**Assignment 3.1**

**Components of HDFS in detail**

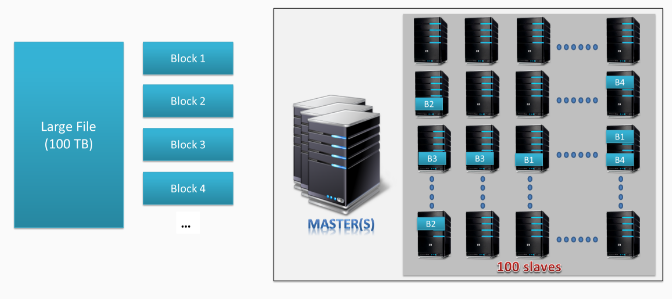
HDFS is the world’s most reliable storage system. HDFS is a Filesystem of Hadoop designed for storing very large files running on a cluster of commodity hardware. HDFS is designed on principle of storage of less number of large files rather than the huge number of small files. It provides fault tolerant storage layer for Hadoop and its other components. Replication of data helps us to attain this feature. It stores data reliably even in the case of hardware failure. It provides high throughput access to application data by providing the data access in parallel.

**HDFS Nodes**:

1. Master Node: Master node (Also called Name node) – As the name suggests, this node manages all the slave nodes and assign work to slaves. It should be deployed on reliable hardware as it is the centerpiece of HDFS.
2. Slave Node: Slave node (Also called data node) – Datanodes are the slaves which are deployed on each machine and provide the actual storage. They are the actual worker nodes. These are responsible for serving read and write requests from the clients. They can be deployed on commodity hardware. If any slave node goes down, namenode automatically replicates the blocks which were present at that data node to other nodes in the cluster.

**HDFS Daemons**:

1. Namenode: This is the daemon that runs on all the masters. Name node stores metadata like filename, the number of blocks, number of replicas, a location of blocks, block IDs etc. This metadata is available in memory in the master for faster retrieval of data. In the local disk, a copy of metadata is available for persistence. So name node memory should be high as per the requirement.
2. Datanode: This is the daemon that runs on the slave. These are actual worker nodes that store the data.

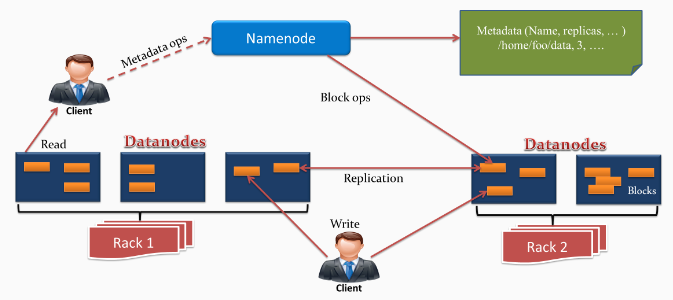


**Data Storage In HDFS:**

Whenever any file has to be written in HDFS, it is broken into small pieces of data known as blocks. HDFS has a default block size of 128 MB which can be increased as per the requirements. These blocks are stored in the cluster in distributed manner on different nodes. This provides a mechanism for [MapReduce](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/) to process the data in parallel in the cluster. Multiple copies of each block are stored across the cluster on different nodes. This is a replication of data. By default, HDFS has a replication factor of 3. It provides fault tolerance, reliability, and high availability. A Large file is split into n number of small blocks. These blocks are stored at different nodes in the cluster in a distributed manner. Each block is replicated and stored across different nodes in the cluster.

**Rack Awareness In HDFS**:

[Hadoop](http://data-flair.training/blogs/hadoop-ecosystem-components/)runs on a cluster of computers which are commonly spread across many racks. NameNode places replicas of a block on multiple racks for improved fault tolerance. NameNode tries to place at least one replica of a block in each rack, so that if a complete rack goes down then also system will be highly available Optimizing replica placement distinguishes HDFS from most other distributed file systems. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization.



**Features Of HDFS**:

1. Distributed Storage – Data is stored in distributed manner
2. Blocks – Data is split into blocks
3. Replication – Blocks are replicated at different nodes
4. High Availability – Data is highly available due to replication
5. Data Reliability – Data is stored reliably in HDFS
6. Fault tolerant – Data replication provides fault tolerance feature
7. Scalability – Nodes in HDFS cluster can be increased on the fly
8. High throughput access to application – Parallel processing provides high throughput access to application

**MapReduce**:

Hadoop MapReduce is the core component of Hadoop which provides data processing. MapReduce is a software framework for easily writing applications that process the vast amount of structured and unstructured data stored in the Hadoop Distributed File system.

MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster. Thus, it improves the speed and reliability of cluster this parallel processing.

Each phase has [key-value pairs](http://data-flair.training/blogs/key-value-pairs-hadoop-mapreduce/) as input and output. In addition, programmer also specifies two functions: map function and reduce function

Map function takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

**Reduce** **function**takes the output from the Map as an input and combines those data tuples based on the key and accordingly modifies the value of the key.

**YARN**:

Hadoop YARN (Yet Another Resource Negotiator) provides the resource management. YARN is called as the operating system of Hadoop as it is responsible for managing and monitoring workloads. It allows multiple data processing engines such as real-time streaming and batch processing to handle data stored on a single platform.

YARN has been projected as a data operating system for [Hadoop2](http://data-flair.training/blogs/setup-hadoop-2-yarn-psedo-distributed-mode/). Main features of YARN are:

**Flexibility** – Enables other purpose-built data processing models beyond MapReduce (batch), such as interactive and streaming. Due to this feature of YARN, other applications can also be run along with Map Reduce programs in Hadoop2.

**Efficiency** – As many applications run on the same cluster, Hence, efficiency of Hadoop increases without much effect on quality of service.

**Shared** – Provides a stable, reliable, secure foundation and shared operational services across multiple workloads. Additional programming models such as graph processing and iterative modelling are now possible for data processing.

**Resource Manager:**

The ResourceManager is mainly concerned with arbitrating available resources in the cluster among competing applications, with the goal of maximum cluster utilization. The ResourceManager includes a pluggable scheduler called the YarnScheduler, which allows different policies for managing constraints such as capacity, fairness, and service level agreements.

The Resource manager manages resources as follows:

Each NodeManager takes instructions from the ResourceManager, reporting and handling containers on a single node. Each ApplicationMaster requests resources from the ResourceManager, then works with containers provided by NodeManagers

The ResourceManager communicates with application clients via an interface called the ClientService, through which a client can submit or terminate an application and gain information about the scheduling queue or cluster statistics. Administrative requests are served by a separate interface called the AdminService, through which operators can get updated information about cluster operation. Behind the scenes, the ResourceTrackerService receives node heartbeats from the NodeManager to track new or decommissioned nodes. The NMLivelinessMonitor and NodesListManager keep an updated status of which nodes are healthy so that the scheduler and the ResourceTrackerService can allocate work appropriately.

A component called the ApplicationMasterService manages ApplicationMasters on all nodes, keeping the scheduler informed. A component called the AMLivelinessMonitor keeps a list of ApplicationMasters and their last heartbeat times, in order to let the ResourceManager know what applications are healthy on the cluster. Any ApplicationMaster that does not heartbeat within a certain interval is marked as dead and re-scheduled to run on a new container.

At the core of the ResourceManager is an interface called the ApplicationsManager, which maintains a list of applications that have been submitted, are running, or are completed. The ApplicationsManager accepts job submissions, negotiates the first container for an application (in which the ApplicationMaster will run) and restarts the ApplicationMaster if it fails. The ResourceManager and NodeManagers communicate via heartbeats.

**ApplicationMaster:**

The ApplicationMaster is an instance of a framework-specific library that negotiates resources from the ResourceManager and works with the NodeManager to execute and monitor the granted resources (bundled as containers) for a given application. An application can be a process or set of processes, a service, or a description of work.

The ApplicationMaster is run in a container like any other application. The ApplicationsManager, part of the ResourceManager, negotiates for the container in which an application’s ApplicationMaster runs when the application is scheduled by the YarnScheduler.

While an application is running, the ApplicationMaster manages the following:

Application lifecycle

Dynamic adjustments to resource consumption

Execution flow

Faults

Providing status and metrics

The ApplicationMaster is architected to support a specific framework, and can be written in any language since its communication with the NodeManagers and the ResourceManager is accomplished using extensible communication protocols. The ApplicationMaster can be customized to extend the framework or run any other code. For this reason, the ApplicationMaster is not considered trustworthy, and is not run as a trusted service.

An ApplicationMaster typically requests resources on multiple nodes to complete a job by sending the ResourceManager requests that include locality preferences and attributes of the containers. When the ResourceManager is able to allocate a resource to the ApplicationMaster, it generates a lease that the ApplicationMaster pulls on a subsequent heartbeat. A security token associated with the lease guarantees its authenticity when the ApplicationManager presents the lease to the NodeManager to gain access to the container.

The Application Master heartbeats to the ResourceManager to communicate its changing resource needs, and to let the ResourceManager know it is still alive. In response, the ResourceManager can return a lease on additional containers on other nodes, or cancel the lease on some containers. The ApplicationMaster can then adjust its execution strategy to fit the increase or decrease in available resources. When cluster resources become scarce, the ResourceManager can also request that the ApplicationMaster relinquish some resources. The ApplicationMaster can move work to other running containers in order to give up resources gracefully.

**Containers:**

A YARN container is a result of a successful resource allocation, meaning that the ResourceManager has granted an application a lease to use a specific set of resources in certain amounts on a specific node. The ApplicationMaster presents the lease to the NodeManager on the node where the container has been allocated, thereby gaining access to the resources.

To launch the container, the ApplicationMaster must provide a container launch context (CLC) that includes the following information:

Environment variables

Dependencies (local resources such as data files or shared objects needed prior to launch)

Security tokens

The command necessary to create the process the application plans to launch

The CLC makes it possible for the ApplicationMaster to use containers to run a variety of different kinds of work, from simple shell scripts to applications to virtual machines.

**HistoryServer:**

A YARN component called the HistoryServer archives job metrics and metadata. Status on completed applications is available via REST APIs.

**MapReduce 1 Compared with MapReduce 2:**

In contrast to Hadoop 1.x, Hadoop 2.x dynamically allocates resources for applications as they execute. The MapReduce framework has been rewritten to run as an application on top of YARN; this new version is called MapReduce 2.0.

The main advancement in YARN architecture is the separation of resource management and job management, which were both handled by the same process (the JobTracker) in Hadoop 1.x. Cluster resources and job scheduling are managed by the ResourceManager, and resource negotiation and job monitoring are managed by an ApplicationMaster for each application running on the cluster. In MapReduce, each node advertises a relatively fixed number of map slots and reduce slots. This can lead to resource under-utilization, for example, when there is a heavy reduce load and map slots are available, because the map slots cannot accept reduce tasks (and vice versa).

YARN generalizes resource management for use by new engines and frameworks, allowing resources to be allocated and reallocated for different concurrent applications sharing a cluster. Existing MapReduce applications can run on YARN without any changes. At the same time, because MapReduce is now merely another application on YARN, MapReduce is free to evolve independently of the resource management infrastructure.