Unit-IV

HADOOP

Introduction to Hadoop – Hadoop Distributed File System – Analysing data with Hadoop – Scaling – Streaming – Clustering: Single Node and Multi Node – Working with Hadoop Commands – Working with Apache Oozie.

What is Hadoop?

Apache Hadoop is an open source software framework used to develop data processing applications which are executed in a distributed computing environment.

Applications built using HADOOP are run on large data sets distributed across clusters of commodity computers. Commodity computers are cheap and widely available. These are mainly useful for achieving greater computational power at low cost.

Similar to data residing in a local file system of a personal computer system, in Hadoop, data resides in a distributed file system which is called as a **Hadoop Distributed File system**. The processing model is based on **'Data Locality'** concept wherein computational logic is sent to cluster nodes(server) containing data. This computational logic is nothing, but a compiled version of a program written in a high-level language such as Java. Such a program, processes data stored in Hadoop HDFS.

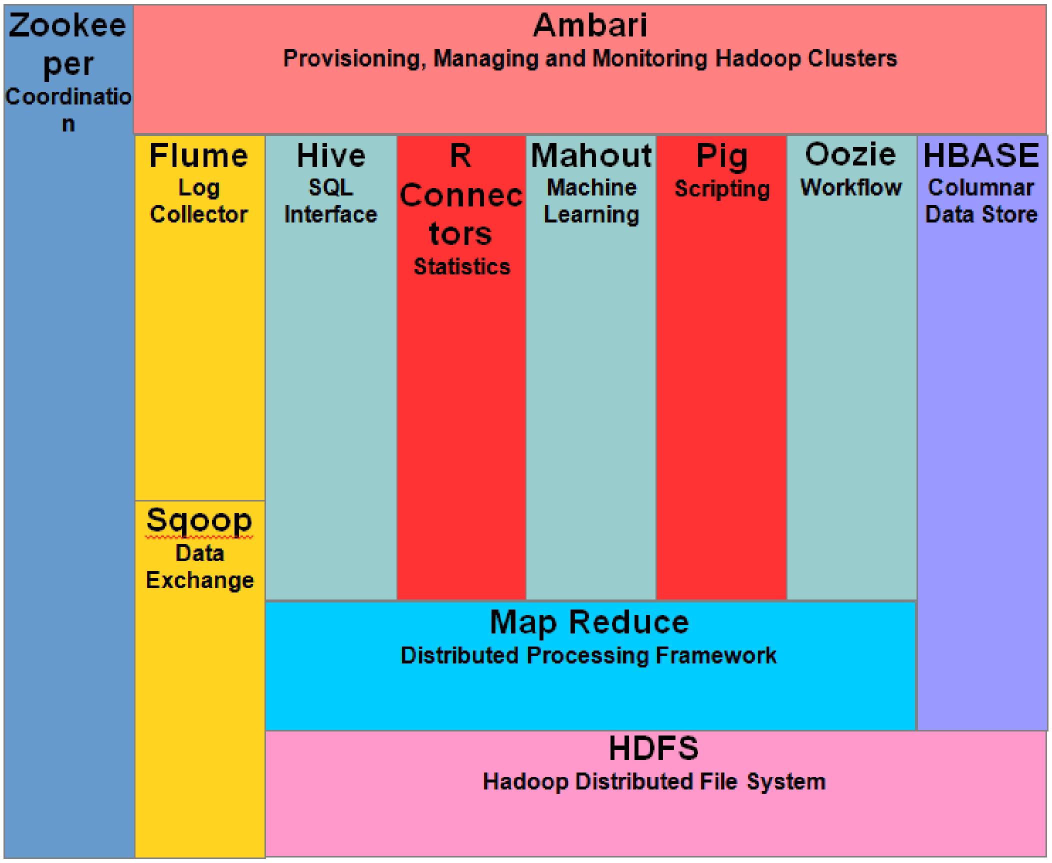
Hadoop EcoSystem and Components

Apache Hadoop consists of two sub-projects –

1. **Hadoop MapReduce:** MapReduce is a computational model and software framework for writing applications which are run on Hadoop. These MapReduce programs are capable of processing enormous data in parallel on large clusters of computation nodes.
2. **HDFS** (**Hadoop Distributed File System**): HDFS takes care of the storage part of Hadoop applications. MapReduce applications consume data from HDFS. HDFS creates multiple replicas of data blocks and distributes them

on compute nodes in a cluster. This distribution enables reliable and extremely rapid computations.

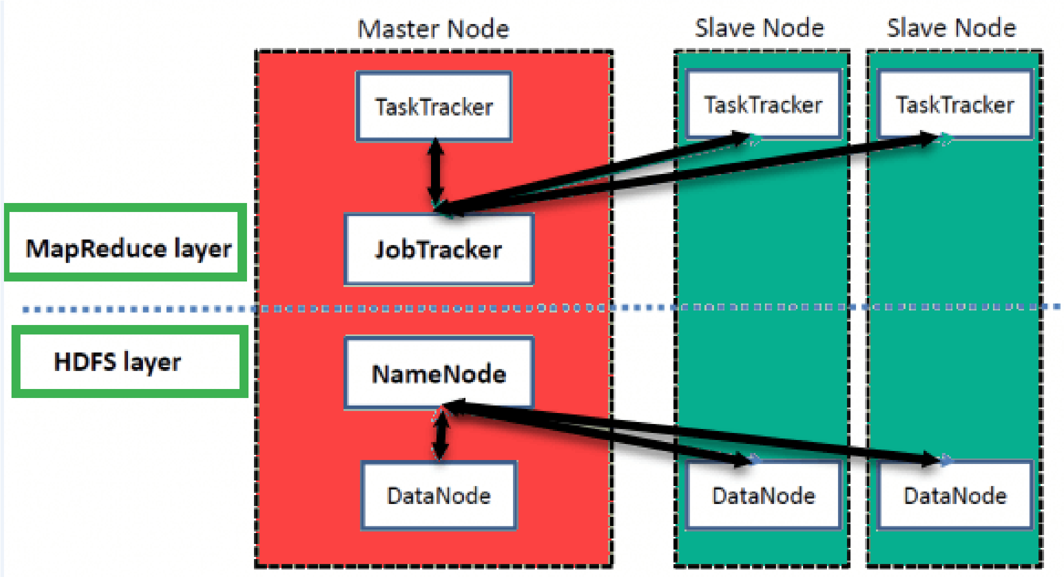
Below diagram shows various components in the Hadoop ecosystem-



Although Hadoop is best known for MapReduce and its distributed file system-HDFS, the term is also used for a family of related projects that fall under the umbrella of distributed computing and large-scale data processing. Other Hadoop-related projects at Apache include are **Hive, HBase, Mahout, Sqoop, Flume, and ZooKeeper.**

Hadoop Architecture

Hadoop has a Master-Slave Architecture for data storage and distributed data processing using MapReduce and HDFS methods.



NameNode:

NameNode represented every files and directory which is used in the namespace

DataNode:

DataNode helps you to manage the state of an HDFS node and allows you to interacts with the blocks

MasterNode:

The master node allows you to conduct parallel processing of data using Hadoop MapReduce.

Slave node:

The slave nodes are the additional machines in the Hadoop cluster which allows you to store data to conduct complex calculations. Moreover, the entire slave node comes with Task Tracker and a DataNode. This allows you to synchronize the processes with the NameNode and Job Tracker respectively. In Hadoop, master or slave system can be set up in the cloud or on-premise

Features Of ‘Hadoop’

• Suitable for Big Data Analysis

As Big Data tends to be distributed and unstructured in nature, HADOOP clusters are best suited for analysis of Big Data. Since it is processing logic (not the actual data) that flows to the computing nodes, less network bandwidth is consumed. This concept is called as **data locality concept** which helps increase the efficiency of Hadoop based applications.

• Scalability

HADOOP clusters can easily be scaled to any extent by adding additional cluster nodes and thus allows for the growth of Big Data. Also, scaling does not require modifications to application logic.

• Fault Tolerance

HADOOP ecosystem has a provision to replicate the input data on to other cluster nodes. That way, in the event of a cluster node failure, data processing can still proceed by using data stored on another cluster node.

Network Topology In Hadoop

Topology (Arrangment) of the network, affects the performance of the Hadoop cluster when the size of the Hadoop cluster grows. In addition to the performance, one also needs to care about the high availability and handling of failures. In order to achieve this Hadoop, cluster formation makes use of network topology.

Typically, network bandwidth is an important factor to consider while forming any network. However, as measuring bandwidth could be difficult, in Hadoop, a network is represented as a tree and distance between nodes of this tree (number of hops) is considered as an important factor in the formation of Hadoop cluster. Here, the distance between two nodes is equal to sum of their distance to their closest common ancestor.

Hadoop cluster consists of a data center, the rack and the node which actually executes jobs. Here, data center consists of racks and rack consists of nodes. Network bandwidth available to processes varies depending upon the location of the processes. That is, the bandwidth available becomes lesser as we go away from-

- Processes on the same node
- Different nodes on the same rack
- Nodes on different racks of the same data center
- Nodes in different data centers

Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS) is the primary data storage system used by Hadoop applications. HDFS employs a NameNode and DataNode architecture to implement a distributed file system that provides high-performance access to data across highly scalable Hadoop clusters.

Hadoop itself is an open source distributed processing framework that manages data processing and storage for big data applications. HDFS is a key part of the many Hadoop ecosystem technologies. It provides a reliable means for managing pools of big data and supporting related big data analytics applications.

How does HDFS work?

HDFS enables the rapid transfer of data between compute nodes. At its outset, it was closely coupled with MapReduce, a framework for data processing that filters and divides up work among the nodes in a cluster, and it organizes and condenses the results into a cohesive answer to a query. Similarly, when HDFS takes in data, it breaks the information down into separate blocks and distributes them to different nodes in a cluster.

With HDFS, data is written on the server once, and read and reused numerous times after that. HDFS has a primary NameNode, which keeps track of where file data is kept in the cluster.

HDFS also has multiple DataNodes on a commodity hardware cluster -- typically one per node in a cluster. The DataNodes are generally organized within the same rack in the data center. Data is broken down into separate blocks and distributed among the various DataNodes for storage. Blocks are also replicated across nodes, enabling highly efficient parallel processing.

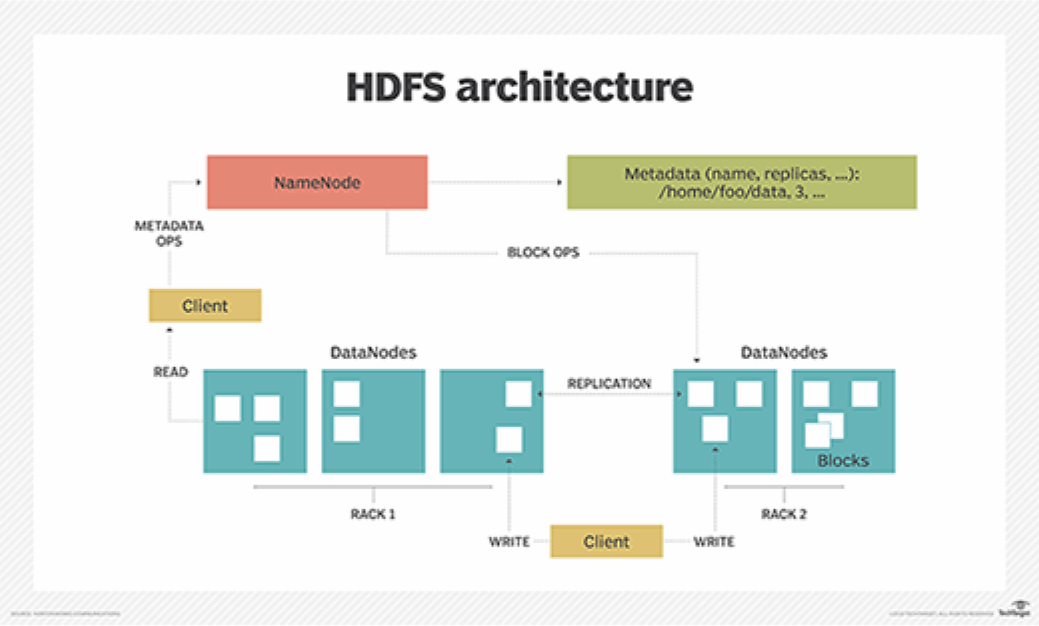
The NameNode knows which DataNode contains which blocks and where the DataNodes reside within the machine cluster. The NameNode also manages access to the files, including reads, writes, creates, deletes and the data block replication across the DataNodes.

The NameNode operates in conjunction with the DataNodes. As a result, the cluster can dynamically adapt to server capacity demand in real time by adding or subtracting nodes as necessary.

The DataNodes are in constant communication with the NameNode to determine if the DataNodes need to complete specific tasks. Consequently, the NameNode is always aware of the status of each DataNode. If the NameNode realizes that one DataNode isn't working properly, it can immediately reassign that DataNode's task to a different node containing the same data block. DataNodes also communicate with each other, which enables them to cooperate during normal file operations.

Moreover, the HDFS is designed to be highly fault-tolerant. The file system replicates -- or copies -- each piece of data multiple times and distributes the copies to individual nodes, placing at least one copy on a different server rack than the other copies.

HDFS architecture centers on commanding NameNodes that hold metadata and DataNodes that store information in blocks. Working at the heart of Hadoop, HDFS can replicate data at great scale.



HDFS architecture, NameNodes and DataNodes

HDFS uses a primary/secondary architecture. The HDFS cluster's NameNode is the primary server that manages the file system namespace and controls client access to files. As the central component of the Hadoop Distributed File System, the NameNode maintains and manages the file system namespace and provides clients with the right access permissions. The system's DataNodes manage the storage that's attached to the nodes they run on.

HDFS exposes a file system namespace and enables user data to be stored in files. A file is split into one or more of the blocks that are stored in a set of DataNodes. The NameNode performs file system namespace operations, including opening, closing and renaming files and directories. The NameNode also governs the mapping of blocks to the DataNodes. The DataNodes serve read and write requests from the clients of the file system. In addition, they perform block creation, deletion and replication when the NameNode instructs them to do so.

HDFS supports a traditional hierarchical file organization. An application or user can create directories and then store files inside these directories. The file system namespace hierarchy is like most other file systems -- a user can create, remove, rename or move files from one directory to another.

The NameNode records any change to the file system namespace or its properties. An application can stipulate the number of replicas of a file that the HDFS should maintain. The NameNode stores the number of copies of a file, called the replication factor of that file.

Features of HDFS

There are several features that make HDFS particularly useful, including:

- **Data replication.** This is used to ensure that the data is always available and prevents data loss. For example, when a node crashes or there is a hardware failure, replicated data can be pulled from elsewhere within a cluster, so processing continues while data is recovered.
- **Fault tolerance and reliability.** HDFS' ability to replicate file blocks and store them across nodes in a large cluster ensures fault tolerance and reliability.
- **High availability.** As mentioned earlier, because of replication across notes, data is available even if the NameNode or a DataNode fails.
- **Scalability.** Because HDFS stores data on various nodes in the cluster, as requirements increase, a cluster can scale to hundreds of nodes.
- **High throughput.** Because HDFS stores data in a distributed manner, the data can be processed in parallel on a cluster of nodes. This, plus data locality (see next bullet), cut the processing time and enable high throughput.
- **Data locality.** With HDFS, computation happens on the DataNodes where the data resides, rather than having the data move to where the computational unit is. By minimizing the distance between the data and the computing process, this approach decreases network congestion and boosts a system's overall throughput.

What are the benefits of using HDFS?

There are five main advantages to using HDFS, including:

1. **Cost effectiveness.** The DataNodes that store the data rely on inexpensive off-the-shelf hardware, which cuts storage costs. Also, because HDFS is open source, there's no licensing fee.

2. **Large data set storage.** HDFS stores a variety of data of any size -- from megabytes to petabytes -- and in any format, including structured and unstructured data.
3. **Fast recovery from hardware failure.** HDFS is designed to detect faults and automatically recover on its own.
4. **Portability.** HDFS is portable across all hardware platforms, and it is compatible with several operating systems, including Windows, Linux and Mac OS/X.
5. **Streaming data access.** HDFS is built for high data throughput, which is best for access to streaming data.

HDFS use cases and examples

The Hadoop Distributed File System emerged at Yahoo as a part of that company's online ad placement and search engine requirements. Like other web-based companies, Yahoo juggled a variety of applications that were accessed by an increasing number of users, who were creating more and more data.

EBay, Facebook, LinkedIn and Twitter are among the companies that used HDFS to underpin big data analytics to address requirements similar to Yahoo's.

HDFS has found use beyond meeting ad serving and search engine requirements. The New York Times used it as part of large-scale image conversions, Media6Degrees for log processing and machine learning, LiveBet for log storage and odds analysis, Joost for session analysis, and Fox Audience Network for log analysis and data mining. HDFS is also at the core of many open source data lakes.

More generally, companies in several industries use HDFS to manage pools of big data, including:

- **Electric companies.** The power industry deploys phasor measurement units (PMUs) throughout their transmission networks to monitor the health of smart grids. These high-speed sensors measure current and voltage by amplitude and phase at selected transmission stations. These companies analyze PMU data to detect system faults in network segments and adjust the grid accordingly. For instance, they might switch to a backup power source or perform a load adjustment. PMU networks clock thousands of records per second, and consequently, power companies can benefit from inexpensive, highly available file systems, such as HDFS.

- **Marketing.** Targeted marketing campaigns depend on marketers knowing a lot about their target audiences. Marketers can get this information from several sources, including CRM systems, direct mail responses, point-of-sale systems, Facebook and Twitter. Because much of this data is unstructured, an HDFS cluster is the most cost-effective place to put data before analyzing it.
- **Oil and gas providers.** Oil and gas companies deal with a variety of data formats with very large data sets, including videos, 3D earth models and machine sensor data. An HDFS cluster can provide a suitable platform for the big data analytics that's needed.
- **Research.** Analyzing data is a key part of research, so, here again, HDFS clusters provide a cost-effective way to store, process and analyze large amounts of data.

HDFS data replication

Data replication is an important part of the HDFS format as it ensures data remains available if there's a node or hardware failure. As previously mentioned, the data is divided into blocks and replicated across numerous nodes in the cluster. Therefore, when one node goes down, the user can access the data that was on that node from other machines. HDFS maintains the replication process at regular intervals.

Analysis of data with Hadoop

Hadoop is designed to analyze large volumes of data in a distributed and parallelized manner. The key to this approach is dividing the data into smaller chunks and processing those chunks in parallel across multiple nodes in the Hadoop cluster. Here's how data is analyzed in Hadoop internally:

Data ingestion: The first step is to ingest the data into the Hadoop cluster. The data is typically stored in HDFS, which is a distributed file system that can handle large volumes of data. The data is divided into blocks of a fixed size, typically 64MB, and each block is stored on a separate node in the cluster.

Map phase: Once the data is ingested, Hadoop processes it in parallel across multiple nodes using a MapReduce job. In the Map phase, the data is processed in parallel across the nodes, with each node processing a subset of the data. The Map phase takes the input data, applies a transformation to it, and outputs a set of key-value pairs.

Shuffle and sort: After the Map phase is complete, the key-value pairs are shuffled and sorted by key across the nodes. This ensures that all key-value pairs with the same key are grouped together and sent to the same node for processing in the Reduce phase.

Reduce phase: In the Reduce phase, the key-value pairs are processed in parallel across the nodes, with each node processing a subset of the data. The Reduce phase takes the key-value pairs as input, applies an aggregation function to them, and outputs the final result.

Output: The final output of the MapReduce job is typically stored in HDFS, where it can be accessed and analyzed by other tools and frameworks, such as Apache Pig or Apache Hive.

This approach allows Hadoop to analyze large volumes of data in a scalable and distributed manner. By dividing the data into smaller chunks and processing those chunks in parallel across multiple nodes, Hadoop can process large volumes of data much more quickly and efficiently than traditional data processing frameworks.

Hadoop Streaming

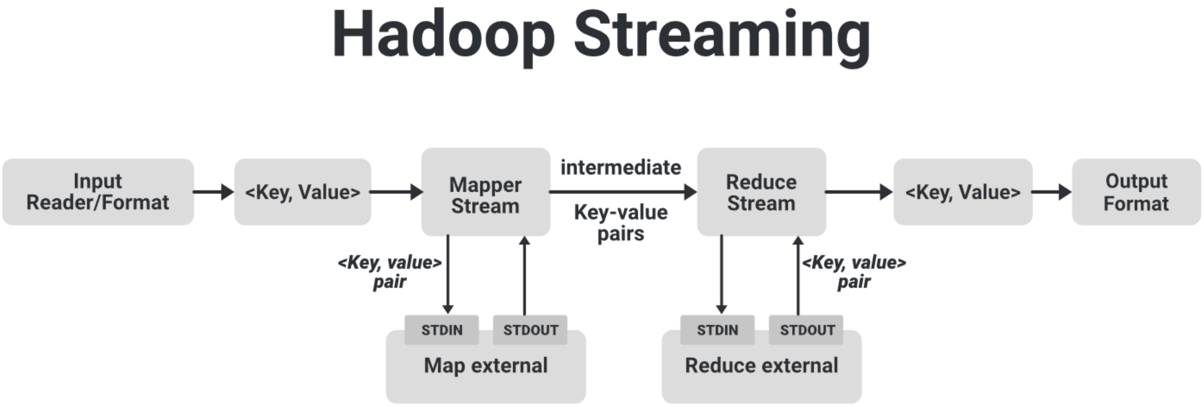
Hadoop Streaming is a utility that allows you to use any programming language, such as Python or Ruby, to write MapReduce jobs for Hadoop. It provides a way for you to write MapReduce jobs without having to write them in Java, the native language of Hadoop.

Hadoop Streaming works by using standard input and output streams to communicate with the MapReduce jobs. The input data is sent to the Map function through standard input, and the output data is sent from the Map function to the Reduce function through standard output.

What is Hadoop Streaming?

It is a utility or feature that comes with a Hadoop distribution that allows developers or programmers to write the Map-Reduce program using different programming languages like Ruby, Perl, Python, C++, etc. We can use any language that can read from the standard input(STDIN) like keyboard input and all and write using standard output(STDOUT). We all know the Hadoop Framework is completely written in java but programs for Hadoop are not necessarily need to code in Java programming language. feature of Hadoop Streaming is available since Hadoop version 0.14.1.

How Hadoop Streaming Works



In the above example image, we can see that the flow shown in a dotted block is a basic MapReduce job. In that, we have an Input Reader which is responsible for reading the input data and produces the list of key-value pairs. We can read data in .csv format, in delimiter format, from a database table, image data(.jpg, .png), audio data etc. The only requirement to read all these types of data is that we have to create a particular input format for that data with these input readers. The input reader contains the complete logic about the data it is reading. Suppose we want to read an image then we have to specify the logic in the input reader so that it can read that image data and finally it will generate key-value pairs for that image data.

If we are reading an image data then we can generate key-value pair for each pixel where the key will be the location of the pixel and the value will be its color value from (0-255) for a colored image. Now this list of key-value pairs is fed to the Map phase and Mapper will work on each of these key-value pair of each pixel and generate some intermediate key-value pairs which are then fed to the Reducer after doing shuffling and sorting then the final output produced by the reducer will be written to the HDFS. These are how a simple Map-Reduce job works.

Now let's see how we can use different languages like Python, C++, Ruby with Hadoop for execution. We can run this arbitrary language by running them as a separate process. For that, we will create our external mapper and run it as an external separate process. These external map processes are not part of the basic MapReduce flow. This external mapper will take input from STDIN and produce output to STDOUT. As the key-value pairs are passed to the internal mapper the internal mapper process will send these key-value pairs to the external mapper where we have written our code in some other language like with python with help of STDIN. Now, these external mappers process these key-value pairs and generate intermediate key-value pairs with help of STDOUT and send it to the internal mappers.

Similarly, Reducer does the same thing. Once the intermediate key-value pairs are processed through the shuffle and sorting process they are fed to the internal reducer which will send these pairs to external reducer process that are working separately through the help of STDIN and gathers the output generated by external reducers with help of STDOUT and finally the output is stored to our HDFS.

This is how Hadoop Streaming works on Hadoop which is by default available in Hadoop. We are just utilizing this feature by making our external mapper and reducers. Now we can see how powerful feature is Hadoop streaming. Anyone can write his code in any language of his own choice.

Scaling

It can be defined as a process to expand the existing configuration (servers/computers) to handle a large number of user requests or to manage the amount of load on the server. This process is called scalability. This can be done either by increasing the current system configuration (increasing RAM, number of servers) or adding more power to the configuration. Scalability plays a vital role in the designing of a system as it helps in responding to a large number of user requests more effectively and quickly.
Hadoop is designed to scale horizontally, meaning that you can add more nodes to a Hadoop cluster to increase its processing power and storage capacity. Scaling Hadoop involves adding more nodes to the cluster and configuring the Hadoop services to use the new nodes. There are two ways to do this

1. **Vertical Scaling**
2. **Horizontal Scaling**

Vertical Scaling

It is defined as the process of increasing the capacity of a single machine by adding more resources such as memory, storage, etc. to increase the throughput of the system. No new resource is added, rather the capability of the existing resources is made more efficient. This is called **Vertical scaling**. Vertical Scaling is also called the Scale-up approach.
Example: **MySQL**

Advantages of Vertical Scaling

1. It is easy to implement
2. Reduced software costs as no new resources are added
3. Fewer efforts required to maintain this single system

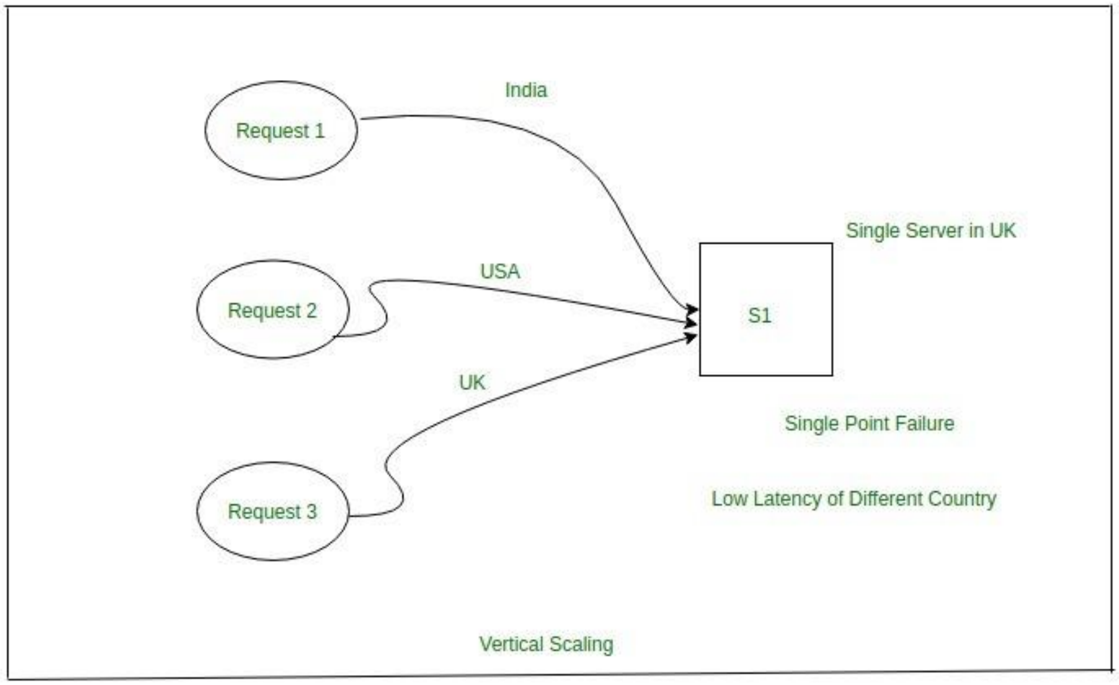
Disadvantages of Vertical Scaling

1. Single-point failure

2. Since when the system (server) fails, the downtime is high because we only have a single server
3. High risk of hardware failures

A Real-time Example of Vertical Scaling

When traffic increases, the server degrades in performance. The first possible solution that everyone has is to increase the power of their system. For instance, if earlier they used 8 GB RAM and 128 GB hard drive now with increasing traffic, the power of the system is affected. So a possible solution is to increase the existing RAM or hard drive storage, for e.g. the resources could be increased to 16 GB of RAM and 500 GB of a hard drive but this is not an ultimate solution as after a point of time, these capacities will reach a saturation point.



Horizontal Scaling

It is defined as the process of adding more instances of the same type to the existing pool of resources and not increasing the capacity of existing resources like in vertical scaling. This kind of scaling also helps in decreasing the load on

the server. This is called horizontal scaling. Horizontal Scaling is also called the Scale-out approach.
In this process, the number of servers is increased and not the individual capacity of the server. This is done with the help of a **Load Balancer** which basically routes the user requests to different servers according to the availability of the server. Thereby, increasing the overall performance of the system. In this way, the entire process is distributed among all servers rather than just depending on a single server. Example: NoSQL, Cassandra and MongoDB

Advantages of Horizontal Scaling

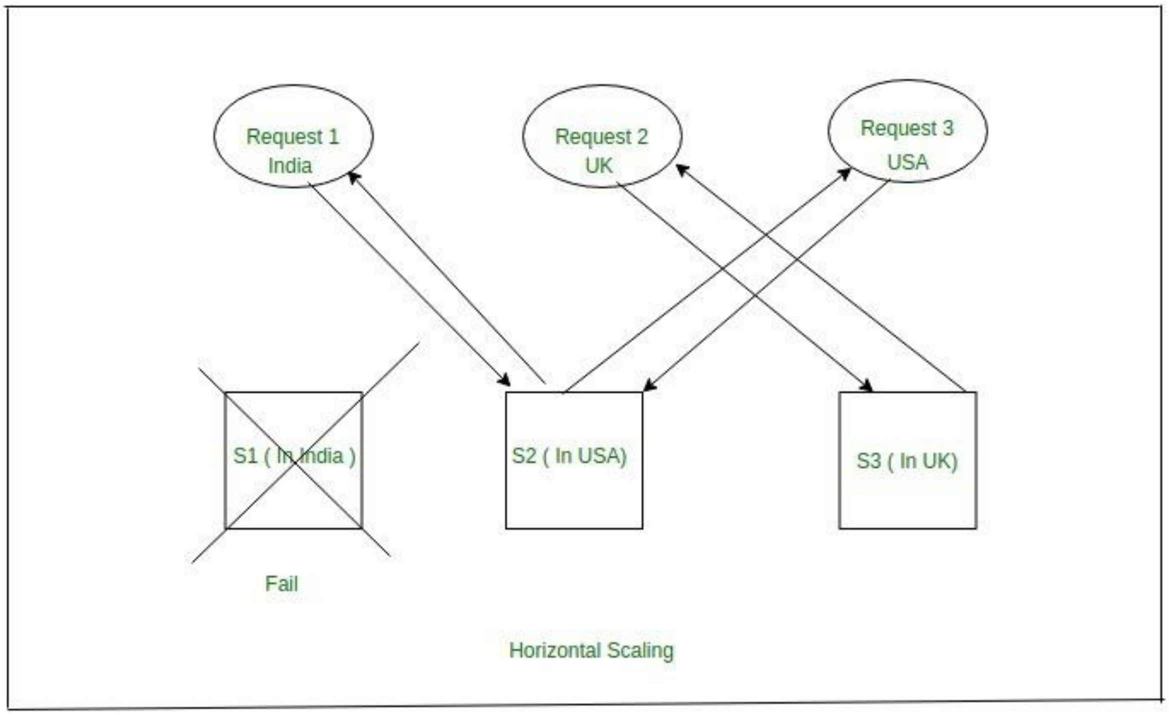
1. **Fault Tolerance** means that there is no single point of failure in this kind of scale because there are 5 servers here instead of 1 powerful server. So if anyone of the servers fails then there will be other servers for backup. Whereas, in **Vertical Scaling** there is single point failure i.e: if a server fails then the whole service is stopped.
2. **Low Latency:** Latency refers to how late or delayed our request is being processed.
3. Built-in backup

Disadvantages of Horizontal Scaling

1. Not easy to implement as there are a number of components in this kind of scale
2. Cost is high
3. Networking components like, router, load balancer are required

A Real-time Example of Horizontal Scaling

For example, if there exists a system of the capacity of 8 GB of RAM and in future, there is a requirement of 16 GB of RAM then, rather than the increasing capacity of 8 GB RAM to 16 GB of RAM, similar instances of 8 GB RAM could be used to meet the requirements.



Steps involved in scaling a Hadoop cluster:

Add more nodes to the cluster: To scale a Hadoop cluster, you need to add more nodes to it. The new nodes should be similar in hardware and software configuration to the existing nodes in the cluster.

Configure the new nodes: Once you have added the new nodes to the cluster, you need to configure them to work with the existing nodes. This involves installing the Hadoop software on the new nodes and configuring them to use the same settings as the existing nodes, such as the same HDFS and MapReduce configurations.

Update the Hadoop services: After you have added and configured the new nodes, you need to update the Hadoop services to use the new nodes. This involves updating the Hadoop configuration files to include the new nodes and restarting the Hadoop services to apply the changes.

Test the cluster: After you have updated the Hadoop services, you should test the cluster to ensure that it is working as expected. You can do this by running some test MapReduce jobs and verifying that the output is correct.

Monitor the cluster: Once you have scaled the cluster, you should monitor it to ensure that it is running smoothly. This involves monitoring the resource usage of the nodes, the health of the Hadoop services, and the performance of the cluster.

Scaling a Hadoop cluster allows you to process larger volumes of data and handle more concurrent users. However, it also requires careful planning and management to ensure that the cluster remains stable and performs well.

CLUSTERING

Clustering is a technique used in data analysis to group similar data points together. In the context of big data, clustering can be used to analyze large datasets and discover patterns and relationships between data points. Clustering can be performed using various algorithms,

1.Single node clustering

Single node clustering, also known as standalone mode, refers to running Hadoop on a single machine. This mode is typically used for development and testing purposes, as well as for small-scale data processing tasks. In this mode, all the Hadoop daemons (NameNode, DataNode, JobTracker, TaskTracker) run on a single machine, and the input and output data are stored on the local file system.

Here are some of the steps involved in setting up a single node Hadoop cluster:

Install Hadoop: First, you need to install Hadoop on your machine. This involves downloading the Hadoop software from the Apache Hadoop website, and extracting the files to a directory on your machine.

Configure Hadoop: Once Hadoop is installed, you need to configure it to work in standalone mode. This involves editing the configuration files (core-site.xml, hdfs-site.xml, mapred-site.xml) to specify the location of the Hadoop daemons, as well as the input and output data directories.

Start Hadoop: Once Hadoop is configured, you can start the Hadoop daemons using the start-all.sh script. This will start the NameNode, DataNode, JobTracker, and TaskTracker daemons on your machine.

Test Hadoop: Once Hadoop is started, you can test it by running a sample MapReduce job. Hadoop provides several sample MapReduce jobs (e.g., WordCount) that you can run to verify that Hadoop is working correctly.

Some of the limitations of using a single node Hadoop cluster include:

Limited processing power: Since all the Hadoop daemons run on a single machine, the processing power is limited to the resources of that machine. This means that the processing speed for large datasets may be slow.

Limited storage capacity: Since the input and output data are stored on the local file system, the storage capacity is limited to the capacity of the machine's hard drive. This means that the amount of data that can be processed is limited.

No fault-tolerance: Since there is only one machine in the cluster, there is no fault-tolerance mechanism to handle hardware or software failures. If the machine fails, the entire cluster will be unavailable until the machine is repaired.

single node clustering is suitable for small-scale data processing tasks, but has limitations in terms of processing power, storage capacity, and fault-tolerance.

2. Multinode clustering

Multi-node clustering, on the other hand, refers to running Hadoop on a cluster of machines. This is used for large-scale data processing tasks, where the data is too large to be processed on a single machine. In a multi-node cluster, the Hadoop daemons are distributed across multiple machines, with each machine running one or more daemons. The input and output data are stored on the Hadoop Distributed

File System (HDFS), which is a distributed file system that spans all the machines in the cluster.

Setting up a multi-node Hadoop cluster involves several steps, including:

Planning the cluster: This involves deciding on the number and type of machines to be used, as well as the configuration of the Hadoop daemons and the HDFS.

Installing Hadoop: This involves installing the Hadoop software on each machine in the cluster, as well as configuring the Hadoop daemons and the HDFS.

Configuring the Hadoop cluster: This involves configuring the Hadoop daemons and the HDFS to work together, as well as setting up the network and security settings for the cluster.

Testing the cluster: This involves testing the cluster to ensure that it is working as expected, including running sample MapReduce jobs and verifying that the output is correct.

Some of the benefits of using a multi-node Hadoop cluster include:

Scalability: A multi-node cluster can be easily scaled by adding more machines to the cluster, allowing for the processing of larger datasets.

Fault-tolerance: Hadoop provides built-in fault-tolerance mechanisms, such as data replication and automatic failover, which help to ensure that the cluster remains operational even in the event of hardware or software failures.

Distributed processing: The distributed nature of Hadoop allows for parallel processing of data, which can significantly improve the processing speed for large datasets.

In summary, while single node clustering is suitable for small-scale data processing tasks, multi-node clustering is required for large-scale data processing tasks, and provides scalability, fault-tolerance, and distributed processing capabilities.

Working with Hadoop commands

Working with Hadoop commands is an important aspect of using Hadoop for data processing and analysis. Hadoop provides a number of command-line tools for managing and working with Hadoop clusters, including the Hadoop Distributed File System (HDFS) and MapReduce.

Here are some commonly used Hadoop commands and their functions:

hdfs dfs: This command is used to interact with the Hadoop Distributed File System (HDFS). Some commonly used options include:
ls: list the contents of a directory in HDFS
mkdir: create a new directory in HDFS
rm: remove a file or directory from HDFS
put: copy a file from the local file system to HDFS
get: copy a file from HDFS to the local file system
For example, to list the contents of the root directory in HDFS, you can use the following command:

```
hdfs dfs -ls /
```

yarn: This command is used to interact with the YARN (Yet Another Resource Negotiator) resource management layer in Hadoop. Some commonly used options include:
application -list: list the currently running YARN applications
application -kill: kill a running YARN application
node -list: list the nodes in the Hadoop cluster
For example, to list the nodes in the Hadoop cluster, you can use the following command:

```
yarn node -list
```

mapred: This command is used to interact with the MapReduce processing layer in Hadoop. Some commonly used options include:
job -list: list the currently running MapReduce jobs
job -kill: kill a running MapReduce job

job -history: view the history of completed MapReduce jobs

For example, to view the history of completed MapReduce jobs, you can use the following command:

```
mapred job -history <job_id>
```

hadoop fs: This command is used to interact with the Hadoop file system, and is similar to the hdfs dfs command. Some commonly used options include:

ls: list the contents of a directory in HDFS

mkdir: create a new directory in HDFS

rm: remove a file or directory from HDFS

put: copy a file from the local file system to HDFS

get: copy a file from HDFS to the local file system

For example, to list the contents of the root directory in HDFS, you can use the following command:

```
hadoop fs -ls /
```

These are just a few examples of the Hadoop commands that can be used to manage and work with Hadoop clusters. By using these commands, you can perform a wide range of tasks, including uploading and downloading data to HDFS, submitting and managing MapReduce jobs, and monitoring the status of the Hadoop cluster.

Working with Apache oozie

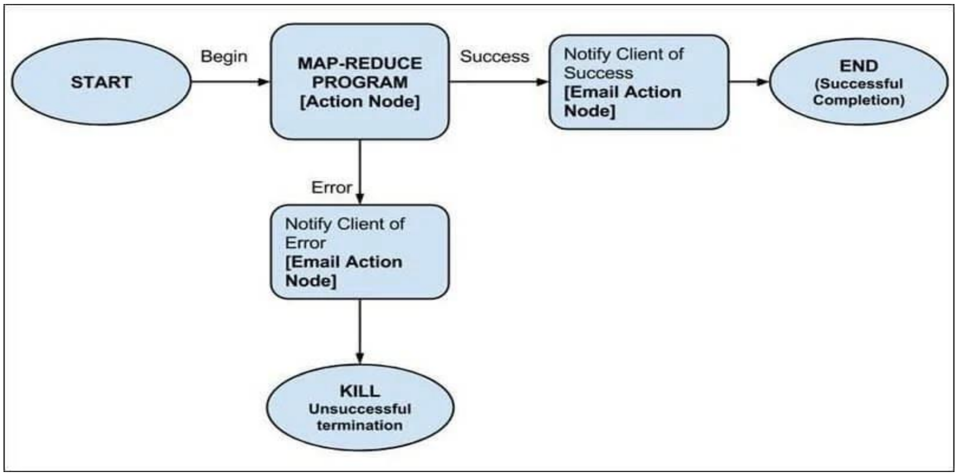
Apache Oozie is a workflow scheduler for Hadoop. It is a system which runs the workflow of dependent jobs. Here, users are permitted to create Directed Acyclic Graphs of workflows, which can be run in parallel and sequentially in Hadoop.

It consists of two parts:

Workflow engine: Responsibility of a workflow engine is to store and run workflows composed of Hadoop jobs e.g., MapReduce, Pig, Hive.

Coordinator engine: It runs workflow jobs based on predefined schedules and availability of data.

Oozie is scalable and can manage the timely execution of thousands of workflows (each consisting of dozens of jobs) in a Hadoop cluster.


START
Begin
MAP-REDUCE PROGRAM [Action Node]
Success
Notify Client of Success [Email Action Node]
END (Successful Completion)
Error
Notify Client of Error [Email Action Node]
KILL Unsuccessful termination

Working of oozie

Oozie runs as a service in the cluster and clients submit workflow definitions for immediate or later processing.

Oozie workflow consists of action nodes and control-flow nodes.

An action node represents a workflow task, e.g., moving files into HDFS, running a MapReduce, Pig or Hive jobs, importing data using Sqoop or running a shell script of a program written in Java.

A control-flow node controls the workflow execution between actions by allowing constructs like conditional logic wherein different branches may be followed depending on the result of earlier action node.

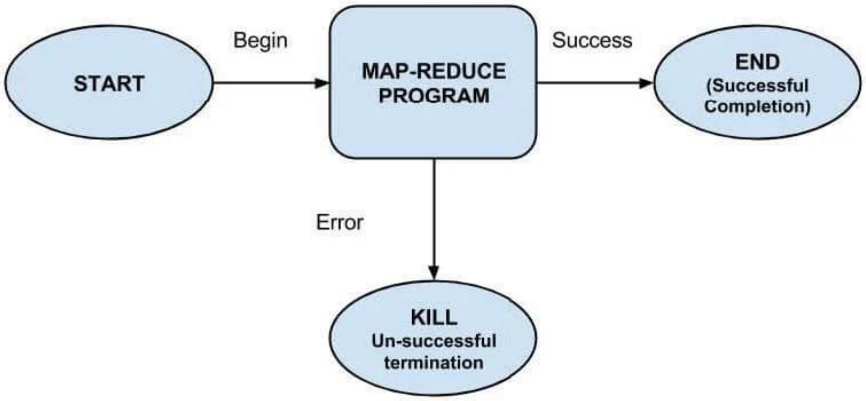
Start Node, End Node, and Error Node fall under this category of nodes.

Start Node, designates the start of the workflow job.

End Node, signals end of the job.

Error Node designates the occurrence of an error and corresponding error message to be printed.

At the end of execution of a workflow, HTTP callback is used by Oozie to update the client with the workflow status. Entry-to or exit from an action node may also trigger the callback.



Features of Oozie

Oozie has client API and command line interface which can be used to launch, control and monitor job from Java application.

Using its Web Service APIs one can control jobs from anywhere.

Oozie has provision to execute jobs which are scheduled to run periodically.

Oozie has provision to send email notifications upon completion of jobs.