UNIT II
HADOOP FRAMEWORK

Hadoop

- Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models.
- A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers.
- Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.
- In short Hadoop is an open source software framework for sorting and processing big data in distributed way on large clusters of commodity hardware.
- Basically it accomplishes the following tasks
  1. Massive Storage
  2. Faster Processing

2.1 Distributed File Systems

- Hadoop can work directly with any **mountable distributed file system** such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).
- The Hadoop Distributed File System (HDFS) is based on the **Google File System (GFS)** and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.
- HDFS uses a **master/slave architecture** where master consists of a single Name Node that manages the file system metadata and one or more slave. Data Nodes that store the actual data.
- **Data Node** A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of Data Nodes. They store and retrieve blocks when they are told to; by client or name node. They report back to name node periodically, with list of blocks that they are storing. The data node being a commodity hardware also does the work of block creation, deletion and replication as stated by the name node.
- The **Name Node** determines the mapping of blocks to the Data Nodes. The Data Nodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode.

2.2 HDFS concepts

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly faulttolerant and designed using low-cost hardware. HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware.

The system having the namenode acts as the master server and it does the following tasks:

• Manages the file system namespace.

• Regulates client's access to files.

• It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

• Datanodes perform read-write operations on the file systems, as per client request.

• They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Goals of HDFS

• Fault detection and recovery :

Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.

• Huge datasets :

HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets. Hardware at data : A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

2.3 MapReduce

MapReduce is a programming model for writing applications that can process Big Data in parallel on multiple nodes. MapReduce provides analytical capabilities for analyzing huge volumes of complex data.

MapReduce is a processing technique and a program model for distributed computing based on java.

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

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

Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers.

2.4 Algorithms using MapReduce

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

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

• The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.

• Input Phase − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
• Map − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
• Intermediate Keys − They key-value pairs generated by the mapper are known as intermediate keys.
• Combiner − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
• Shuffle and Sort − The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.
• Reducer − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.
• Output Phase − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

Algorithm 1: The Map Function

1 **for** *each element* m_{ij} *of* M **do**
2 produce $(key, value)$ pairs as $((i,k),(M,j,m_{ij}))$, for $k = 1,2,3,..$ up to the number of columns of N
3 **for** *each element* n_{jk} *of* N **do**
4 produce $(key, value)$ pairs as $((i,k),(N,j,n_{jk}))$, for $i = 1,2,3,...$ up to the number of rows of M
5 **return** *Set of (key, value) pairs that each key,* (i,k)*, has a list with values* (M,j,m_{ij}) *and* (N,j,n_{jk}) *for all possible values of* j

Algorithm 2: The Reduce Function

1 **for** *each key (i,k)* **do**
2 sort values begin with M by j in $list_M$
3 sort values begin with N by j in $list_N$
4 multiply m_{ij} and n_{jk} for j_{th} value of each list
5 sum up $m_{ij} * n_{jk}$
6 **return** $(i,k), \sum_{j=1} m_{ij} * n_{jk}$

Matrix-Vector Multiplication
Multiplication is a fundamental nontrivial matrix operation

Problem: Some people want to use enormous matrices. Cannot be handled on one machine

Take advantage of map-reduce parallelism to approach this problem.

Heuristics:

10,000x10,000: 100,000,000 entries

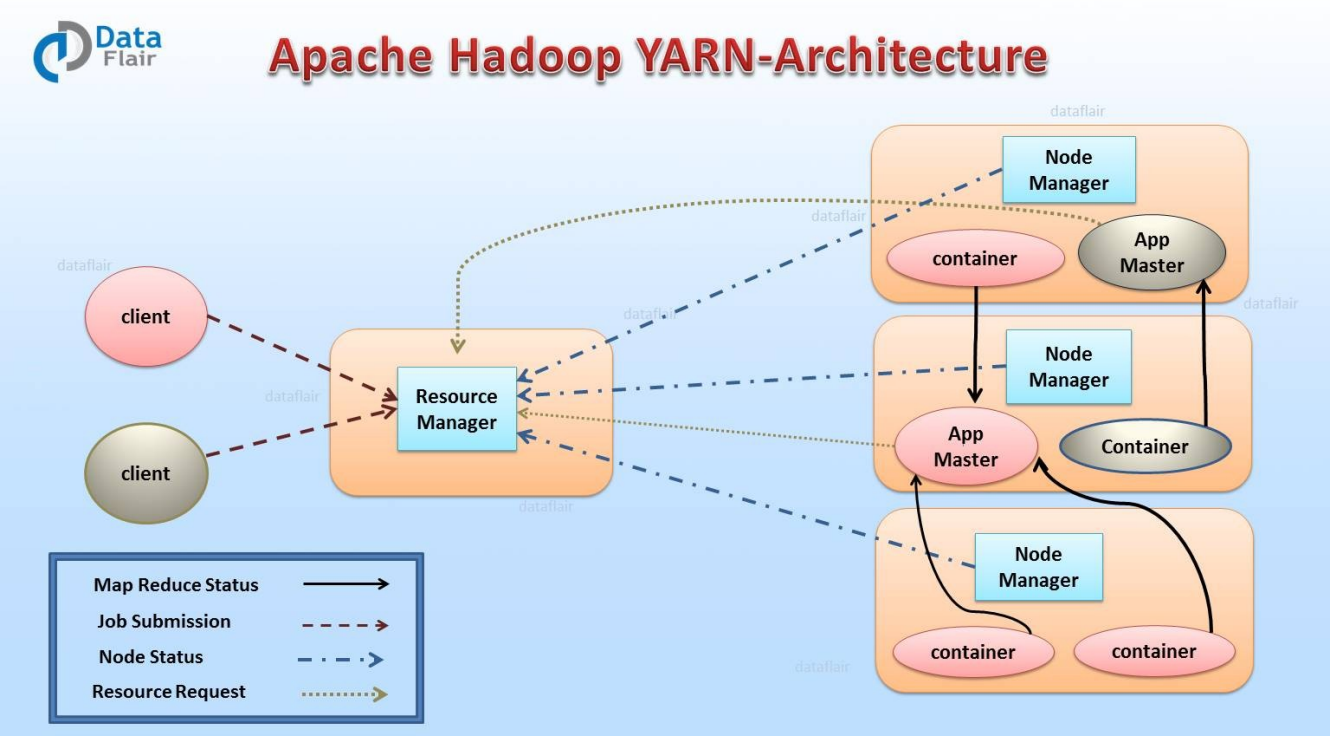
100,000x100,000: 10,000,000,000 entries

Hadoop YARN

YARN "**Y**et**A**nother**R**esource**N**egotiator" is the resource management layer of**Hadoop**. The Yarn was introduced in Hadoop 2.x. Yarn allows different data processing engines like graph processing, interactive processing, stream processing as well as batch processing to run and process data stored in **HDFS**(Hadoop Distributed File System). Apart from resource management, Yarn also does job Scheduling. Yarn extends the power of Hadoop to other evolving technologies, so they can take the advantages of HDFS (most reliable and popular storage system on the planet) and economic cluster.

Apache yarn is also a data operating system for Hadoop 2.x. This architecture of Hadoop 2.x provides a general purpose data processing platform which is not just limited to the**MapReduce**. It enables Hadoop to process other purpose-built data processing system other than MapReduce. It allows running several different frameworks on the same hardware where Hadoop is deployed.

Apache Yarn Framework consists of a master daemon known as "Resource Manager", slave daemon called node manager (one per slave node) and Application Master (one per application).



1. Resource Manager (RM)

It is the master daemon of Yarn. RM manages the global assignments of resources (CPU and memory) among all the applications. It arbitrates system resources between competing applications.follow Resource Managerguideto learn Yarn Resource manager in great detail.

Resource Manager has two Main components

- Scheduler
- Application manager

a) Scheduler

The scheduler is responsible for allocating the resources to the running application. The scheduler is pure scheduler it means that it performs no monitoring no tracking for the application and even doesn't guarantees about restarting failed tasks either due to application failure or hardware failures.

b) Application Manager

It manages running Application Masters in the cluster, i.e., it is responsible for starting application masters and for monitoring and restarting them on different nodes in case of failures.

2. Node Manager (NM)

It is the slave daemon of Yarn. NM is responsible for containers monitoring their resource usage and reporting the same to the ResourceManager. Manage the user process on that machine. Yarn NodeManager also tracks the health of the node on which it is running. The design also allows plugging long-running auxiliary services to the NM; these are application-specific services, specified as part of the configurations and loaded by the NM during startup. Ashuffle is a typical auxiliary service by the NMs for MapReduce applications on YARN

3. Application Master (AM)

One application master runs per application. It negotiates resources from the resource manager and works with the node manager. It Manages the application life cycle.

The AM acquires containers from the RM's Scheduler before contacting the corresponding NMs to start the application's individual tasks.

4. Resource Manager Restart

Resource Manager is the central authority that manages resources and schedules applications running on YARN. Hence, it is potentially an SPOF in an Apache YARN cluster.

There are two types of restart for Resource Manager:

- **Non-work-preserving RM restart –**This restart enhances RM to persist application/attempt state in a pluggable state-store. Resource Manager will reload the same info from state-store on the restart and re-kick the previously running apps. Users does not need to re-submit the applications. Node manager and clients during down time of RM will keep polling RM until RM comes up, when RM comes up, it will send a re-sync command to all the NM and AM it was talking to via heartbeats. The NMs will kill all its manager's containers and re-register with RM
- **Work-preserving RM restart –**This focuses on reconstructing the running state of RM by combining the container status from Node Managers and container requests from Application Masters on restart. The key difference from Non-work-preserving RM restart is that alreadyrunning apps will not be stopped after masterrestarts, so applications will not lose its processed databecause of RM/master outage. RM recovers its running state by taking advantage of container status which is sent from all the node managers. NM will not kill the containers when it re-syncs with the restarted RM. It continues managing the containers and sends the container status across to RM when it re-registers.

5. Yarn Resource Manager High availability

The ResourceManager (master) is responsible for handlingthe resources in a cluster, and scheduling multiple applications (e.g.,**spark**apps or MapReduce). Beforeto Hadoop v2.4, the master

(RM)was the SPOF (single point of failure). The High Availability feature adds redundancy in the form of an Active/Standby ResourceManager pair to remove this otherwise single point of failure.
ResourceManager HA is realized through an Active/Standby architecture – at any point in time, one in the mastersis Active, and otherResource Managers are in Standby mode, they are waiting to take over when anything happens to the Active. The trigger to transition-to-active comes from either the admin (through CLI) or through the integrated failover-controller when automatic failover is enabled.

6. Yarn Web Application Proxy

It is also the part of Yarn. By default, it runs as a part of RM but we can configure and run in a standalone mode. Hence, the reason of the proxy is to reduce the possibility of the web-based attack through Yarn.

In Yarn, the AM has a responsibility to provide a web UI and send that link to RM. RM runs as trusted user, and provide visiting that web address will treat it and link it provides to them as trusted when in reality the AM is running as non-trusted user, application Proxy mitigate this risk by warning the user that they are connecting to an untrusted site.

7. Yarn Docker Container Executor

Docker combines an easy to use interface to**Linux**container with easy to construct files for those containers. Docker generates light weighted virtual machine. The Docker Container Executor allows the Yarn NodeManager to launch yarn container to Docker container. hence, these containers provide a custom software environment in which user's code run, isolated from a software environment of NodeManager.

Hence, Docker for YARN provides both consistency (all YARN containers will have similarenvironment) and isolation (no interference with other componentsinstalled on the same machine).

8. Yarn Timeline Server

The storage and retrieval of application's current and historic information in a generic fashion is addressed by the timeline service in Yarn.

8.1. Persisting application specific information

The collection or retrieval of information completely specific to a specific application or framework. For Example, Hadoop MapReduce framework consists the pieces of information about the map task, reduce task and counters. Application developer publishes their specific information to the Timeline Server via TimeLineClient in the application Master or application container.

8.2. Persisting general information about completed applications

Generic information includes application-level data such as:

- Queue-name
- User information and the like set in the ApplicationSubmissionContext
- Info about each application-attempt
- A list of application-attempts that ran for an application
- The list of containers run under each application-attempt
- Information about each container

9. Yarn Timeline service version 2

It is the major iteration of the timeline server. Thus, V2 addresses two major challenges:

- The previous version does not well scale up beyond small cluster.
- And single instance available for the write and read.

Hence, In the v2 there is a different collector for write and read, it uses distributed collector, one collector for each Yarn application.

Hadoop Installation Steps

1)Add a Hadoop system user using below command

sudo addgroup hadoop

sudo adduser --ingroup hadoop hduser

Note:

"hduser is not in the sudoers file. This incident will be reported."This error can be resolved by

sudo adduser hduser sudo

2) Install and Configure SSH

In order to manage nodes in a cluster, Hadoop require SSH access

First, switch user, enter following command

su – hduser

Installing SSH

SSH ("Secure SHell") is a protocol for securely accessing one machine from another. Hadoop uses SSH for accessing another slaves nodes to start and manage all HDFS and MapReduce daemons.

Command to install SSH

sudo apt-get install openssh-server openssh-client

Configuring SSH

Generate ssh key for hduser account

ssh-keygen -t rsa -P ""

Copy id_rsa.pub to authorized keys from hduser which Enable SSH access to local machine using this key.

cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys

Now test SSH setup by connecting to locahost as 'hduser' user.

ssh localhost.

Step 3) Download Hadoop

Download and copy the hadoop-2.6.1 package to home directory

cd /home/vvcoe

Extract Hadoop source

hduser@osl-003:/home/vvcoe$ sudo tar xvzf hadoop-2.6.1.tar.gz

hduser@osl-003:/home/vvcoe$cd

create a directory path /usr/local/hadoop

hduser@osl-003:~$ sudo mkdir /usr/local/hadoop

Move extracted hadoop-2.6.1 to /usr/local/hadoop and check it.

hduser@osl-003:~$ cd /home/vvcoe

hduser@osl-003:/home/vvcoe$ sudo mv hadoop-2.6.1/* /usr/local/hadoop

hduser@osl-003:/home/vvcoe$ cd /usr/local/hadoop

hduser@osl-003:/usr/local/hadoop$ ls

bin etc include lib libexec LICENSE.txt NOTICE.txt README.txt sbin sha

Assign ownership of this folder to Hadoop user

hduser@osl-003:/usr/local/hadoop$sudo chown hduser:hadoop -R /usr/local/hadoop

Create Hadoop temp directories for Namenode and Datanode

hduser@osl-003:/usr/local/hadoop$

sudo mkdir -p /usr/local/hadoop_tmp/hdfs/namenode

hduser@osl-003:/usr/local/hadoop$

sudo mkdir -p /usr/local/hadoop_tmp/hdfs/datanode

Again assign ownership of this Hadoop temp folder to Hadoop user
hduser@osl-003:/usr/local/hadoop$
sudo chown hduser:hadoop -R /usr/local/hadoop_tmp/

hduser@osl-003:/usr/local/hadoop$cd

Step 4:Update Hadoop configuration files
Find the path to Java bin and set it in the environment variables
hduser@osl-003:~$ sudo gedit ~/.bashrc

Update hduser configuration file by appending the
following environment variables at the end of this file.
-- HADOOP ENVIRONMENT VARIABLES START --
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
-- HADOOP ENVIRONMENT VARIABLES END --
Source your file, to apply the changes to the system shell:
hduser@osl-003:~$ source ~/.bashrc
Step 5: Configuration file :hadoop-env.sh
hduser@osl-003:~$ cd /usr/local/hadoop/etc/hadoop
hadoop@hadoop:/usr/local/hadoop/etc/hadoop$ sudo gedit hadoop-env.sh
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
Step 6: Configuration file : core-site.xml
hadoop@hadoop:/usr/local/hadoop/etc/hadoop$ sudo gedit core-site.xml
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
Step 7: Configuration file : hdfs-site.xml
Set up the environment variables for Hadoop
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/usr/local/hadoop_tmp/hdfs/namenode</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file:/usr/local/hadoop_tmp/hdfs/datanode</value>

</property>

Step 8: Configuration file : yarn-site.xml

hadoop@hadoop:/usr/local/hadoop/etc/hadoop$ sudo gedit yarn-site.xml

```
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
<property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

Step 9:

Format Namenode

hadoop@hadoop:/usr/local/hadoop/etc/hadoop$ cd

hadoop@hadoop:~$ hdfs namenode -format

15/04/18 14:43:12 INFO util.ExitUtil: Exiting with status 0
15/04/18 14:43:12 INFO namenode.NameNode: SHUTDOWN_MSG:
/**
SHUTDOWN_MSG: Shutting down NameNode at laptop/192.168.1.1
**/

Step 10:: Start all Hadoop daemons

Start hdfs daemons

hadoop@hadoop:~$ start-dfs.sh

hadoop@hadoop:~$ start-yarn.sh

Instead both of these above command you can also use start-all.sh, but its now deprecated so its not recommended to be used for better Hadoop operations. Track/Monitor/Verify

Step 11:Verify Hadoop daemons: Check if all services are running in the cluster using the command 'jps'

hadoop@hadoop:~$ jps

9026 NodeManager
7348 NameNode
9766 Jps
8887 ResourceManager
7507 DataNode
5647 SecondryNode

Step12 :Check the Browser for the HDFS Manager and Cluster Manager:

Click the following URL for ResourceManager and NameNode.

- HDFS Manager /Name node is available at http://localhost:50070.
- ResourceManager / Cluster Manager is available at http://localhost:8088