**Session 3 : Exploring Map Reduce**

**Assignment 3.1**

**List the Components of Hadoop 2.x and explain each component in detail.**

Hadoop comprises of four core components.

1. Based on storage component- HDFS
2. Based on Data processing component – Map reduce
3. Yarn
4. Hadoop Common
5. **HDFS (Hadoop Distributed File System)**

HDFS is a distributed file system which stores the data in distributed manner. Rather than storing a complete file it divides a file into small blocks (of 64 or 128 MB size) and distributes them across the cluster. Each block is replicated (3 times as per default configuration) multiple times and is stored on different nodes to ensure data availability. Normally HDFS can be installed on native file systems like xfs, ext3 or ext4 (Similar to Unix/Linux file systems). HDFS comprises of 3 important components-NameNode, DataNode and Secondary NameNode.

* **Namenode**: The HDFS namespace consists of files and directories. These files and directories are represented by inodes on the NameNode. These Inodes have the task to keep a track of attributes e.g. permissions, modification and access times, the allotted quota for namespace and disk space. Content of the file is broken into large blocks usually a size of 128 megabytes, but user can also set the block size depending upon the situation. And each block of the file is independently replicated at multiple DataNodes. Normally the data is replicated on three datanode instances but user can set this count as per need.
* **Datanode**: The DataNode replica block consists of two files on the local filesystem. The first file is for the data while the second file is for recording the block's metadata. The metadata here includes the checksums for the data and the generation stamp. The data file size should be the same of the actual length of the block. It and does not require any extra space to round it up to the nominal block size as in the traditional file systems. Hence if any of the blocks is half full it requires only half of the space of the full block on the local drive.

During the startup each DataNode connects to its corresponding NameNode and does the handshaking. This handshaking verifies the namespace ID and the software version of the DataNode. If there is any mismatch found, the DataNode goes down automatically.

The namespace ID is assigned to the file system instance as soon as it is formatted. This namespace ID is stored on all nodes of the cluster. The nodes which have a different namespace ID will not be allowed to join the cluster. This protects the integrity of the file system. A DataNode which is newly initialized and does not have any namespace ID is allowed to join the cluster and get the cluster's namespace ID.

Once the handshaking is done, the DataNode gets registered with the NameNode. The DataNodes store their unique storage IDs. These storage IDs are internal identifiers of the DataNodes. This makes it uniquely identifiable even if it is restarted on a different IP address or port. The storage ID gets assigned to the DataNode when it is registered with the NameNode for the first time and it never changes after that.

A DataNode identifies the block replicas under its possession to the NameNode by sending a block report. A block report is a combination of the block ID, the generation stamp and the length for each block replica the server hosts. Report from the first block is sent immediately after the DataNode registration. The subsequent block reports are then sent every hour and provide the NameNode with an up-to-date view of where block replicas are located on the cluster.

While doing the usual operation, the DataNodes sends signals to the corresponding NameNode in order to confirm that the DataNode is operating and the block replicas which it had hosted, are live. On default, these signal heartbeat interval is three seconds. If the NameNode does not receive any signal from a DataNode for ten minutes, the NameNode considers that the DataNode is out of service and the block replicas which are hosted by that DataNode becomes unavailable. The NameNode then schedules the formation of new replicas of those blocks on other DataNodes.

Signals from the DataNode also carry the information about the total storage capacity, fraction of the storage in use, and the number of data transfers currently in progress. These statistics are used for the NameNode's block allocation and load balancing decisions.

* **Secondary NameNode**: this node 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**: this 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**: this node is an extension to the Checkpoint node. In addition to check-pointing, 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.

1. **Map Reduce**

MapReduce is the algorithm of executing any task on distributed system. Using MapReduce one can process a large file in parallel manner. MapReduce framework executes any task on different nodes as full file is distributed across the cluster in a form of various blocks.

It has two phases, Map (Mapper Task) and Reduce (Reducer Task)

* Each of these tasks would run on individual blocks of the data
* First mapper task would take each line of elements as an input and generates intermediate key value pairs
* Each mapper task is executed on a single block of data
* Than reducer task will take list of key value pairs for same keys, process the data and generates the final output
* A phase called shuffle and sort will take place between mapper and reducer task will send the data to proper reducer tasks
* Shuffle process maps the mapper output with the same key to the collection of values as a value

For example (key1, val1) and (key1, val2) will be converted to (key1, [val1, val2])

* The mapper and reducer tasks would in parallel
* The reducer tasks can start their word as soon as mapper tasks are completed

It cannot be installed on Windows directly. To run HDFS on your Windows machine You need to use tools like Cygwin.

You can write file and read file from HDFS. You cannot updated any file on HDFS. Recently Hadoop has added the support of appending content to the file which was not there in previous releases.

1. **Hadoop Common**

Apache Foundation has pre-defined set of utilities and libraries that can be used by other modules within the Hadoop ecosystem. For example, if HBase and Hive want to access HDFS they need to make of Java archives (JAR files) that are stored in Hadoop Common.

## **YARN**

[YARN](https://www.dezyre.com/article/hadoop-2-0-yarn-framework-the-gateway-to-easier-programming-for-hadoop-users/84) forms an integral part of Hadoop 2.0. YARN is great enabler for dynamic resource utilization on Hadoop framework as users can run various Hadoop applications without having to bother about increasing workloads.

YARN has total three major components

* ResourceManager
* NodeManager
* ApplicationMaster

## 1) ResourceManager

* This daemon process resides on the Master Node (not necessarily on NameNode of Hadoop)
* Responsible for,
  + Managing resources scheduling for different compute applications in an optimum way
  + Coordinating with two process on master node, **Scheduler** and **ApplicationManager**

#### Scheduler

* This daemon process resides on the Master Node (runs along with ResourceManager daemon )
* Responsible for,
  + Scheduling the job execution as per submission request received by ResourceManager
  + Allocating resources to applications submitted to the cluster
  + Coordinating with ApplicationManager daemon and keeping track of resources of running applications

#### ApplicationManager

* This daemon process resides on the Master Node (runs along with ResourceManager daemon )
* Responsible for,
  + Helping Scheduler daemon to keeps track of running application by coordination
  + Accepting job submissions from client
  + Negotiating first container for executing application specific task with suitable ApplicationMaster on slave node

## 2) NodeManager

* This daemon process resides on the slave nodes (runs along with DataNode daemon)
* Responsible for,
  + Managing and executing containers
  + Monitoring resource usage (i.e. usage of memory, cpu, network etc..) and reporting it back to ResourceManager daemon
  + Periodically sending heart-bits to ResourceManager for its health status update

## 3) ApplicationMaster

* This daemon process runs on the slave node (along with the NodeManager daemon)
* It is per application specific library works with NodeManager to execute the task
* The instance of this daemon is per application, which means in case of multiple jobs submitted on cluster, it may have more than one instances of ApplicationMaster on slave nodes
* Responsible for,
  + Negotiating suitable resource containers on slave node from ResourceManager
  + Working with one or multiple NodeManagers to monitor task execution on slave nodes