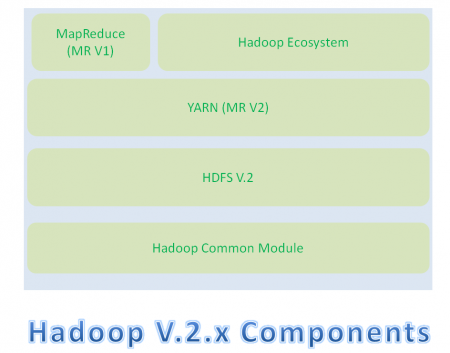
**Hadoop 2.x Architecture**



* Hadoop Common Module is a Hadoop Base API (A Jar file) for all Hadoop Components. All other components works on top of this module.
* HDFS stands for Hadoop Distributed File System. It is also know as HDFS V2 as it is part of Hadoop 2.x with some enhanced features. It is used as a Distributed Storage System in Hadoop Architecture.
* YARN stands for Yet Another Resource Negotiator. It is new Component in Hadoop 2.x Architecture. It is also know as “MR V2”.
* MapReduce is a Batch Processing or Distributed Data Processing Module. It is also know as “MR V1” as it is part of Hadoop 1.x with some updated features.
* Remaining all Hadoop Ecosystem components work on top of these three major components: HDFS, YARN and MapReduce. We will discuss all Hadoop Ecosystem components in-detail in my coming posts.

When compared to Hadoop 1.x, Hadoop 2.x Architecture is designed completely different. It has added one new component : YARN and also updated HDFS and MapReduce component’s Responsibilities.

### Hadoop 2.x Major Components

Hadoop 2.x has the following three Major Components:

* HDFS
* YARN
* MapReduce

These three are also known as Three Pillars of Hadoop 2. Here major key component change is YARN. It is really game changing component in BigData Hadoop System

Hadoop 2.x components follow this architecture to interact each other and to work parallel in a reliable, highly available and fault-tolerant manner

HDFS :

HDFS is the primary distributed storage used by Hadoop applications. A HDFS cluster primarily consists of a NameNode that manages the file system metadata and DataNodes that store the actual data. The HDFS Architecture Guide describes HDFS in detail. This user guide primarily deals with the interaction of users and administrators with HDFS clusters. The HDFS architecture diagram depicts basic interactions among NameNode, the DataNodes, and the clients. Clients contact NameNode for file metadata or file modifications and perform actual file I/O directly with the DataNodes.

The following are some of the salient features that could be of interest to many users.

* Hadoop, including HDFS, is well suited for distributed storage and distributed processing using commodity hardware. It is fault tolerant, scalable, and extremely simple to expand. MapReduce, well known for its simplicity and applicability for large set of distributed applications, is an integral part of Hadoop.
* HDFS is highly configurable with a default configuration well suited for many installations. Most of the time, configuration needs to be tuned only for very large clusters.
* Hadoop is written in Java and is supported on all major platforms.
* Hadoop supports shell-like commands to interact with HDFS directly.
* The NameNode and Datanodes have built in web servers that makes it easy to check current status of the cluster.
* New features and improvements are regularly implemented in HDFS. The following is a subset of useful features in HDFS:
  + File permissions and authentication.
  + Rack awareness: to take a node’s physical location into account while scheduling tasks and allocating storage.
  + Safemode: an administrative mode for maintenance.
  + fsck: a utility to diagnose health of the file system, to find missing files or blocks.
  + fetchdt: a utility to fetch DelegationToken and store it in a file on the local system.
  + Balancer: tool to balance the cluster when the data is unevenly distributed among DataNodes.
  + Upgrade and rollback: after a software upgrade, it is possible to rollback to HDFS’ state before the upgrade in case of unexpected problems.
  + Secondary NameNode: performs periodic checkpoints of the namespace and helps keep the size of file containing log of HDFS modifications within certain limits at the NameNode.
  + Checkpoint node: performs periodic checkpoints of the namespace and helps minimize the size of the log stored at the NameNode containing changes to the HDFS. Replaces the role previously filled by the Secondary NameNode, though is not yet battle hardened. The NameNode allows multiple Checkpoint nodes simultaneously, as long as there are no Backup nodes registered with the system.
  + Backup node: An extension to the Checkpoint node. In addition to checkpointing it also receives a stream of edits from the NameNode and maintains its own in-memory copy of the namespace, which is always in sync with the active NameNode namespace state. Only one Backup node may be registered with the NameNode at once.

## Prerequisites

The following documents describe how to install and set up a Hadoop cluster:

* [Single Node Setup](https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html) for first-time users.
* [Cluster Setup](https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/ClusterSetup.html) for large, distributed clusters.

The rest of this document assumes the user is able to set up and run a HDFS with at least one DataNode. For the purpose of this document, both the NameNode and DataNode could be running on the same physical machine.

MapReduce :

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see [HDFS Architecture Guide](https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html)) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

The MapReduce framework consists of a single master ResourceManager, one slave NodeManager per cluster-node, and MRAppMaster per application (see [YARN Architecture Guide](https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.html)).

Minimally, applications specify the input/output locations and supply *map* and *reduce* functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the *job configuration*.

The Hadoop *job client* then submits the job (jar/executable etc.) and configuration to the ResourceManager which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

Although the Hadoop framework is implemented in Java™, MapReduce applications need not be written in Java.

* [Hadoop Streaming](https://hadoop.apache.org/docs/r2.7.2/api/org/apache/hadoop/streaming/package-summary.html) is a utility which allows users to create and run jobs with any executables (e.g. shell utilities) as the mapper and/or the reducer.
* [Hadoop Pipes](https://hadoop.apache.org/docs/r2.7.2/api/org/apache/hadoop/mapred/pipes/package-summary.html) is a [SWIG](http://www.swig.org/)-compatible C++ API to implement MapReduce applications (non JNI™ based).

## Inputs and Outputs

The MapReduce framework operates exclusively on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and value classes have to be serializable by the framework and hence need to implement the [Writable](https://hadoop.apache.org/docs/r2.7.2/api/org/apache/hadoop/io/Writable.html) interface. Additionally, the key classes have to implement the [WritableComparable](https://hadoop.apache.org/docs/r2.7.2/api/org/apache/hadoop/io/WritableComparable.html) interface to facilitate sorting by the framework.

Input and Output types of a MapReduce job:

(input) <k1, v1> -> **map** -> <k2, v2> -> **combine** -> <k2, v2> -> **reduce** -> <k3, v3> (output)

YARN :

The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons. The idea is to have a global ResourceManager (*RM*) and per-application ApplicationMaster (*AM*). An application is either a single job or a DAG of jobs.

The ResourceManager and the NodeManager form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

The ResourceManager has two main components: Scheduler and ApplicationsManager.

The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based the resource requirements of the applications; it does so based on the abstract notion of a resource *Container* which incorporates elements such as memory, cpu, disk, network etc.

The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the various queues, applications etc. The current schedulers such as the [CapacityScheduler](http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html) and the [FairScheduler](http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html) would be some examples of plug-ins.

The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure. The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

MapReduce in hadoop-2.x maintains **API compatibility** with previous stable release (hadoop-1.x). This means that all MapReduce jobs should still run unchanged on top of YARN with just a recompile.

