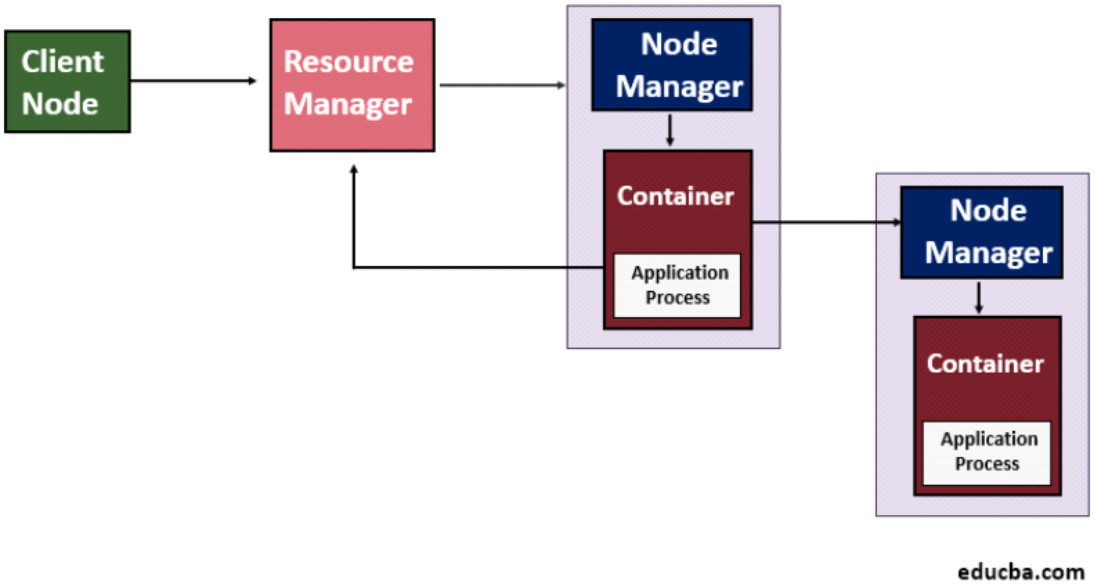
Big Data Assignment 1

1. Briefly describe the YARN architecture.

Hadoop YARN Architecture is the reference architecture for resource management for Hadoop framework components. YARN, which is known as Yet Another Resource Negotiator, is the Cluster management component of Hadoop 2.0. It includes Resource Manager, Node Manager, Containers, and Application Master. The Resource Manager is the major component that manages application management and job scheduling for the batch process. Node manager is the component that manages task distribution for each data node in the cluster. Containers are the hardware components such as CPU, RAM for the Node that is managed through YARN. Application Master is for monitoring and managing the application lifecycle in the Hadoop cluster.



[Architecture of Hadoop YARN]

YARN introduces the concept of a Resource Manager and an Application Master in Hadoop 2.0. The Resource Manager sees the usage of the resources across the Hadoop cluster whereas the life cycle of the applications that are running on a particular cluster is supervised by the Application Master. Basically, we can say that for cluster resources, the Application Master negotiates with the Resource Manager. This task is carried out by the containers which hold definite memory restrictions. Then these containers are used to run the application-specific processes and also these containers are supervised by the Node Managers which are running on nodes in the cluster. This will confirm that

no more than the allocated resources are used by the application.

Various Components of YARN:

1. Resource Manager

YARN works through a Resource Manager which is one per node and Node Manager which runs on all the nodes. The Resource Manager manages the resources used across the cluster and the Node Manager lunches and monitors the containers. Scheduler and Application Manager are two components of the Resource Manager.

i. Scheduler: Scheduling is performed based on the requirement of resources by the applications. YARN provides few schedulers to choose from and they are Fair and Capacity Scheduler. In case of any hardware or application failure, the Scheduler does not ensure to restart the failed tasks. Also, Scheduler allocates resources to the running applications based on the capacity and queue.

ii. Application Manager: It manages the running of Application Master in a cluster and on the failure of the Application Master Container, it helps in restarting it. Also, it bears the responsibility of accepting the submission of the jobs.

2. Node Manager

Node Manager is responsible for the execution of the task in each data node. The Node Manager in YARN by default sends a heartbeat to the Resource Manager which carries the information of the running containers and regarding the availability of resources for the new containers. It is responsible for seeing to the nodes on the cluster individually and manages the workflow and user jobs on a specific node. Chiefly it manages the application containers which are assigned by the Resource Manager. The Node Manager starts the containers by creating the container processes which are requested and it also kills the containers as asked by the Resource Manager.

3. Containers

The Containers are set of resources like RAM, CPU, and Memory etc on a single node and they are scheduled by Resource Manager and monitored by Node Manager. The Container Life Cycle manages the YARN containers by using container launch context and provides access to the application for the specific usage of resources in a particular host.

4. Application Master

It monitors the execution of tasks and also manages the lifecycle of applications running on the cluster. An individual Application Master gets associated with a job when it is submitted to the framework. Its chief responsibility is to negotiate the resources from the Resource Manager. It works with the Node Manager to monitor and execute the tasks.

2. In Hadoop, what is map and reduce? Create real-world examples that explain. how to use maps and reduce?

MapReduce is a software framework and programming model used for processing huge amounts of data. MapReduce program work in two phases, namely, Map and Reduce. Map tasks deal with splitting and mapping of data while Reduce tasks shuffle and reduce the data.

Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. The programs of Map Reduce in cloud computing are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

The input to each phase is key-value pairs. In addition, every programmer needs to specify two functions: map function and reduce function.

MapReduce Architecture in Big Data explained in detail

The whole process goes through four phases of execution namely, splitting, mapping, shuffling, and reducing.

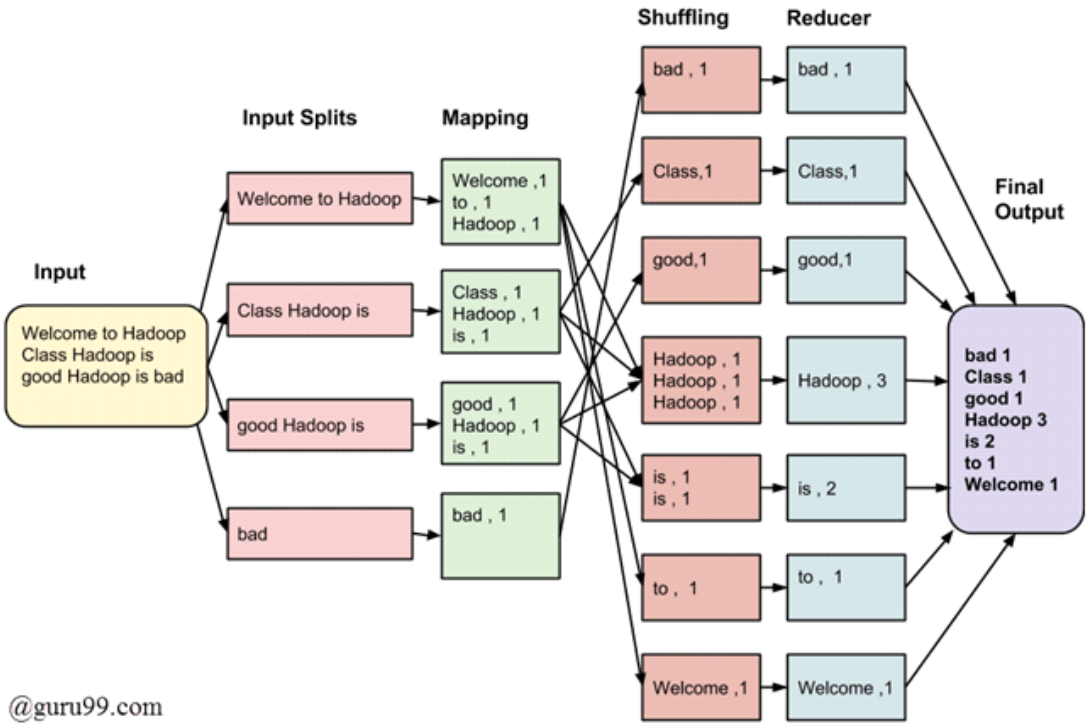
let's understand with a MapReduce example–

Consider you have following input data for your MapReduce in Big data Program

Welcome to Hadoop Class

Hadoop is good

Hadoop is bad



MapReduce Architecture

The final output of the MapReduce task is

bad	1
Class	1
good	1
Hadoop	3
is	2
to	1
Welcome	1

The data goes through the following phases of MapReduce in Big Data

i) **Input Splits:**

An input to a MapReduce in Big Data job is divided into fixed-size pieces called input splits Input split is a chunk of the input that is consumed by a single map

ii) **Mapping:**

This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

iii) **Shuffling:**

This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubed together along with their respective frequency.

iv) **Reducing:**

In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.

How MapReduce Organizes Work?

Hadoop divides the job into tasks. There are two types of tasks:

Map tasks (Splits & Mapping)

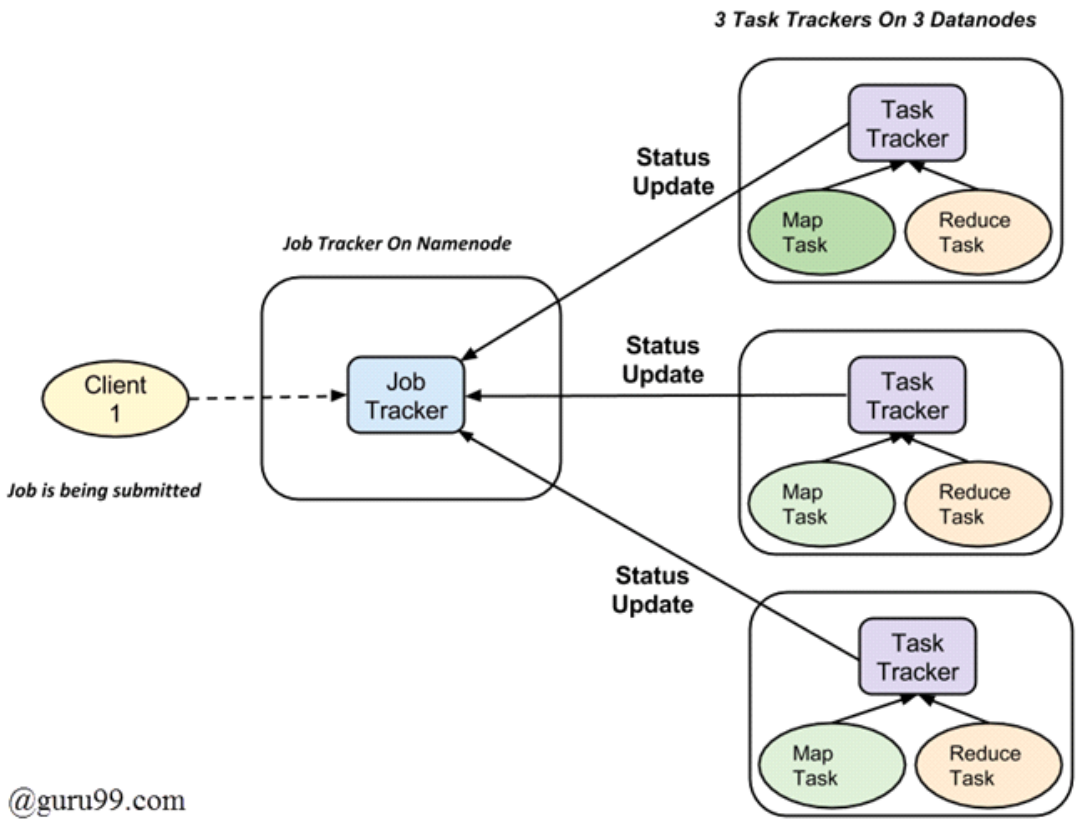
Reduce tasks (Shuffling, Reducing)

The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a

Jobtracker: Acts like a master (responsible for complete execution of submitted job)

Multiple Task Trackers: Acts like slaves, each of them performing the job

For every job submitted for execution in the system, there is one Jobtracker that resides on Namenode and there are multiple tasktrackers which reside on Datanode.



How Hadoop MapReduce Works

A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.

It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes.

Execution of individual task is then to look after by task tracker, which resides on every data node executing part of the job.

Task tracker's responsibility is to send the progress report to the job tracker.

In addition, task tracker periodically sends 'heartbeat' signal to the Jobtracker so as to notify him of the current state of the system.

Thus job tracker keeps track of the overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.

3. What are the differences between HBASE and NoSQL? What are the

most important features of HBase?

NoSQL:

NoSQL provides the new data management technologies designed to meet the increasing volume, velocity, and variety of data. It can store and retrieve data that is modeled in means other than the tabular relations used in relational databases. NoSQL systems are also called “Not only SQL” to emphasize that they may also support SQL-like query languages.

The Relational Databases have the following challenges:

Not good for large volume (Petabytes) of data with variety of data types (eg. images, videos, text)

Cannot scale for large data volume

Cannot scale-up, limited by memory and CPU capabilities

Cannot scale-out, limited by cache dependent Read and Write operations

Sharding (break database into pieces and store in different nodes) causes operational problems (e.g. managing a shared failure)

Complex RDBMS model

Consistency limits the scalability in RDBMS

Compared to relational databases, NoSQL databases are more scalable and provide superior performance. NoSQL databases address the challenges that the relational model does not by providing the following solution:

A scale-out, shared-nothing architecture, capable of running on a large number of nodes

A non-locking concurrency control mechanism so that real-time reads will not conflict writes

Scalable replication and distribution – thousands of machines with distributed data

An architecture providing higher performance per node than RDBMS

Schema-less data model

HBase:

Wide-column store based on Apache Hadoop and on concepts of BigTable.

Apache HBase is a NoSQL key/value store which runs on top of HDFS. Unlike Hive, HBase operations run in real-time on its database rather than MapReduce jobs. HBase is partitioned to tables, and tables are further split into column families. Column families, which must be declared in the schema, group together a certain set of columns (columns don't require schema definition). For example, the "message" column family may include the columns: "to", "from", "date", "subject", and "body". Each key/value pair in HBase is defined as a cell, and each key consists of row-key, column family, column, and time-stamp. A row in HBase is a grouping of key/value mappings identified by the row-key. HBase enjoys Hadoop's infrastructure and scales horizontally using off the shelf servers.

HBase works by storing data as key/value. It supports four primary operations: put to add or update rows, scan to retrieve a range of cells, get to return cells for a specified row, and delete to remove rows, columns or column versions from the table. Versioning is available so that previous values of the data can be fetched (the history can be deleted every now and then to clear space via HBase compactions). Although HBase includes tables, a schema is only required for tables and column families, but not for columns, and it includes increment/counter functionality.

HBase queries are written in a custom language that needs to be learned. SQL-like functionality can be achieved via Apache Phoenix, though it comes at the price of maintaining a schema. Furthermore, HBase isn't fully ACID compliant, although it does support certain properties. Last but not least - in order to run HBase, ZooKeeper is required - a server for distributed coordination such as configuration, maintenance, and naming.

HBase is perfect for real-time querying of Big Data. Facebook use it for messaging and real-time analytics. They may even be using it to count Facebook likes.

Hbase has centralized architecture where The Master server is responsible for monitoring all RegionServer(responsible for serving and managing regions) instances in the cluster, and is the interface for all metadata changes. It provides CP(Consistency, Availability) form CAP theorem.

HBase is optimized for reads, supported by single-write master, and resulting strict consistency model, as well as use of Ordered Partitioning which supports row-scans. HBase is well suited for doing Range based scans.

Linear Scalability for large tables and range scans -

Due to Ordered Partitioning, HBase will easily scale horizontally while still supporting rowkey range scans.

i) Secondary Indexes -

Hbase does not natively support secondary indexes, but one use-case of Triggers is that a trigger on a ""put"" can automatically keep a secondary index up-to-date, and therefore not put the burden on the application (client)."

ii) Simple Aggregation-

Hbase Co Processors support out-of-the-box simple aggregations in HBase. SUM, MIN, MAX, AVG, STD. Other aggregations can be built by defining java-classes to perform the aggregation

The features of Hbase NoSQL DB

Apache HBase is a column oriented database which supports dynamic database schema. It mainly runs on top of the HDFS and supports MapReduce jobs. HBase also supports other high level languages for data processing.

Let us have a look at the different features of HBase:.

- **Scalability:** HBase supports scalability in both linear and modular form
- **Sharding:** HBase supports automatic sharding of tables. It is also configurable.
- **Distributed storage:** HBase supports distributed storage like HDFS
- **Consistency:** It supports consistent read and write operations
- **Failover support:** HBase supports automatic failover
- **API support:** HBase supports Java APIs so clients can access it easily
- **MapReduce support:** HBase supports MapReduce for parallel processing of large

volume of data

- **Back up support:** HBase supports back up of Hadoop MapReduce jobs in HBase tables
- **Real time processing:** It supports block cache and Bloom filters. So, real time query processing is easy

Apart from the above major features, HBase also supports REST-ful web services, jruby-based shell, Ganglia and JMX. So, HBase has a very strong presence in the NoSQL database world.

4. In HBase, what data manipulation commands are available?

HBase provide many data manipulation shell commands that you can use on the interactive shell to manipulate the data. The Commonly used HBase data manipulation shell commands are: Count, Put, Get, Delete, Delete all, Truncate, Scan.

5. In HBase, explain tombstone markers.

There are 3 different types of tombstone markers in HBase for deletion-

- **Family Delete Marker** - This marker marks all columns for a column family.
- **Version Delete Marker** - This marker marks a single version of a column.
- **Column Delete Marker** - This markers mark all the versions of a column.