiring comparatively fewer resources than the data nodes that stores the actual data. These data nodes are commodity hardware in the distributed environment. Undoubtedly, making Hadoop cost effective.
- HDFS maintains all the coordination between the clusters and hardware, thus working at the heart of the system.

HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software.

A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

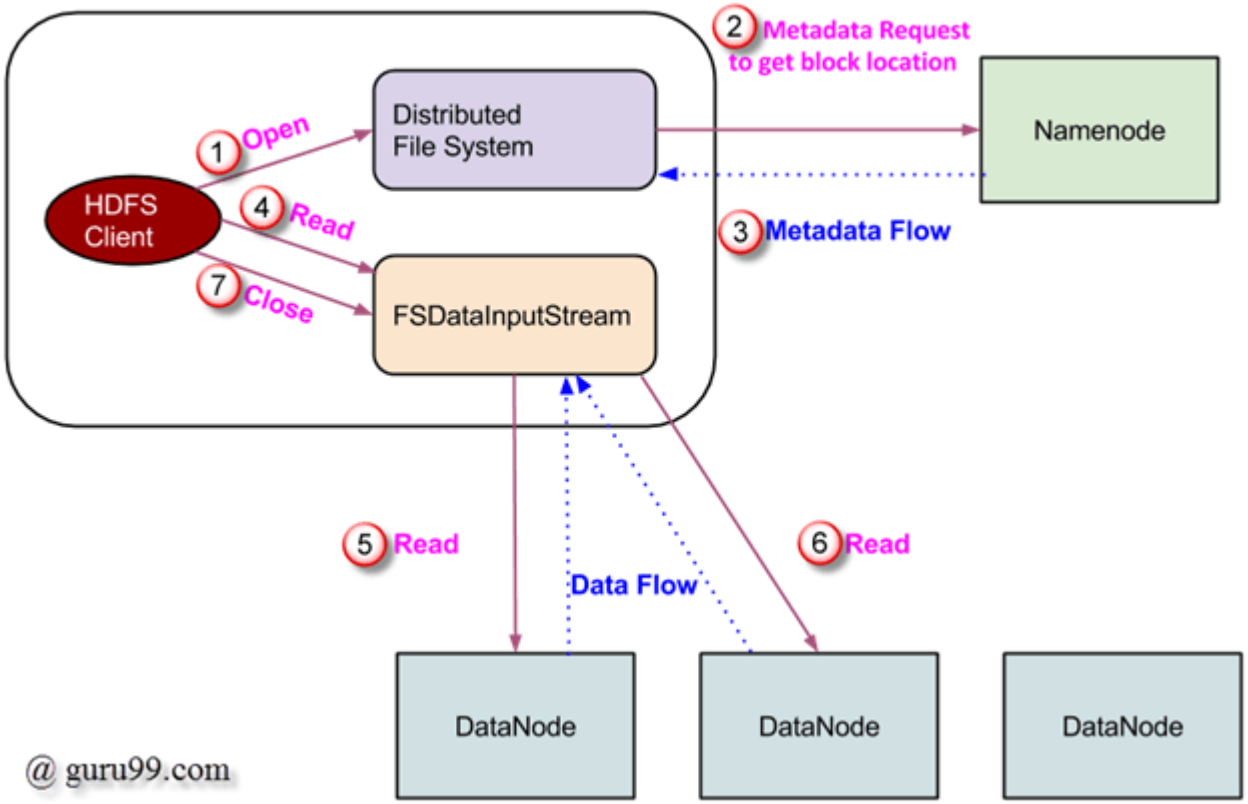
Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file.

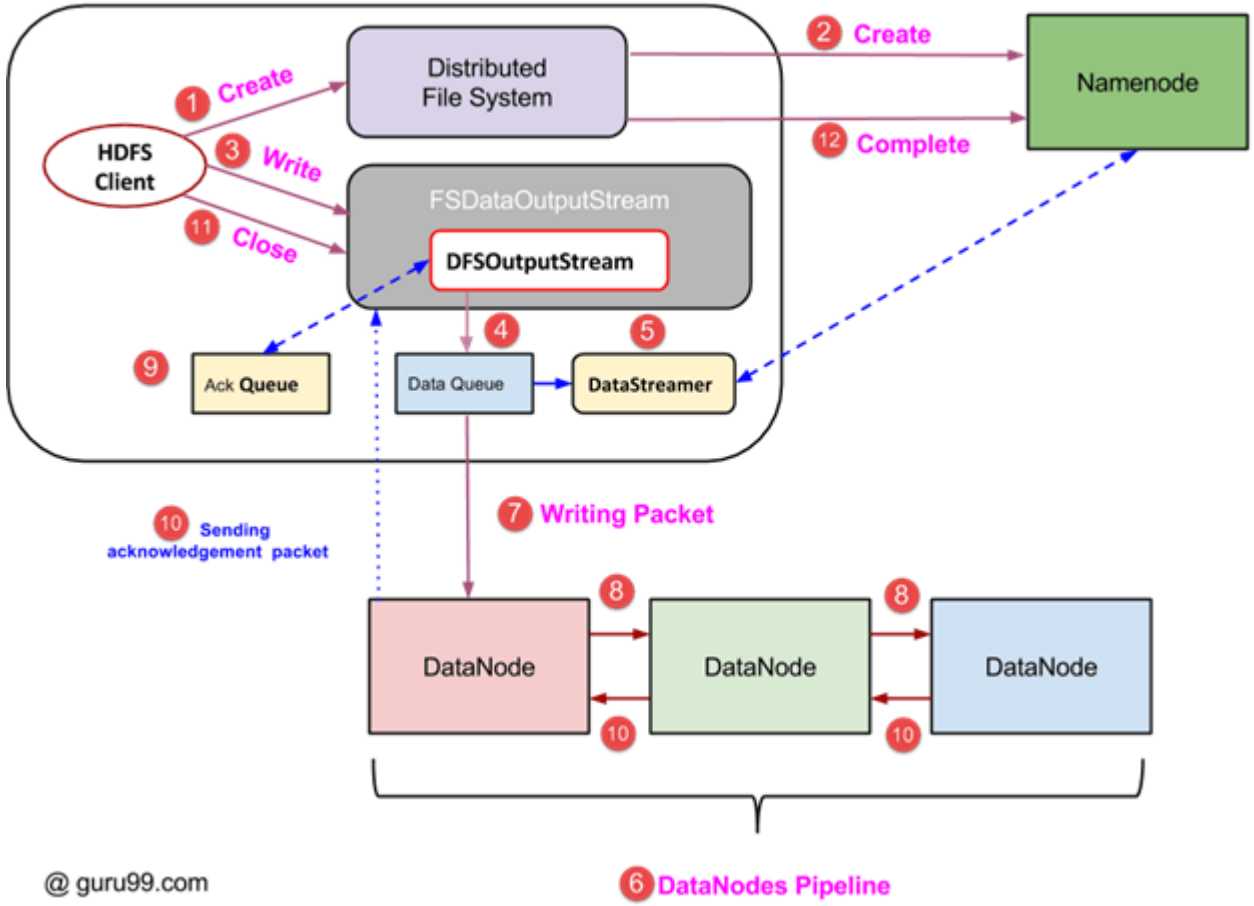


Read Operation In HDFS

Data read request is served by HDFS, NameNode, and DataNode. Let's call the reader as a 'client'. Below diagram depicts file read operation in Hadoop.


2 Metadata Request
to get block location
Distributed
File System
Namenode
1 Open
HDFS
Client
4 Read
3 Metadata Flow
7 Close
FSDataInputStream
5 Read
6 Read
Data Flow
DataNode
DataNode
DataNode
@ guru99.com

Write Operation In HDFS



MapReduce:

- By making the use of distributed and parallel algorithms, MapReduce makes it possible to carry over the processing's logic and helps to write applications which transform big data sets into a manageable one.

MapReduce makes the use of two functions i.e. Map() and Reduce() whose task is:

1. **Map()** performs sorting and filtering of data and thereby organizing them in the form of group. Map generates a key-value pair based result which is later on processed by the Reduce() method.
2. **Reduce(),** as the name suggests does the summarization by aggregating the mapped data. In simple, Reduce() takes the output generated by Map() as input and combines those tuples into smaller set of tuples.

Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

The input to each phase is **key-value** pairs. In addition, every programmer needs to specify two functions: **map function** and **reduce function**.

How MapReduce Works? Complete Process

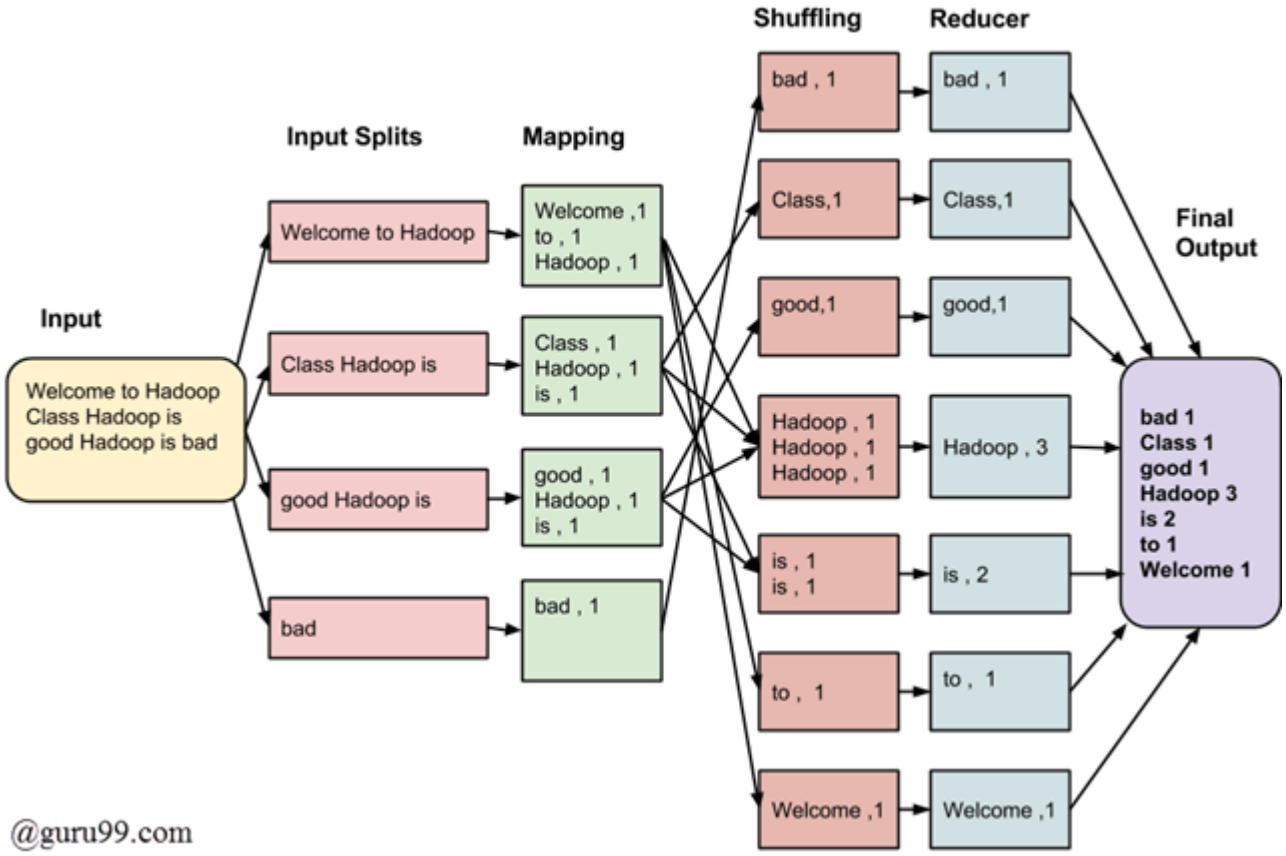
The whole process goes through four phases of execution namely, splitting, mapping, shuffling, and reducing.

Let's understand this with an example –

Consider you have following input data for your Map Reduce Program

```
Welcome to Hadoop Class
```

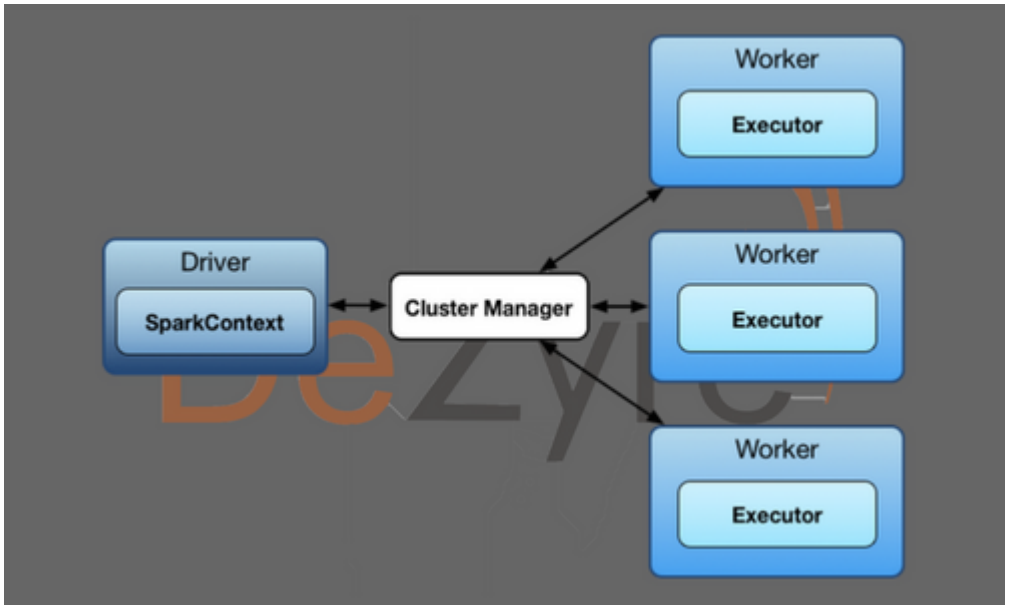
```
Hadoop is good
Hadoop is bad
```



Example Usecase:

Sales related information like Product name, price, payment mode, city, country of client etc. The goal is to ***find out Number of Products Sold in Each Country.***

Apache Spark Architecture



Apache Spark has a well-defined and layered architecture where all the spark components and layers are loosely coupled and integrated with various extensions and libraries. Apache Spark Architecture is based on two main abstractions-

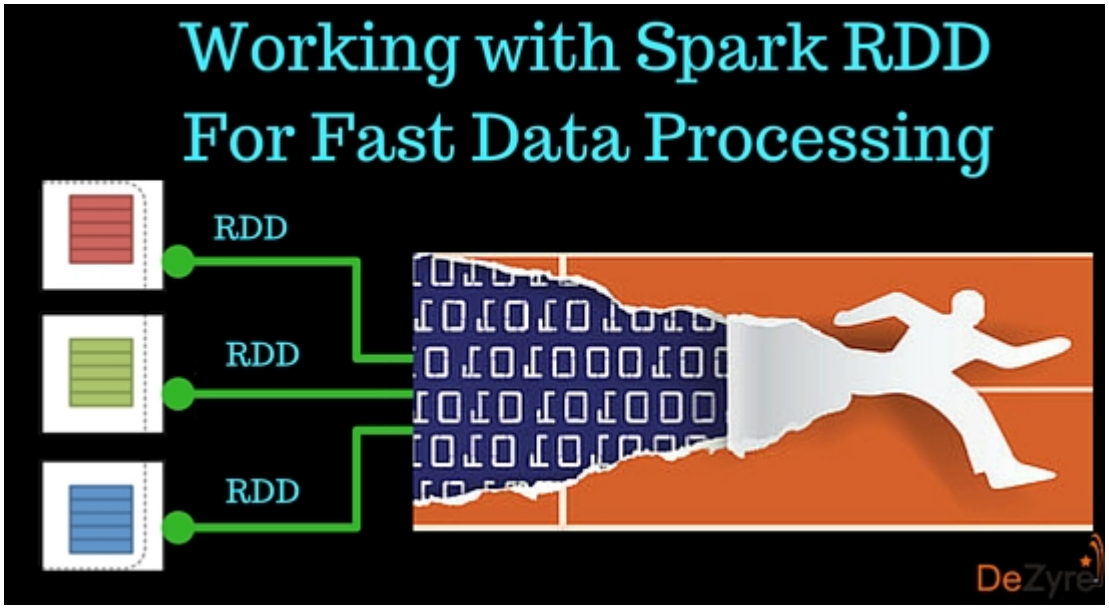
- Resilient Distributed Datasets (RDD)
- Directed Acyclic Graph (DAG)

RDD's are collection of data items that are split into partitions and can be stored in-memory on workers nodes of the spark cluster. In terms of datasets, apache spark supports two types of RDD's – Hadoop Datasets which are created from the files stored on HDFS and parallelized collections which are based on existing Scala collections. Spark RDD's support two different types of operations – Transformations and Actions.

DAG

This sequence of commands implicitly defines a DAG of RDD objects (RDD lineage) that will be used later when an action is called. Each RDD maintains a pointer to one or more parents along with the metadata about what type of relationship it has with the parent. For example, when we call val b = a.map() on a RDD, the RDD b keeps a reference to its parent a, that's a lineage.

Hadoop MapReduce well supported the batch processing needs of users but the craving for more flexible developed big data tools for real-time processing, gave birth to the big data darling Apache Spark. Spark is setting the big data world on fire with its power and fast data processing speed.



The centre of attraction for Apache Spark is fast data processing speed and its ability to handle event streaming. These features make the open source superstar Apache Spark the sweetheart of the big data world. All thanks to the main abstractions- Resilient Distributed Datasets which have made Apache Spark the big data darling of many enterprises. Spark RDD is the bread and butter of the Apache Spark ecosystem and to learn Spark - mastering the concepts of apache spark RDD is extremely important.

What are Resilient Distributed Datasets (RDDs)?

According to the original paper -Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, RDDs in Spark can be defined as follows-

Resilient Distributed Datasets (RDDs) are a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner.

Resilient
Meaning it provides fault tolerance through lineage graph. A lineage graph keeps a track of transformations to be executed after an action has been called. RDD lineage graph helps recomputed any missing or damaged partitions because of node failures.

Distributed
RDDs are distributed - meaning the data is present on multiple nodes in a cluster.

Datasets
Collection of partitioned data with primitive values.

Resilient Distributed Datasets

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes. Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

There are two ways to create RDDs − parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

Data Sharing is Slow in MapReduce

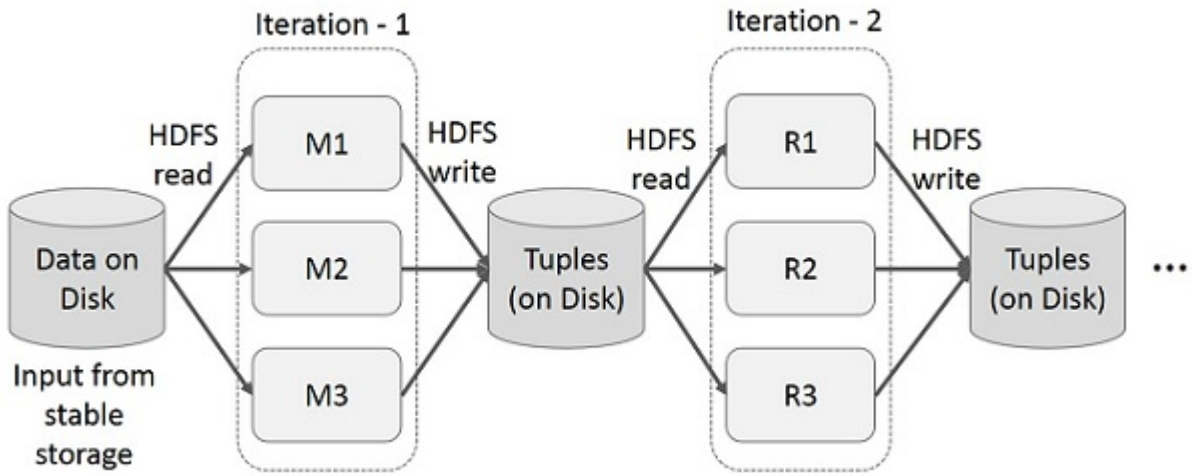
MapReduce is widely adopted for processing and generating large datasets with a parallel, distributed algorithm on a cluster. It allows users to write parallel computations, using a set of high-level operators, without having to worry about work distribution and fault tolerance.

Unfortunately, in most current frameworks, the only way to reuse data between computations (Ex − between two MapReduce jobs) is to write it to an external stable storage system (Ex − HDFS). Although this framework provides numerous abstractions for accessing a cluster's computational resources, users still want more.

Both Iterative and Interactive applications require faster data sharing across parallel jobs. Data sharing is slow in MapReduce due to replication, serialization, and disk IO. Regarding storage system, most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

Iterative Operations on MapReduce

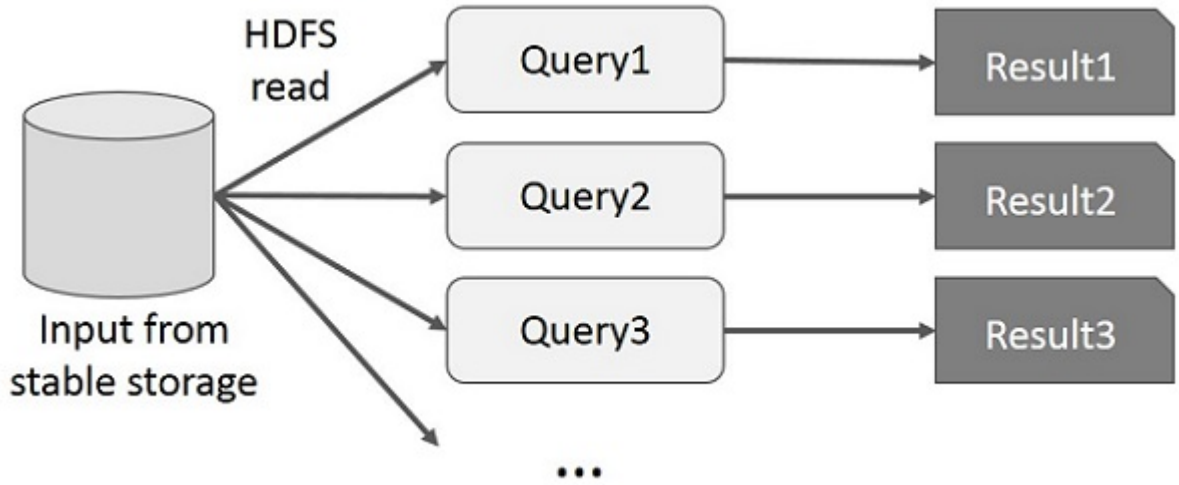
Reuse intermediate results across multiple computations in multi-stage applications. The following illustration explains how the current framework works, while doing the iterative operations on MapReduce. This incurs substantial overheads due to data replication, disk I/O, and serialization, which makes the system slow.



Interactive Operations on MapReduce

User runs ad-hoc queries on the same subset of data. Each query will do the disk I/O on the stable storage, which can dominate application execution time.

The following illustration explains how the current framework works while doing the interactive queries on MapReduce.



Data Sharing using Spark RDD

Data sharing is slow in MapReduce due to replication, serialization, and disk IO. Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

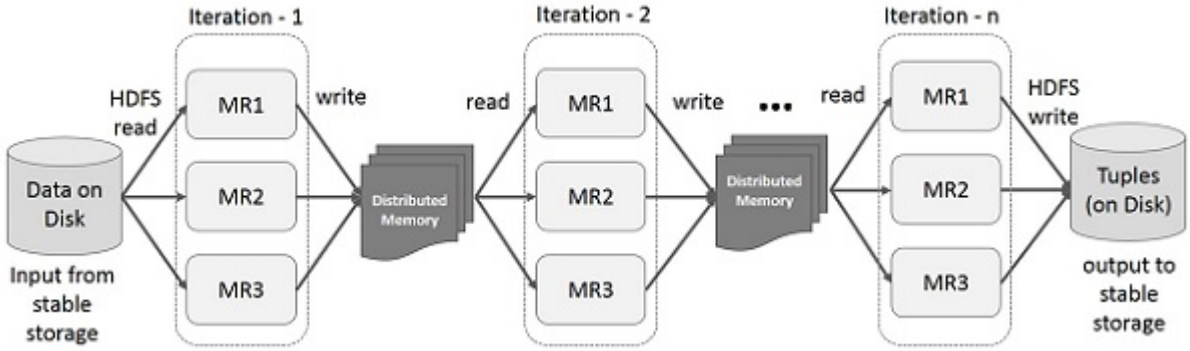
Recognizing this problem, researchers developed a specialized framework called Apache Spark. The key idea of spark is Resilient Distributed Datasets (RDD); it supports in-memory processing computation. This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs. Data sharing in memory is 10 to 100 times faster than network and Disk.

Let us now try to find out how iterative and interactive operations take place in Spark RDD.

Iterative Operations on Spark RDD

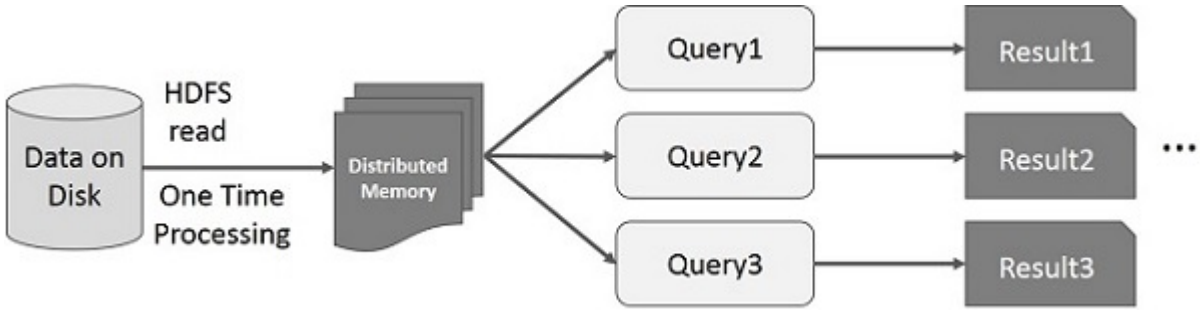
The illustration given below shows the iterative operations on Spark RDD. It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

Note − If the Distributed memory (RAM) is not sufficient to store intermediate results (State of the JOB), then it will store those results on the disk.



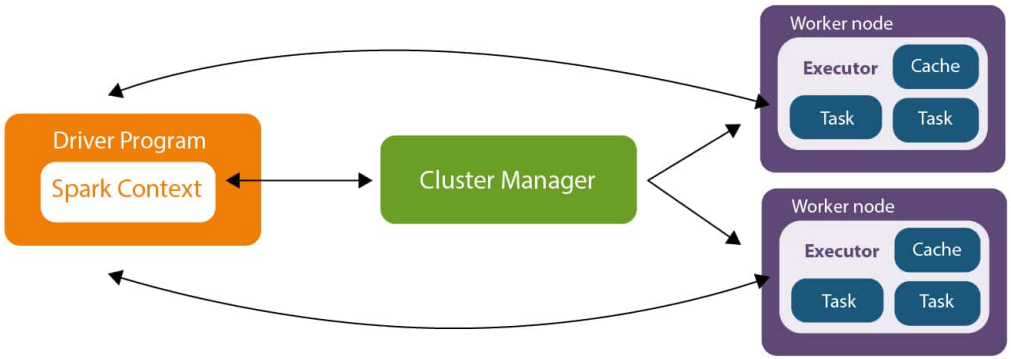
Interactive Operations on Spark RDD

This illustration shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.



By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also persist an RDD in memory, in which case Spark will keep the elements around on the cluster for much faster access, the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

Working of the Apache Spark Architecture



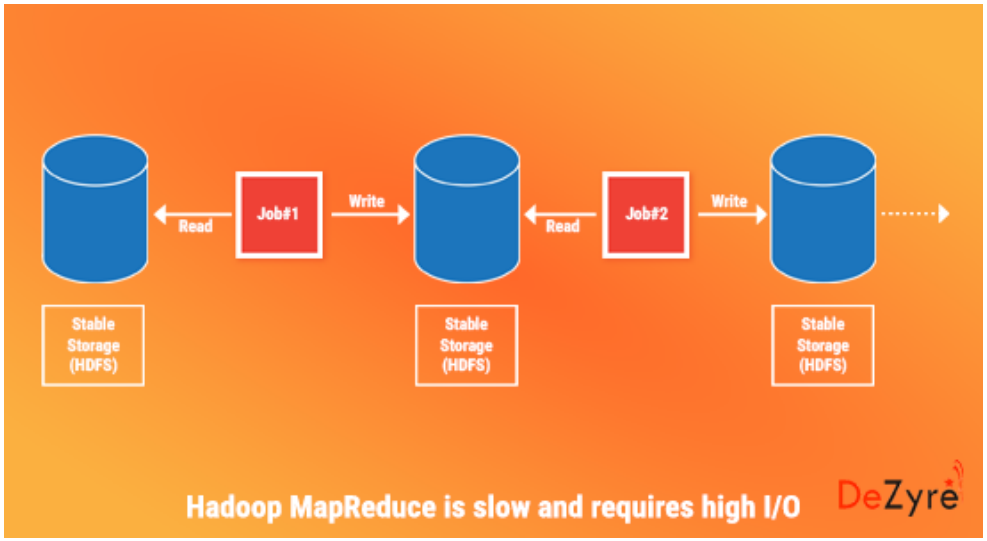
Driver Program in the Apache Spark architecture calls the main program of an application and creates SparkContext. A SparkContext consists of all the basic functionalities. Spark Driver contains various other components such as DAG Scheduler, Task Scheduler, Backend Scheduler, and Block Manager, which are responsible for translating the user-written code into jobs that are actually executed on the cluster.

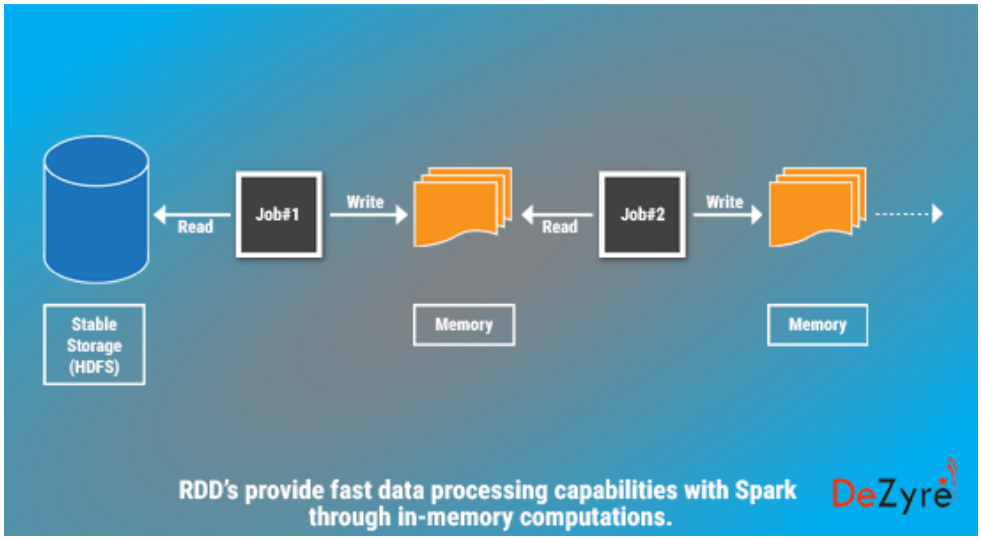
Spark Driver and SparkContext collectively watch over the job execution within the cluster. Spark Driver works with the Cluster Manager to manage various other jobs. Cluster Manager does the resource allocating work. And then, the job is split into multiple smaller tasks which are further distributed to worker nodes.
Whenever an RDD is created in the SparkContext, it can be distributed across many worker nodes and can also be cached there.
Worker nodes execute the tasks assigned by the Cluster Manager and return it back to the Spark Context.
An executor is responsible for the execution of these tasks. The lifetime of executors is the same as that of the Spark Application. If we want to increase the performance of the system, we can increase the number of workers so that the jobs can be divided into more logical portions.

Cluster Managers

The SparkContext can work with various Cluster Managers, like Standalone Cluster Manager, Yet Another Resource Negotiator (YARN), or Mesos, which allocate resources to containers in the worker nodes. The work is done inside these containers.

Mapreduce Vs. Spark





Install Java, hadoop
https://www.google.com/search?q=install+hadoop+in+windows&rlz=1C1CHZL_enIN837IN837&oq=install+hadoop+in+window&aqs=chrome.0.0i457j69i57j0i22i30l6.8490j0j15&sourceid=chrome&ie=UTF-8#kpvalbx=_ecasX7rZLJGGmgfw9ZT4BQ22

Install java jdk 8
Install hadoop 3.2.1

Install pyspark
Spark 3.0.1

https://changhsinlee.com/install-pyspark-windows-jupyter/

Word Count Implementation Video Reference
https://www.youtube.com/watch?v=jg7Z8ctKpEs