**Hadoop**

Hadoop is an “approach” to data processing rather than a single, monolithic framework(it is more like an ecosystem). It is mainly used to analyze large *amounts* and *types* of data. This data typically is:

1. Unstructured ( in contrast to RDBMS data which are organized and well structured).
2. Massive ranging up to 1015 byte range.

Core components of Hadoop are:

1. Hadoop Distributed File System(HDFS) - this is where the data is stored. The data is stored across multiple servers.
2. Yet Another Resource Negotiator(YARN) - this is a resource manager system that manages the Hadoop cluster, scheduling jobs and managing resources.

**What is Big Data ?**

The “Big Data” encompasses more than just a large *volume* of data. Other characteristics of Big Data include:

1. Variety - the data may come from a variety of sources and they sources may not necessarily be related to one another.
2. Velocity - how fast is the data generated and how fast can it be processed ?
3. Variability - the data may be highly varied, incomplete, unstructured, and even inconsistent with one another.

Hadoop’s approach to Big Data can be described as a “Data lake”. Traditional databases have a *predetermined schema* that the data must be organized into. So when new data is added, there is a 3 step process involved:

1. Extract - extract the necessary information from raw data
2. Transform - convert it to a form that can be stored into a database
3. Load - finally, load the transformed data.

This 3-step process, *ETL*, falls under the *schema on write* paradigm in which the developers already have designed a system and have made some assumptions on how the data will be used.

In contrast, Hadoop and Data Lake approach follows a *schema on read* paradigm. There are no prior assumptions on how data will be used so all the raw data is dumped into central repository called *Data Lake*. So when an application needs data, something similar to ETL happens on the fly where the programmer defines the structure of the data is needed. As a result, Hadoop is considered *schema on read* because users can define schemas that fit their current requirement.

So the Data Lake approach has the following advantages:

1. All data is readily available on its raw form. There are no assumptions made about the structure or uses of the data.
2. Data is highly available to everyone.

**MapReduce**

MapReduce is a two-step process that involves a *mapping* step followed by a *reducing* step. MapReduce is independent of Hadoop Distributed File System(HDFS) but in the context of Hadoop, it is best when used together. So before carrying out a MapReduce on Hadoop, there is the preliminary step of loading the raw data into HDFS. When the raw data is copied over to HDFS, the first thing that happens is that the data gets split among various nodes(or computers/servers) of the Hadoop cluster. Each slice of data is part of the whole set. It is important to note this splitting is *only* physical - from a logical standpoint of a user, the uploaded data appears as a single pile and the data itself is unaffected. For example if the user uploads a single text file, the text within the file gets split among the nodes BUT if the user were to run the “ls” command, they would only see the single file.

Once the raw data is in HDFS, the MapReduce algorithm can be applied to a user query. The Map step applies a query to each individual slice of data independently to produce a set of output from each slice. Then the Reduce step takes the set of outputs from the slices and combine them into single piece of meaningful data.

*Example:* Find the number of times the term “Kutuzov” appears in the book *War and Peace*(one of the longest novels ever written).

*Step 0* - Load text of *War and Peace* into HDFS.

HDFS will split the text among its nodes although to a user the novel is singular coherent body of text.

*Step 1* - The Map step

Our query is the frequency of the term “Kutuzov”. This query will be mapped independently to each of the text slices. Result will be a set of outputs such as,

(slice1, 12)

(slice2, 23)

(slice3, 45)

.

.

.

(sliceN, X)

Where the first value is the text slice and the second value is the frequency of the word “Kutuzov” in that piece of the text.

*Step 2* - The Reduce step

Next, take each of the output from the Map step and Reduce it into a meaningful value(relative to the original query). In this case, the Reduce step will add up value of each slice to get the grand total, which is the answer to our original query.

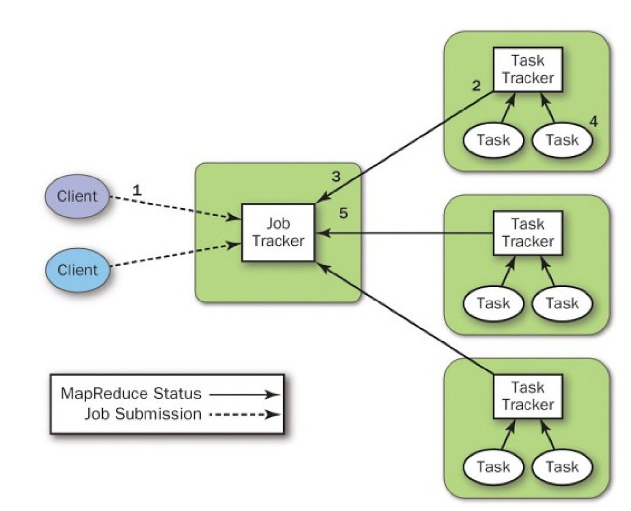
Answer = 12 + 23 + 45 + …. + X

MapReduce process has the following benefits:

1. MapReduce process is *functional*- the data is fed into the process so original data is not altered in anyway.
2. *Highly Scalable* - if we have more nodes, then the problem can be solved even faster because each node would then have less work to do. The process is linearly scalable.
3. *Fault Tolerant* - if one or more nodes fail, the process does NOT have to be restarted. It will be slower but complete failure is avoidable as long as even a single node is available.

**How MapReduce works on a cluster(in Hadoop V1)**

Following is a description of the version 1 Hadoop which is NOT used anymore.



There is a single *Job Tracker* object to schedule, launch, and track all MapReduce jobs on the cluster. The Job Tracker can run on its own cluster node or with something else; it doesn’t matter.

*Task Tracker* nodes are where the actual work gets done. Each Task Tracker gets jobs from the Job Tracker and manages its own subjobs. These subjobs can be the Map step, Reduce step, or some HDFS task since these nodes can also store data(they are also HDFS nodes).

Typical workflow is as follows:

1. Client submits a MapReduce job to the JobTracker. The JobTracker only accepts MapReduce jobs.
2. The JobTracker schedules and allocates the job to the cluster nodes. Preference is usually given to nodes that contain the data that needs to be worked on.
3. The JobTracker tracks the work done Task Tracker nodes and allocates resources as nodes become free.

V1 was poor because a Job Tracker failure can kill the entire process and some other preset limitations to the node size.

**Hadoop V2**

To address the issues with V1 of Hadoop, V2 was designed with the JobTracker functionality separated into different components. Under the new system,

1. YARN is used to carry out scheduling and resource management separate from the cluster.
2. MapReduce is a service that asks YARN for cluster nodes to service a query. So MapReduce runs on top of YARN.
3. YARN is also a *generalized framework* that almost any application written in any programming language can use to indirectly use Hadoop.

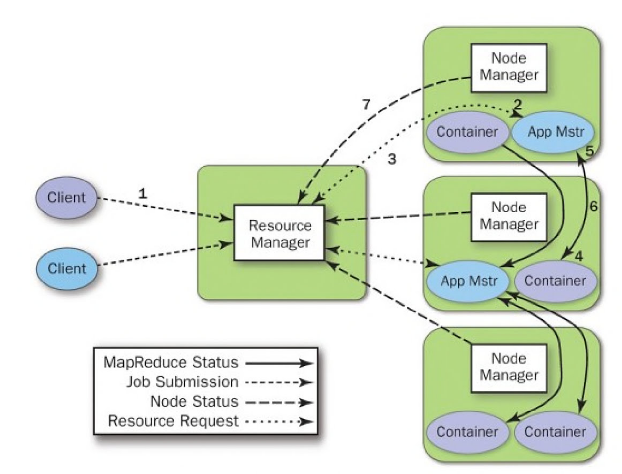
Under V2 of Hadoop we have several new components:

Resource Manager - schedules and manages all jobs on the cluster

Node Manager -

Container - computing resource which usually includes processing core and some memory

Application Master - runs within a container



**Cassandra**

Cassandra is a *decentralized* *distributed* database with no single point of failure. This means Cassandra can run on multiple machines(nodes), and each node knows and does everything in the same manner as the rest. There are no Master - Slave relationships because all the nodes are *identical*. As such, if one node fail, the other can successfully serve requests without a problem. This is called a *symmetrical system.*

Cassandra is also *elastically scalable* - we can add or remove Cassandra nodes without having to do any work. Cassandra can automatically find and send work to a new machine or scale back when nodes are removed.

Cassandra’s main features are high availability and scalability.

**Uses of Cassandra**

1. Lots of Writes, Statistics, and Analysis

Cassandra is good for applications that involve frequent writes to the database such as product reviews, and social networks.