Unit IV : Hadoop Environment

Setting up a Hadoop Cluster - Cluster specification - Cluster Setup and Installation – Hadoop Configuration-Security in Hadoop - Administering Hadoop – HDFS – Monitoring-Maintenance-Hadoop benchmarks- Hadoop in the cloud

Hadoop Cluster

- Hadoop cluster is just a computer cluster used for handling huge volume of data distributedly.
- Hadoop Clusters are designed to store and analyse huge amount of unstructured data in a distributed computing enivironment.
  - A collection of nodes is called cluster.
  - A node is a point of intersection/connection within a network, ie a server
- Hadoop clusters have two types of machines,
  - Master: HDFS NameNode, YARN ResourceManager.
  - Slaves: HDFS DataNodes, YARN NodeManagers.
- Master and Slave nodes should be separated, because
  - Task/application workloads on the slave nodes should be isolated from the masters.
  - Slaves nodes are frequently decommissioned for maintenance.

Advantages of a Hadoop Cluster

- The cluster helps in increasing the speed of the analysis process.
- It is inexpensive.
- These clusters are failure resilient.
- Hadoop clusters are “scalability”, we can scale a Hadoop cluster by adding new servers to the cluster if needed.
- Hadoop Clusters deal with data from many sources and formats in a very quick, easy manner.
- It is possible to deploy Hadoop using a single-node installation, for evaluation purposes.

Apache Ambari and Cloudera Manager are two important tools used for Hadoop Cluster Management.

Setting up a Hadoop Cluster

- Hadoop is designed to run on commodity hardware.

- *A typical choice of machine for running a Hadoop datanode and tasktracker in late 2008 would have the following specifications:
  - Processor -2 quad-core Intel Xeon 2.0GHz CPUs
  - Memory-8 GB ECC RAM
  - Storage-41 TB SATA disks
  - Network-Gigabit Ethernet

Best Practices

Building a Hadoop cluster is a complex task that requires consideration of several factors like choosing the right hardware, sizing the hadoop cluster and configuring it correctly.

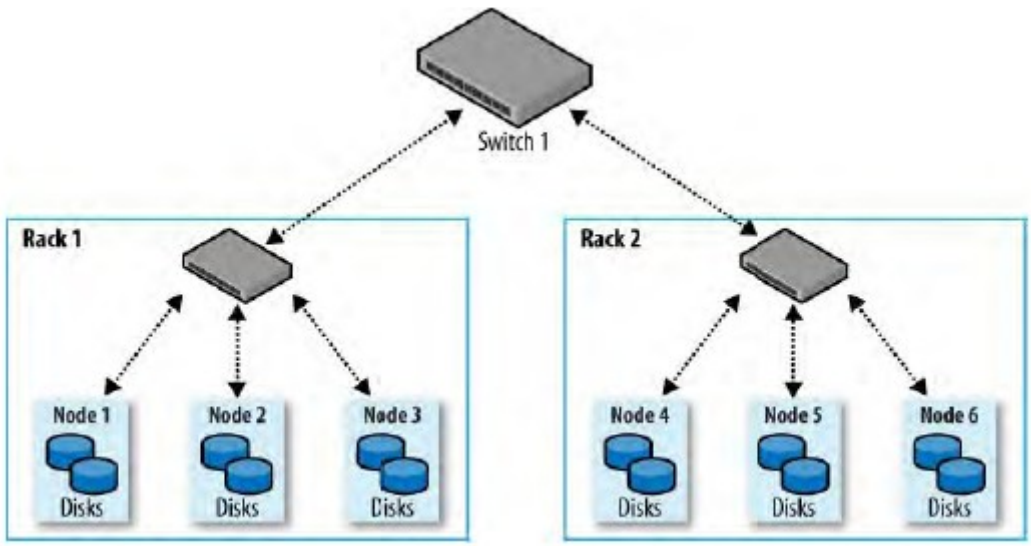
- **Choosing the Right Hardware for a Hadoop Cluster**
  - Thorough testing and validation is required for choosing a right hardware for hadoop cluster, to fully optimize it.
  - The no. Of machines and hardware specifications depend on: Volume of the Data, type of workload that needs to be processed, Data storage methodology (Data container , data compression technique used , if any), Data retention policy (How long can you afford to keep the data before flushing it out)

- **Sizing a Hadoop Cluster**
  - Best practice to size a hadoop cluster is sizing it based on the amount of storage required.

- **Configuring the Hadoop Cluster**
  - Finding the ideal configuration for a hadoop cluster is not easy.
  - The best way is to run hadoop jobs with the default configuration available.
  - Analyze the job history to see whether any resource is weak or if the time taken to run the jobs is higher than expected.
  - Repeat the process to fine tune the cluster configuration.
  - The number of CPU cores and memory resources that need to be allocated to the daemons also has a great impact on the performance of the cluster.

***Network Topology**

- A common Hadoop architecture consists of a two-layer network topology



  There are 30 to 40 servers per rack, with a 1 GB switch for the rack.
- An uplink to a core switch or router typically 1 GB.

***Rack Awareness**

- Hadoop components are rack-aware.
- It is important to configure Hadoop so that it knows the topology of your network.
- If your cluster runs on a single rack, then there is nothing more to do, since this is the default.
- For multirack clusters, map nodes to racks.

- HDFS block placement will use rack awareness for fault tolerance by placing one block replica on a different rack.
- Network locations such as nodes and racks are represented in a tree, which reflects the network "distance" between locations.
- For the network in the above figure, the rack topology is described by two network locations, say, /switch1/rack1 and /switch1/rack2. Or simply /rack1 and /rack2.
- **The namenode uses the network location when determining where to place block replicas.**
- **The jobtracker uses network location to determine where the closest replica is as input for a map task that is scheduled to run on a tasktracker.**

Mapping node address and network Locations

- *The map between node addresses and network locations is described by a Java interface, **DNSToSwitchMapping.**

```
public interface DNSToSwitchMapping {
        public List<String> resolve(List<String> names);
                }
```

  - names parameter-List of IP addresses
  - the return value is a list of corresponding network location strings.
    For the network in example figure, we can map node1, node2, and node3 to /rack1, and node4, node5, and node6 to /rack2.

*Cluster Setup and Installation

Hadoop installation and configration canbe done in a no.of ways.

- *From scratch using the Apache Hadoop distribution
- *Start with Cloudera's Distribution by using RPMs and Debian packages.

Pre-installation Setup

Before installing Hadoop , there are 2 prerequisites

- Set up Linux environment
- Install Java

Setting Up Linux Environment

Create a Hadoop User

- Create a separate Hadoop user and separate the Hadoop installation from other services running on the machine.
- User's home directory can be an NFS mounted drive.
- If NFS is used, user can mount the NFS filesystem on demand, when the system accesses it.
- It also provides some protection against the NFS server failing.
- It also allows to use replicated filesystems for failover.

SSH Set up and Key Generation

- SSH setup is required to do different operations on a cluster such as starting, stopping, distributed daemon shell operations.

- SSH needs to be set up to allow password-less login for the hadoop user from machines in the cluster.
- The simplest way to achieve this is to generate a public/private key pair, and it will be shared across the cluster using NFS.[1]

Installing Java

- Java 6 or later is required to run Hadoop.
- First verify whether Java is already installed in the machine using the following commands:

```
% java -version
java version "1.6.0_12"
Java(TM) SE Runtime Environment (build 1.6.0_12-b04)
Java HotSpot(TM) 64-Bit Server VM (build 11.2-b01, mixed mode)
```

- If Java is not there follow the below steps to install Java.

  Step 1- Download the latest version of Java.

  Step 2- Extract using the tar zxf command.

  Step 3-Move it to the location "/usr/local/",to make it available for all the users

  Step 4- For setting up **PATH** and **JAVA_HOME** variables, add the following commands to **~/.bashrc** file.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71
export PATH=$PATH:$JAVA_HOME/bin
```

  Step 5- Now apply all the changes into the current running system.

```
$ source ~/.bashrc
```

After setting up the above environment download hadoop and install it.

Installing Hadoop

- Download Hadoop from the Apache Hadoop releases page
  - http://hadoop.apache.org/core/releases.html
- Unpack the contents of the distribution in a sensible location, such as /usr/local or /opt.
  - `cd /usr/local`
  - `sudo tar xzf hadoop-x.y.z.tar.gz`
- Change the owner of the Hadoop files to be the hadoop user and group.
  - `sudo chown -R hadoop:hadoop hadoop-x.y.z`

1 A distributed file system protocol that allows access to files on a remote computer in a manner similar to how a local file system is accessed.

***Hadoop Configuration**

The following table shows the files for controlling the configuration of a Hadoop installation.

- ***hadoop-env.sh*** Bash script Environment variables that are used in the scripts to run Hadoop.
- ***core-site.xml*** Hadoop configuration XML Configuration settings for Hadoop Core, such as I/O settings that are common to HDFS and MapReduce.
- ***hdfs-site.xml*** Hadoop configuration XML Configuration settings for HDFS daemons: the namenode, the secondary namenode, and the datanodes.
- ***mapred-site.xml*** Hadoop configuration XML Configuration settings for MapReduce daemons: the jobtracker, and the tasktrackers.
- ***masters*** Plain text A list of machines (one per line) that each run a secondary namenode.
- ***slaves*** Plain text A list of machines (one per line) that each run a datanode and a tasktracker.
- ***hadoop-metrics.properties*** Java Properties Properties for controlling how metrics are published in Hadoop
- ***log4j.properties*** Java Properties Properties for system logfiles, the namenode audit log, and the task log for the tasktracker child process

All these files are found in the *conf* directory of Hadoop distribution.

- Each node in the cluster will have its own configuration files.
- *Administrators synchronize them across the systems, using *rsync.*
- *"rsync"* utility can be used to replicated data between HDFS clusters.
- *During the expansion of the clusters, with new machines of different hardware sepcification different configuration has to be used.
- This helps to take the full advantage of the newly added resources.
- In such cases the concept of *class* of machine have to be used.
- Maintain a separate configuration files for each class.
- There are several other open source configuration management tools for doing this,for example, Puppet, cfEngine, bcfg2.

*__Control Scripts__

- Used for running commands, starting and stopping deamons across the whole cluster.
- The scripts are found in the *bin* directory.
- *Two files, masters and slaves, contains list of host machines or IP addresses in the cluster.
- *The masters file determines which machine or machines should run a secondary namenode.
- *The slaves file lists the machines that the datanodes and tasktrackers should run on.
- *The *start-dfs.sh* script, starts all the HDFS daemons in the cluster, runs the namenode on the machine the script is run on.
  - Starts a namenode on the local machine (the machine that the script is run on).
  - Starts a datanode on each machine listed in the slaves file.
  - Starts a secondary namenode on each machine listed in the masters file.
- **start-mapred.sh* starts all the MapReduce daemons in the cluster.
  - Starts a jobtracker on the local machine.
  - Starts a tasktracker on each machine listed in the slaves file.
- *stop-dfs.sh and stop-mapred.sh scripts are used to stop the daemons started by the script.

***Configurations for running master daemons in a Hadoop cluster**

- The namenode, secondary namenode, and jobtracker are the master daemons.
- For a small sized cluster these daemons can be put on a single machine.
- But for large clusters, it is better to separate them.
- The namenode must have a larger memory since it holds the files and meta data in the entire namespace.
- Two points must be followed on running master daemons:
  - Run the HDFS control scripts from the namenode machine.
  - The masters file should contain the address of the secondary namenode.
  - Run the MapReduce control scripts from the jobtracker machine.

***Environment Setting**

- Set the variables in hadoop-env.sh.

Variables	Purpose
HADOOP_HEAPSIZE	For allocating 1 GB of memory to each daemon it runs.
HADOOP_NAMENODE_OPTS	For increasing the namenode's memory without changing the memory allocated to other Hadoop daemons.
HADOOP_SECONDARYNAMENODE_OPTS	For changing the secondary namenode's memory allocation.
JAVA_HOME	For determining the location of the Java implementation
HADOOP_LOG_DIR	To change the location of the System log files produced by Hadoop. By default System logfiles produced by Hadoop are stored in $HADOOP_INSTALL/logs. `export HADOOP_LOG_DIR=/var/log/hadoop`
HADOOP_SSH_OPTS	To pass extra options to SSH.Empty by default.
HADOOP_MASTER	host:path where hadoop code should be rsync'd from. Unset by default.
ConnectTimeout option	SSH setting. To reduce the connection timeout
StrictHostKeyChecking	SSH setting.set to no to auto-matically add new host keys to the known hosts files.

*Important Hadoop Daemon Properties

These properties are set in the Hadoop site files: **core-site.xml, hdfs-site.xml,** and **mapred-site.xml.**

HDFS

- fs.default.name is a HDFS filesystem URI, whose host is the namenode's hostname or IP address.
- The default port is 8020.

- The property dfs.name.dir specifies a list of directories where the namenode stores.
- dfs.data.dir property specifies a list of directories for a datanode to store its blocks.
- The fs.checkpoint.dir property specifies a list of directories where the checkpoints are kept.

Property name	Type	Default value	Description
`fs.default.name`	URI	`file:///`	The default filesystem. The URI defines the hostname and port that the jobtracker's RPC server runs on. The default port is 8020. This property should be set in *core-site.xml*.
`dfs.name.dir`	comma-separated directory names	`${hadoop.tmp.dir}/dfs/name`	The list of directories where the namenode stores its persistent metadata. The namenode stores a copy of the metadata in each directory in the list.
`dfs.data.dir`	comma-separated directory names	`${hadoop.tmp.dir}/dfs/data`	A list of directories where the datanode stores blocks.
`fs.checkpoint.dir`	comma-separated directory names	`${hadoop.tmp.dir}/dfs/namesecondary`	A list of directories where the secondary namenode stores checkpoints. It stores a copy of the checkpoint in each directory in the list.

MapReduce

- To run MapReduce, you need to designate one machine as a jobtracker.
- To do this, set the mapred.job.tracker property to the hostname or IP address and port that the jobtracker will listen on.

Property name	Type	Default value	Description
`mapred.job.tracker`	hostname and port	`local`	The hostname and port that the jobtracker's RPC server runs on. If set to the default value of `local`, then the jobtracker is run in-process on demand when you run a MapReduce job (you don't need to start the MapReduce daemons in this case).
`mapred.local.dir`	comma-separated directory names	`${hadoop.tmp.dir}/mapred/local`	A list of directories where the MapReduce stores intermediate data for jobs. The data is cleared out when the job ends.
`mapred.system.dir`	URI	`${hadoop.tmp.dir}/mapred/system`	The directory relative to `fs.default.name` where shared files are stored, during a job run.
`mapred.tasktracker.map.tasks.maximum`	`int`	`2`	The number of map tasks that may be run on a tasktracker at any one time.

mapred.task tracker.reduce.tasks. maximum	int	2	The number of reduce tasks that may be run on a tasktracker at any one time.
mapred.child.java.opts	String	-Xmx200m	The JVM options used to launch the tasktracker child process that runs map and reduce tasks. This property can be set on a per-job basis, which can be useful for setting JVM properties for debugging, for example.

Other Properties

- ***Cluster Memberships**
  - Inorder to add and remove nodes in future, you can specify a list of authorized machines that may join the cluster as datanodes or tasktrackers.
  - The list is specified using the dfs.hosts (for datanodes) and mapred.hosts (for tasktrackers) properties.
  - The corresponding dfs.hosts.exclude and mapred.hosts.exclude files used for decommissioning.

- ***Trash**
  - In Hadoop deleted files are not actually deleted, but rather are moved to a trash folder.
  - In trash they remain for a minimum period before being permanently deleted by the system.
  - The minimum period in minutes that the file should be remained in the trash can be specified by setting the fs.trash.interval configuration property in core-site.xml.
  - By default the trash interval is zero, which disables trash.
  - It is possible to use the trash programmatically, by constructing a `Trash` instance, then calling its `moveToTrash()` method with the `Path` of the file intended for deletion.
  - The method returns a value indicating success; a value of `false` means either that trash is not enabled or that the file is already in the trash.
  - When trash is enabled, each user has her own trash directory called *`.Trash`* in the home directory.
  - *****You can expunge the trash, which will delete files that have been in the trash longer than their minimum period, using the filesystem shell:

```
% hadoop fs -expunge
```

* Creation of User

- Create a home directory for each user, and setting ownership permissions on it:

```
% hadoop fs -mkdir /user/username
% hadoop fs -chown username:username /user/username
```

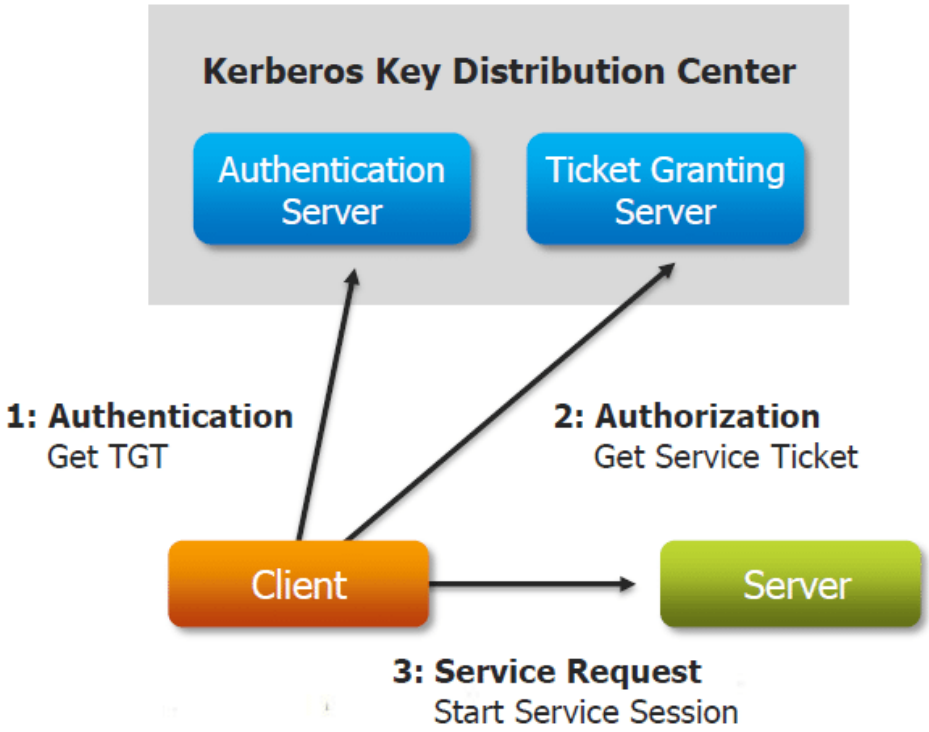
- Set space limits on the directory. The following sets a 1 TB limit on the given user directory:

```
% hadoop dfsadmin -setSpaceQuota 1t /user/username
```

Kerberos

- Kerberos is a network authentication protocol designed to provide strong authentication for client/server applications by means of secret-key cryptography.

Components of Kerberos



- Kerberos comprises of 3 components; Key Distribution Center (KDC), Client User and Server with the desired service to access. The KDC performs 2service functions:
    - Authentication Service (AS)
    - Ticket-Granting Service (TGS)
- Three exchanges occurs when the client accesses a server:
    - AS Exchange
    - TGS Exchange
    - Client/Server (CS) Exchange

How Does Kerberos Work?

- A Request Ticket is placed from authentication server
- The Ticket along with the encrypted request is sent to application server .
- Tickets can be requested without repeatedly sending credentials through Ticket granting ticket (TGT).

***Hadoop Security Design With Kerberos:**

The new Hadoop security design makes use of Delegation Tokens, Job Tokens and Block Access Tokens in Kerberos. Each of these tokens is similar in structure.

- Delegation Tokens – Used for clients to communicate with the NameNode to gain access to HDFS data.
- Block Access Tokens – Used to secure communication between the NameNode and DataNodes to implement HDFS file system permissions.
- The Job Token – Used to secure communications between the MapReduce engine, Task Tracker and individual tasks.

***Steps that a client must take to accept the service**

At a high level, there are three steps that a client must take to access a service when using Kerberos, each of which involves a message exchange with a server:

1. **Authentication**. The client authenticates itself to the Authentication Server and receives a timestamped Ticket-Granting Ticket (TGT).
2. **Authorization**. The client uses the TGT to request a service ticket from the Ticket Granting Server.
3. **Service Request**. The client uses the service ticket to authenticate itself to the server that is providing the service the client is using. In the case of Hadoop, this might be the namenode or the jobtracker.

***Delegation Tokens**

- In order to access a service when using Kerberos, the client must go through a three-step Kerberos ticket exchange protocol to authenticate each call.
- This may present a high load on a key distribution centre (KDC) on a busy cluster.
- To avoid this, Hadoop uses ***delegation tokens*** to allow later authenticated access without having to contact the KDC again.
- A delegation token is generated by the server (the namenode in this case), and can be thought of as a shared secret between the client and the server.
- On the first RPC call to the namenode, the client has no delegation token, so it uses Kerberos to authenticate, and as a part of the response it gets a delegation token from the namenode.
- In subsequent calls, it presents the delegation token, which the namenode can verify (since it
- generated it using a secret key), and hence the client is authenticated to the server.
- When it wants to perform operations on HDFS blocks, the client uses a special kind of delegation token, called a ***block access token***, that the namenode passes to the client in response to a metadata request.
- The client uses the block access token to authenticate itself to datanodes.