SESSION 3

ASSIGNMENT 1

Different components of Hadoop 2.x are as follows:

1. **HDFS** (**H**adoop **D**istributed **F**ile **S**ystem)
   HDFS is a Java-based file system that provides scalable and reliable data storage. HDFS is a scalable, fault-tolerant, distributed storage system that works closely with a wide variety of concurrent data access applications, coordinated by YARN.

Feature	Description
Rack awareness	Considers a node's physical location when allocating storage and scheduling tasks
Minimal data motion	Hadoop moves compute processes to the data on HDFS and not the other way around. Processing tasks can occur on the physical node where the data resides, which significantly reduces network I/O and provides very high aggregate bandwidth.
Utilities	Dynamically diagnose the health of the file system and rebalance the data on different nodes
Rollback	Allows operators to bring back the previous version of HDFS after an upgrade, in case of human or systemic errors
Standby NameNode	Provides redundancy and supports high availability (HA)
Operability	HDFS requires minimal operator intervention, allowing a single operator to maintain a cluster of 1000s of nodes

2. **YARN** (**Y**et **A**nother **R**esource **N**egotiator)

YARN combines a central resource manager that reconciles the way applications use Hadoop system resources with node manager agents that monitor the processing operations of individual cluster nodes.

YARN is a software rewrite that decouples MapReduce's resource management and scheduling capabilities from the data processing component, enabling Hadoop to support more varied processing approaches and a broader array of applications.

Feature	Description
Multi-tenancy	YARN allows multiple access engines (either open-source or proprietary) to use Hadoop as the common standard for batch, interactive and real-time engines that can simultaneously access the same data set. Multi-tenant data processing improves an enterprise's return on its Hadoop investments.
Cluster utilization	YARN's dynamic allocation of cluster resources improves utilization over more static MapReduce rules used in early versions of Hadoop
Scalability	Data center processing power continues to rapidly expand. YARN's ResourceManager focuses exclusively on scheduling and keeps pace as clusters expand to thousands of nodes managing petabytes of data.
Compatibility	Existing MapReduce applications developed for Hadoop 1 can run YARN without any disruption to existing processes that already work

3. MapReduce

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

A MapReduce program is composed of a Map() procedure (method) that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() method that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

MapReduce is useful for batch processing on terabytes or petabytes of data stored in Apache Hadoop.

A MapReduce job splits a large data set into independent chunks and organizes them into key, value pairs for parallel processing. This parallel processing improves the speed and reliability of the cluster, returning solutions more quickly and with greater reliability.

The **Map** function divides the input into ranges by the InputFormat and creates a map task for each range in the input. The JobTracker distributes those tasks to the worker nodes. The output of each map task is partitioned into a group of key-value pairs for each reduce.

The **Reduce** function then collects the various results and combines them to answer the larger problem that the master node needs to solve. Each reduce pulls the relevant partition from the machines where the maps executed, then writes its output back into HDFS. Thus, the reduce is able to collect the data from all of the maps for the keys and combine them to solve the problem.