

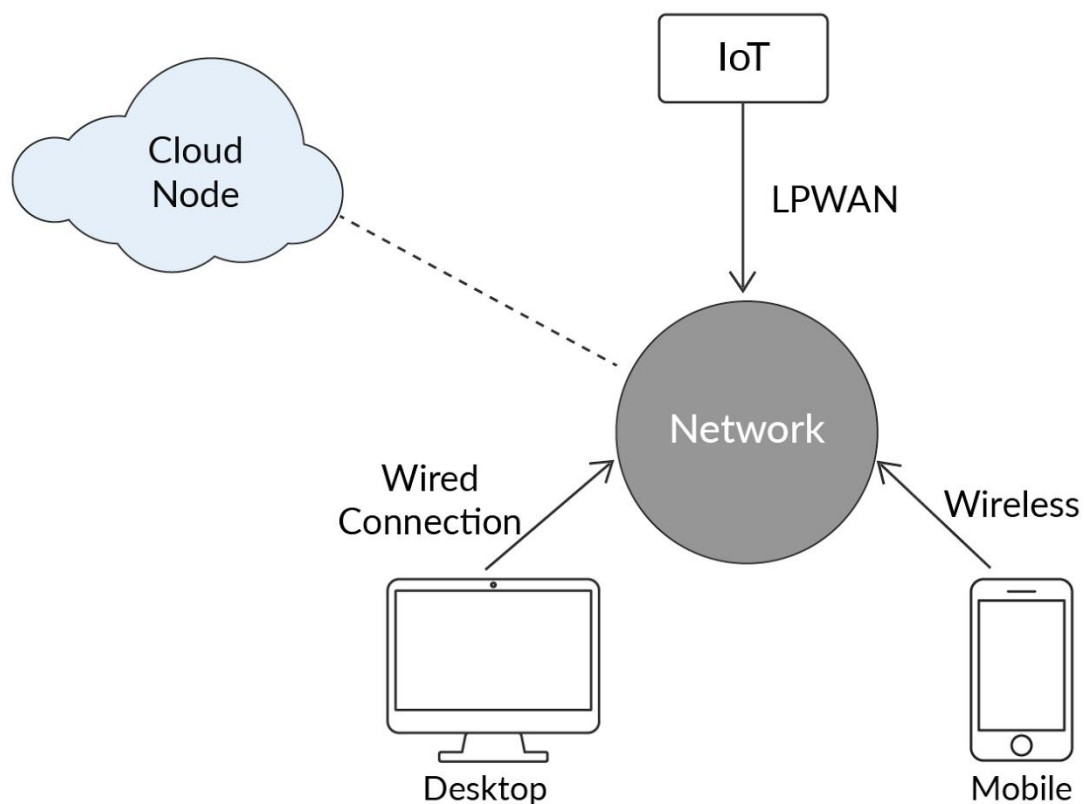
Summary

Hadoop Distributed File System

In this session, you were introduced to the Hadoop Distributed File System (HDFS). You began the session by understanding the concept of distributed systems. Then you learnt about the CAP theorem and its relation with distributed systems. Next, you learnt about the working of the HDFS, its features and its architecture in detail. Finally, you learnt how to set up and work with an on-premise Hadoop cluster.

Understanding a Distributed System

A distributed system is composed of many independent computers or devices that are connected to each other over a network and run a set of services, thus making the whole system appear as a single machine. These devices are referred to as nodes, and they communicate and coordinate their actions by sending and receiving messages among each other to solve tasks.



Fault Tolerance

The two major types of system failure are as follows:

1. **Node failure:** A node failure refers to a situation where there are issues in the nodes of the distributed system. It includes failures such as fail-stop failure and Byzantine failure.
 - a. A **Byzantine failure** is said to occur when a node misbehaves by sending wrong messages to other nodes in the network.
 - b. A **fail-stop failure** is said to occur when a distributed system fails to work when one of the nodes fails or crashes.
2. **Network failures:** These are caused due to issues with the connections among the different nodes in the system.

When any of the failures listed above occurs, the data stored in the node becomes unavailable for the rest of the system. This is where the concept of replication comes into the picture.

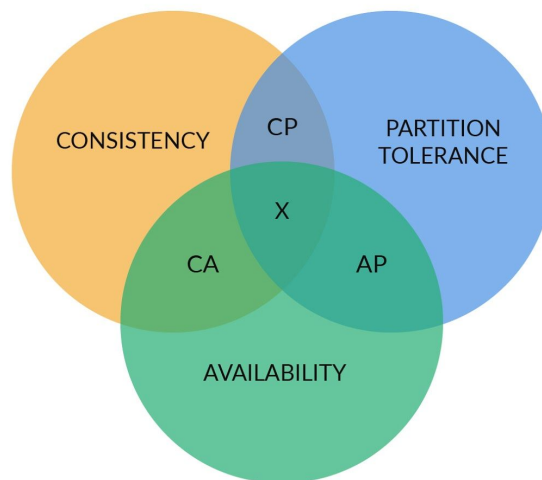
Properties of a distributed system:

1. **High availability:** A distributed system should ensure availability, i.e., even if one of the system's components fails, the system itself should not.
2. **Scalability:** A distributed system should not fail when the workload increases. It should be flexible so that you can easily increase the number of machines.
3. **Transparency:** A distributed system should provide a global view of all the underlying machines as a single machine and hide the internal workings of the system from the end users.

CAP Theorem

The CAP theorem states the main challenges that you might encounter while designing a distributed system. These challenges can be represented in terms of three main dimensions, namely, **Consistency**, **Availability** and **Partition tolerance**.

1. When you replicate this data for fault tolerance, it is essential to achieve **consistency** among all the replicas.
2. **Availability** states that the system should be available even if some of the nodes on which the data is replicated are down.
3. A distributed system is said to be **partition tolerant** if it doesn't fail in case of failure of any partition or node.



It is possible to achieve only two of the three characteristics above while designing distributed systems. Hence, distributed systems are designed to provide partition tolerance and availability, with consistency achieved eventually.

Hadoop Distributed File System

A filesystem can be defined as an abstraction over a storage medium, and it provides an interface to perform various operations on a file, such as create, open, read, update and append. In Linux, files are divided into blocks 4 KB in size. Each block of a file is stored at a random location on the hard disk. In order to keep track of all 256 blocks of the file, the mapping between each block and the memory location is stored/saved as metadata. One of the primary reasons for breaking a file into blocks is that it reduces the I/O overhead, i.e., if a user wants to write at a specific location in a file, then they can fetch only the block containing that specific location, instead of retrieving the entire file.

Earlier, there were two popular file systems:

1. Network File System

In the case of a **Local Area Network (LAN)** or a **Wide Area Network (WAN)**, files are stored in servers and are made available to the client wherever the user logs in a transparent manner.

A Network File System (NFS) is designed in such a way that small pieces of user files can be stored in servers and a small chunk of a file (usually a few KB) is made available to the client where the user has logged in. The cached chunk of data at the client is invalidated (about 30 ms) and the client again fetches the data from the server to keep the cached chunk of data up to date with the updates made on the server. This protocol allows the NFS system to scale to only a few tens of nodes (~100 nodes).

2. Andrew File System

The Andrew File System protocol allows the client to download the full file from the server and leave a notify lock on the file at the server. The server takes the responsibility of invalidating and updating the client copy of the file when it changes. This protocol allows file-sharing to scale to 1,000 nodes.

Now, suppose the size of a file is greater than the storage space available in a single machine. There are two ways to tackle this problem:

1. **Vertical scaling**, where you upgrade the existing setup, or
2. **Horizontal scaling**, where you provide a unified view of all the files and directories even if they are stored on different systems.

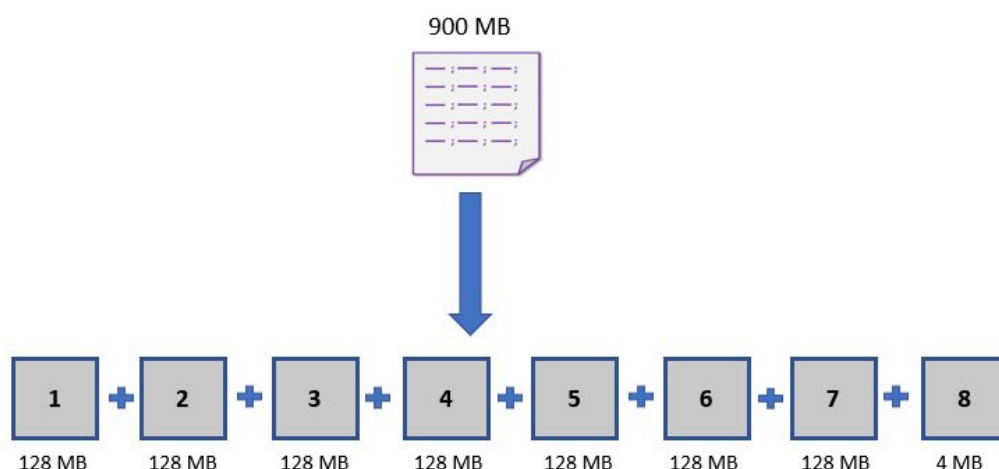
Vertical scaling is a costly process, and it is much easier to get access to cheap hardware; hence, horizontal scaling is preferred.

Hadoop was designed as an open-source project under the Apache Software Foundation by two Yahoo employees. This framework has the following components:

1. **Hadoop Distributed File System (HDFS)**: It is a distributed file-system that stores data on commodity machines.
2. **MapReduce**: It is a programming model for large-scale data processing.
3. **YARN**: It is a resource management platform that is responsible for managing compute resources in a cluster of machines and using them to schedule jobs.

Storage in HDFS

Similar to Linux, the default block sizes in the HDFS are 128 MB and 64 MB in Hadoop 2.x and Hadoop 1.x, respectively.



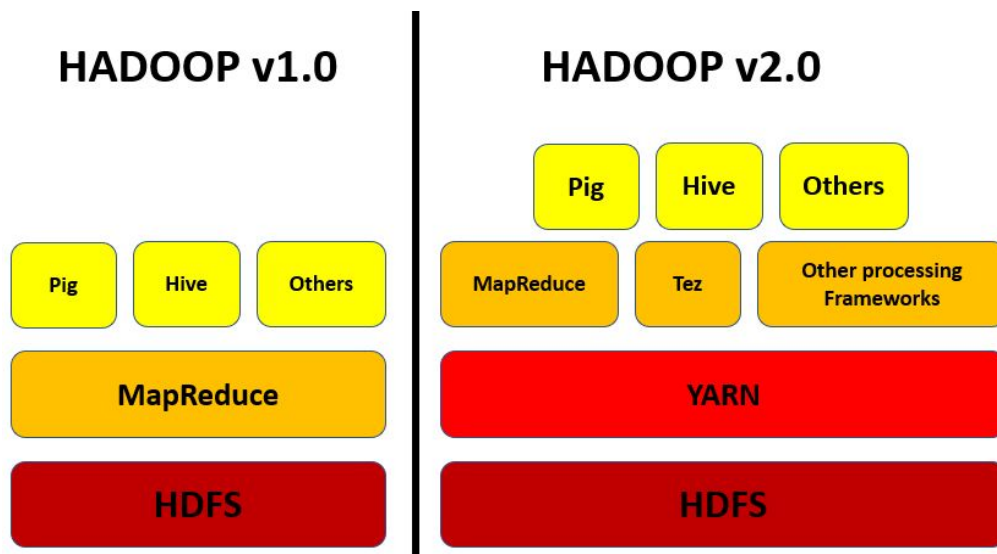
Having a large block size (128 MB) has the following advantages:

- **Reduces the number of I/O reads:** The client application can read the entire 128 MB in one go, thereby reducing the number of TCP/IP connections required to read an entire file.
- **Reduces the size of the metadata:** The HDFS has to maintain the metadata for each block of a file. Thus, if the number of blocks generated from a file is few, then the metadata stored to manage these blocks would also be less.

Limitations of having a large block size (128 MB) are as follows:

- **Reduces the throughput:** The larger the block size is, the fewer is the number of blocks. Therefore, the number of parallel tasks reading the blocks also is reduced.

Given below is a pictorial representation of the difference between Hadoop 1.x and Hadoop 2.x

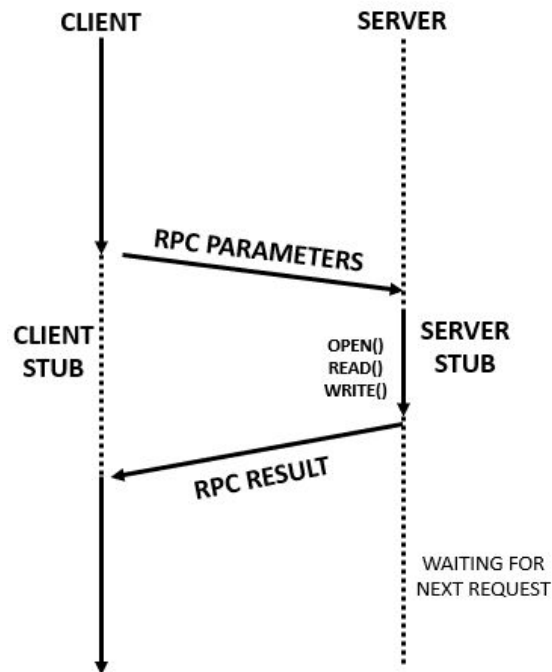


Remote Procedure Call

A Remote Procedure Call (RPC) is an interprocess communication technique that is used by distributed, client-server-based applications as a communication protocol. An RPC is also referred to as a function call or a subroutine call. The list below shows the sequence of events that occur when a client server wants to make an RPC request:

1. The client procedure calls the client stub procedure (method/function) and passes the parameters to be involved in the RPC.
2. The client stub procedure creates a message for the RPC.
3. The client OS then sends the message to the remote server.
4. The remote server OS receives the messages and passes them to the remote server stub.
5. The server stub unpacks the messages and calls the server procedure requested in the message with the parameters provided by the client server.
6. The server procedure executes and returns the results to the remote server stub.
7. The remote server stub packs the returned value as a message.
8. Once the reply message is ready, the remote server OS sends it to the client OS.

9. The client OS receives the message and assigns it to the client stub.
10. The client stub unpacks the message, and the execution returns to the client procedure.

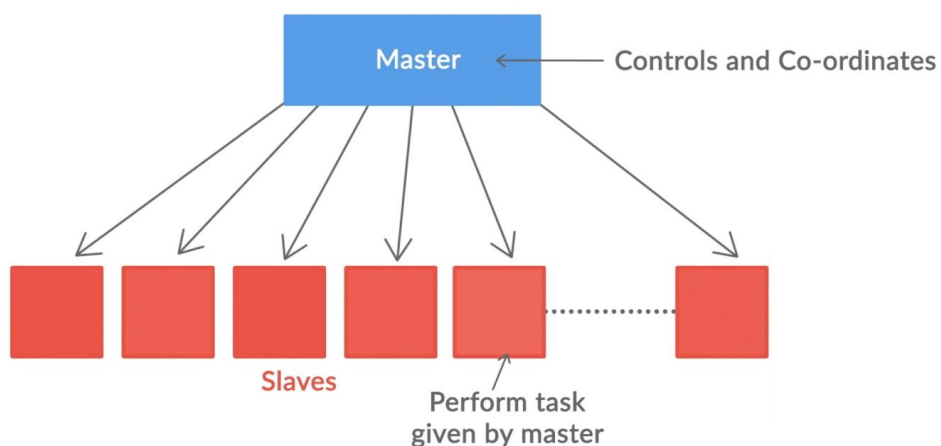


All internal messages exchanged between the procedures/servers in the RPC are kept hidden from the user.

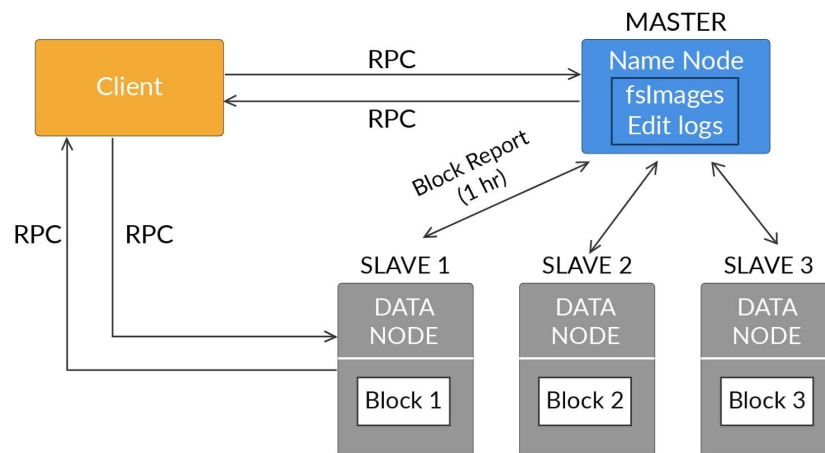
HDFS Architecture

The master node in a cluster acts as the master, which controls and coordinates tasks, whereas the slave nodes are the nodes that perform the tasks assigned by the master node.

MASTER - SLAVE LAYOUT



When a file is stored in the HDFS, the data blocks are stored in the DataNodes and the NameNode maintains information regarding the blocks' locations. When a read/write request is made to an existing file in the HDFS, first the client application interacts with the NameNode to get the metadata of all necessary blocks, and then, based on the information received from the NameNode, a connection is established with the respective DataNodes.



NameNode

It stores the metadata, which includes details of data blocks, replicas and so on. The metadata consists of two files:

- **FsImage**: This file contains complete information about file-to-block mapping, directory hierarchy and so on. It basically contains the complete state of the system at any point in time.
- **Edit Logs**: This file contains all the changes/updates to the file system in correspondence to the latest FsImage. When the NameNode starts, the existing FsImage and Edit Logs on the disk are combined into a new FsImage file and loaded in memory. Further, fresh Edit logs are initiated.

The NameNode regulates the client's access to the files; manages the file system namespace; and executes file system operations such as naming, closing and opening of files and directories.

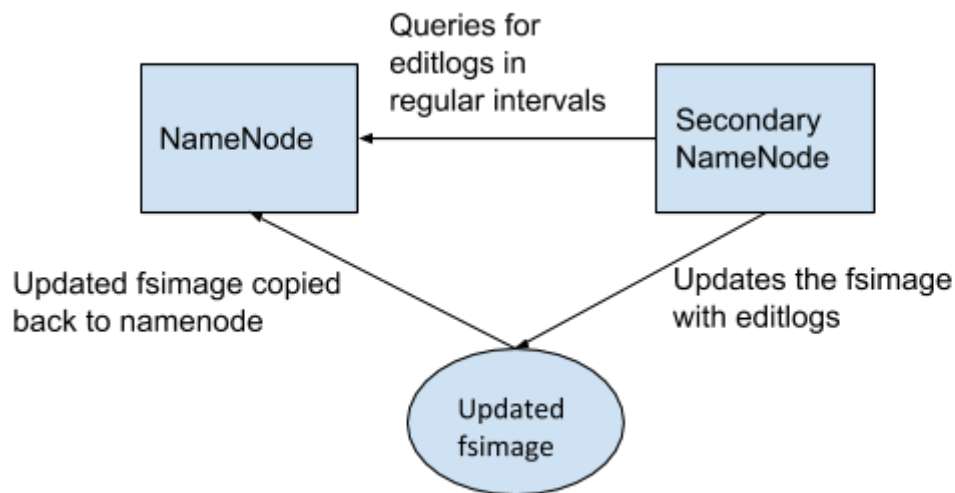
DataNodes

DataNodes have the following features:

- They store the entire data available on the HDFS.
- They create, replicate and delete blocks according to the instructions provided by the NameNode.
- They are provided with large amounts of hard disk space, as they actually store the data.
- They send block reports to the NameNode to indicate the blocks' states.

Secondary NameNode

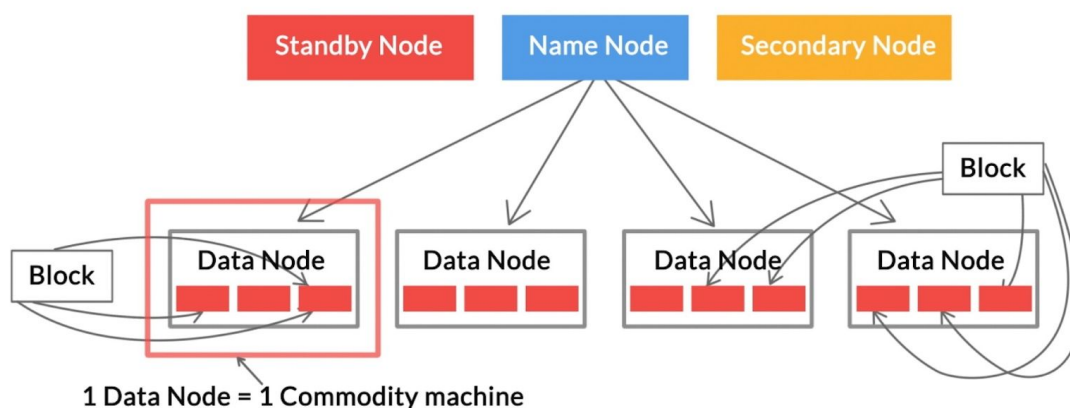
The secondary NameNode is just a helper node to the main node. Edit logs often have a large amount of data, and combining them with the FsImage takes a lot of time. The secondary NameNode acts as a checkpoint during this operation, and it reduces the time taken to combine these two files. It gets edit logs from the NameNode regularly, applies them to the FsImage and then moves the updated FsImage back to the NameNode. The secondary NameNode is also known as the checkpoint node.



Standby NameNode

Hadoop version 2 offers a high-availability feature with a standby NameNode, which stays in sync with the main NameNode. It is a backup for the main NameNode and takes over the responsibilities of the main NameNode in case it fails. In fact, a secondary NameNode is not required if a standby NameNode is in use. This feature was not available in Hadoop V1, and, hence, it was a single-point-of-failure (SPOF) system.

Hadoop Distributed File System (HDFS)



Replication and its Importance

The components of the HDFS are prone to failure because it uses commodity machines to store data. The only way to prevent data loss due to component failure is to replicate the data. Depending on the replication factor, the NameNode decides the DataNode addresses on which the blocks are to be replicated. By default, the HDFS keeps one original copy and two replicas of a data block. In total, there are three replicas for every block of a file in the HDFS. The default replication factor is, therefore, set to 3.

The DataNodes regularly send heartbeat messages and block reports to the NameNode to indicate that they are alive, and the NameNode acts in case of over-replication or under-replication of data blocks. If the NameNode receives no heartbeat message from a DataNode for a timeout period of 10 mins, then it assumes that the DataNode is no longer available, and based on the block information available, it replicates the blocks across other nodes to maintain appropriate replication level.

Note: For faster lookups, block reports are stored in the main memory, and FSImage files and EditLog files are stored on the disk as they are large in size.

On-Premise Hadoop Cluster

<code>hdfs fs -ls /</code>	This command is used to list the contents in the HDFS.
<code>hdfs fs -mkdir /input1</code>	This command is used to create a new directory in the present working directory.
<code>hdfs fs -put test.txt /test_data/</code>	This command is used to copy a file in the current working directory to a new directory named test_data.
<code>hdfs fs -help</code>	Using the help command, you can see all related HDFS commands.
<code>hdfs fs -chmod</code>	This command is used to change the permissions for a file. To do this, the user must be the owner of the file, or a super-user.
<code>hdfs fs -cp</code>	This command is used to copy files from the source to the destination.

hdfs fs -copyFromLocal	This command is used to copy a local file to Hadoop.
hdfs fs -count	This command is used to count the number of files, directories and bytes under a given path.
hdfs fs -rm	This command is used to delete directories and files.
hdfs fs -tail	This command is used to display the last kilobyte of data in a path.
hadoop fs -distcp	This command allows the user to perform inter-cluster/intra-cluster copying.

To learn about HDFS Shell Commands, you can refer to this [link](#).

To learn about Hadoop Distributed Copy, you can go [here](#).

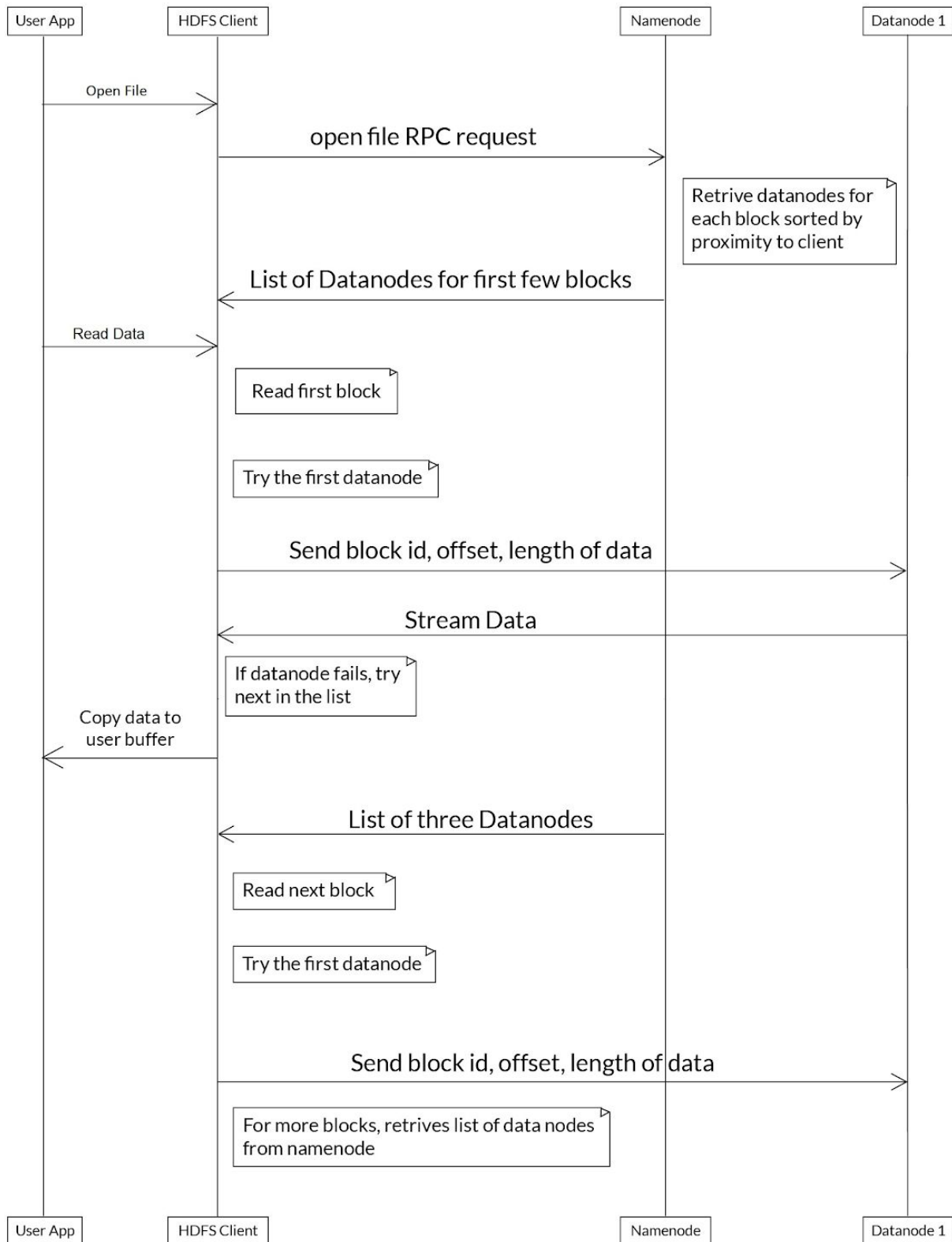
Read and Write Operations in the HDFS

Read Operation in the HDFS

Following are the steps involved in the file read process:

1. The user application provides a file name to be opened to the HDFS client.
2. The HDFS client sends an RPC request to the NameNode for reading the file.
3. The NameNode retrieves the list of DataNodes for the first few blocks of the file and sorts the list according to their proximity to the client.
4. For each block, the HDFS client connects to the first DataNode (the one closest to the user) in the sorted list and then sends the block ID, offset and length of the data to be read to the connected DataNode. The DataNode then streams the data back to the HDFS client directly.
5. The data packet is received by the HDFS client, which checks for information mismatch. If there is a mismatch, then it informs the NameNode and contacts the next DataNode in the list to read the block again.
6. Similarly, to retrieve the remaining blocks, the HDFS clients send RPC messages to the NameNode for fetching the addresses of the DataNodes storing them.

Open and Read File in HDFS

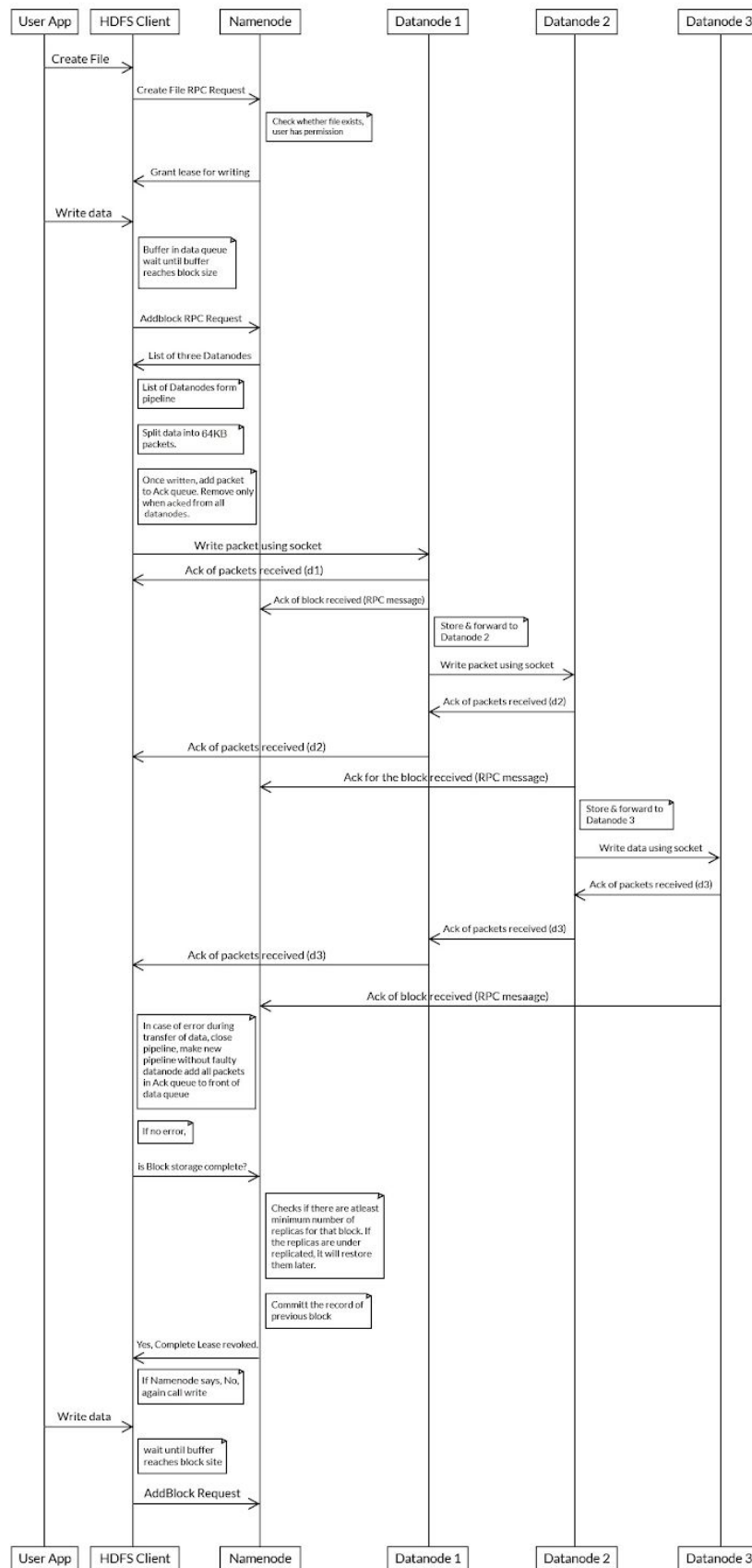


Write Operation in the HDFS

Following are the steps to write a data block in the HDFS:

1. The user application gives a filename as an input to the HDFS client. The HDFS client sends an RPC request to the NameNode for creating a file. After creating the file, the NameNode grants a data write lease to the HDFS client.
2. As the user application starts writing data, the HDFS client buffers the data until the size of a block is reached. After this, the HDFS client sends another RPC request to the namenode for adding a block.
3. The NameNode returns a list of the datanodes to which the block can be added. The number of DataNodes in the list is equal to the replication factor of the cluster. These DataNodes form a pipeline.
4. The HDFS client splits the data block into fixed-size packets and adds them to the Data queue.
5. The HDFS client starts writing the data packets to the first DataNode, which stores the data locally and then propagates the same data packet to the second DataNode. Similarly, the second DataNode stores the data packet locally and passes the packet to the third DataNode.
6. When the first DataNode receives the packet, it acknowledges the same to the HDFS client. Similarly, the other DataNodes acknowledge the same through the pipeline in the reverse direction. In the pipeline, the DataNodes can continue accepting new packets while sending acknowledgements for older packets.
7. As the HDFS client writes a packet, it is moved from the Data queue to another queue known as the Acknowledgment queue (the ack queue). When a packet is acknowledged by all the datanodes, it is removed from the Ack queue.
8. When all the packets are acknowledged, that is, the ack queue is empty, the HDFS client sends a 'write complete RPC' request to the Namenode.
9. On receiving the 'write complete RPC' request, the namenode checks whether the minimum number of replicas (default: 1) have been written. If yes, then it commits the block in the EditLogs and returns it successfully to the HDFS client.
10. If a DataNode in the pipeline fails, then the HDFS client does not receive an acknowledgement for any of the packets, that is, the ack queue is not empty. Then, the HDFS client transfers these packets from the Ack queue to the Data queue, after which it creates a new pipeline and ignores the failed datanodes. All the steps after step 5 are repeated for the remaining packets added to the data queue from the ACK queue. If the namenode daemon notices under-replication of the block, then it arranges for another DataNode where the block could be replicated.
11. Furthermore, the HDFS client assigns a new identity to the current block that is being written, and the same is communicated to the namenode. This is done so that the partially written blocks on the failed DataNode can be deleted when the DataNode recovers.
12. The DataNode periodically sends block reports to the namenode in order to inform it about newly written blocks. After receiving these block reports, the namenode updates its metadata accordingly.

Create and Write File in HDFS



Summary

MapReduce and YARN

In this session, you learnt about the two major components of Hadoop, MapReduce and YARN. You began the session by understanding how MapReduce works, and then you learnt how Middleware is involved in executing MapReduce jobs. After this, you learnt about YARN and, finally, you went through a practical demonstration of executing MapReduce jobs on a three-node Hadoop cluster.

MapReduce Programming

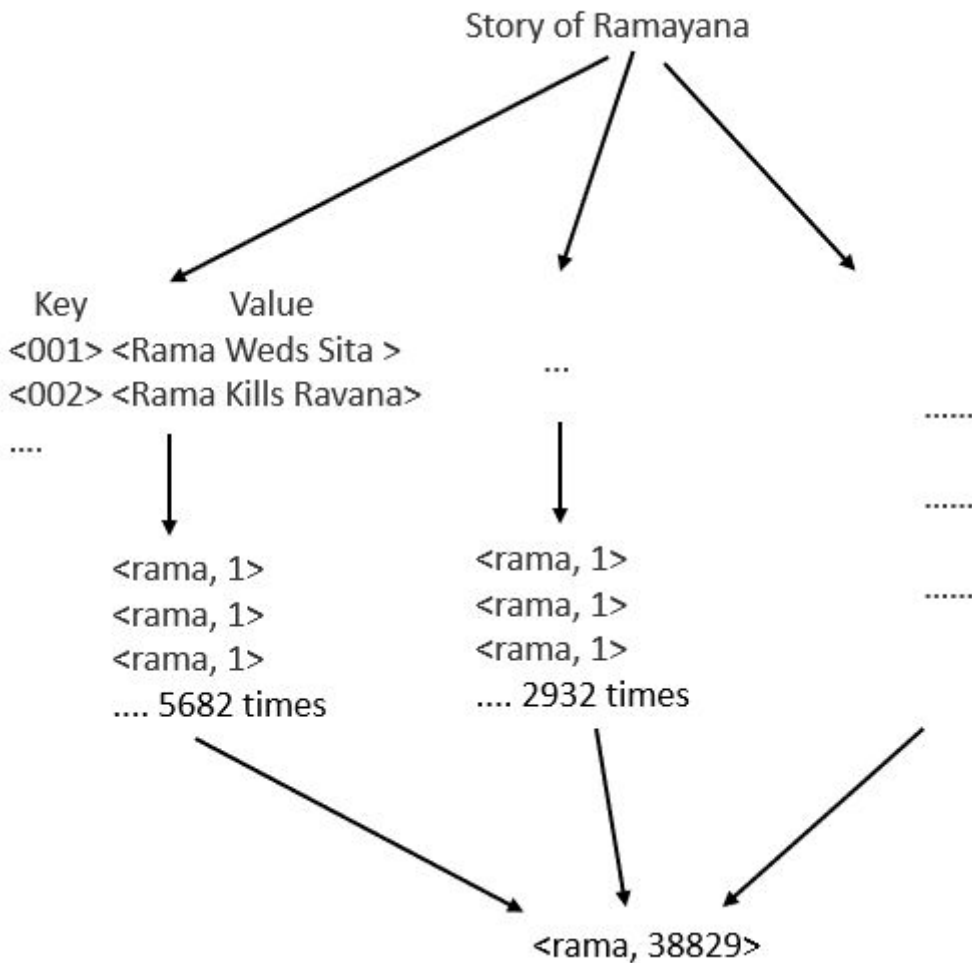
The Hadoop framework essentially has two components:

1. **Map task:** The map task takes a block of data and converts it into key-value pairs. The key and value are decided on the basis of the task that is to be performed.
2. **Reduce task:** The reduce task takes the output from the map as an input and combines those key-value pairs to get the desired output.

For communication to take place between the different machines in a distributed system, you need to represent each piece of data with a unique identity so that all the machines can identify a given piece of information with this unique identity.

Let us summarise the MapReduce paradigm as follows:

1. As you are aware, data will be in the form of blocks. As the mapper phase begins, the data is converted into key-value pairs.
2. Next, in the reduce phase, we take the output from all the mappers to perform a global aggregation of results.
3. The output of the final MapReduce program will be a key-value pair, where the key is the field that you want to keep unique.



Involvement of Middleware in Executing MapReduce Jobs

The job tracker and the task tracker are two components that help execute MapReduce programs.

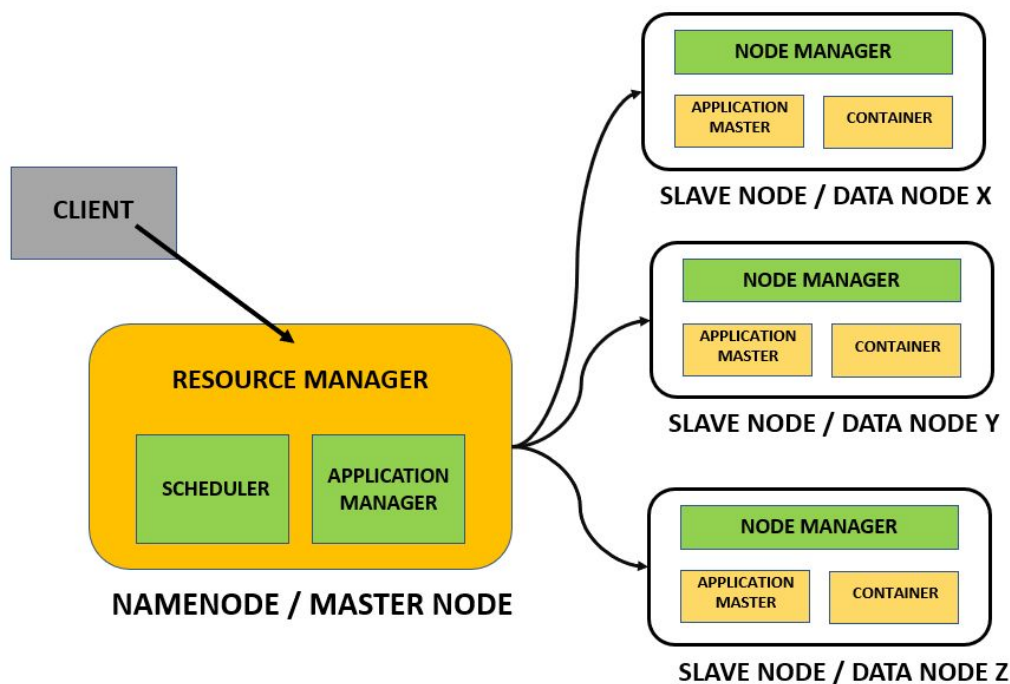
1. When a MapReduce job is launched, the job tracker consults the NameNode to obtain blocks of the input file from it. Next, the job tracker decides the nodes on which the map tasks can be launched and then informs the corresponding task trackers of those data nodes to start the task execution.
2. Similarly, the job tracker also decides the nodes on which reducer tasks are to be launched and then informs the corresponding task trackers of these nodes to launch the reducer tasks. Generally, it is possible to make the reducers start late, i.e., after the map tasks have produced the output.
3. The task tracker and the job tracker regularly exchange heartbeat messages, which are configurable, every second and the task tracker updates the job tracker about the progress made in executing the tasks.

During the process, the reducer task consults the job tracker to find out the map task nodes from which it should pull data. The map output is sorted and written onto different files for the corresponding reducers

to pull the data. After pulling the data from different map task nodes, the reducers perform a multi-way merge of files. The reducer then executes the task and writes its final output onto the HDFS file.

YARN

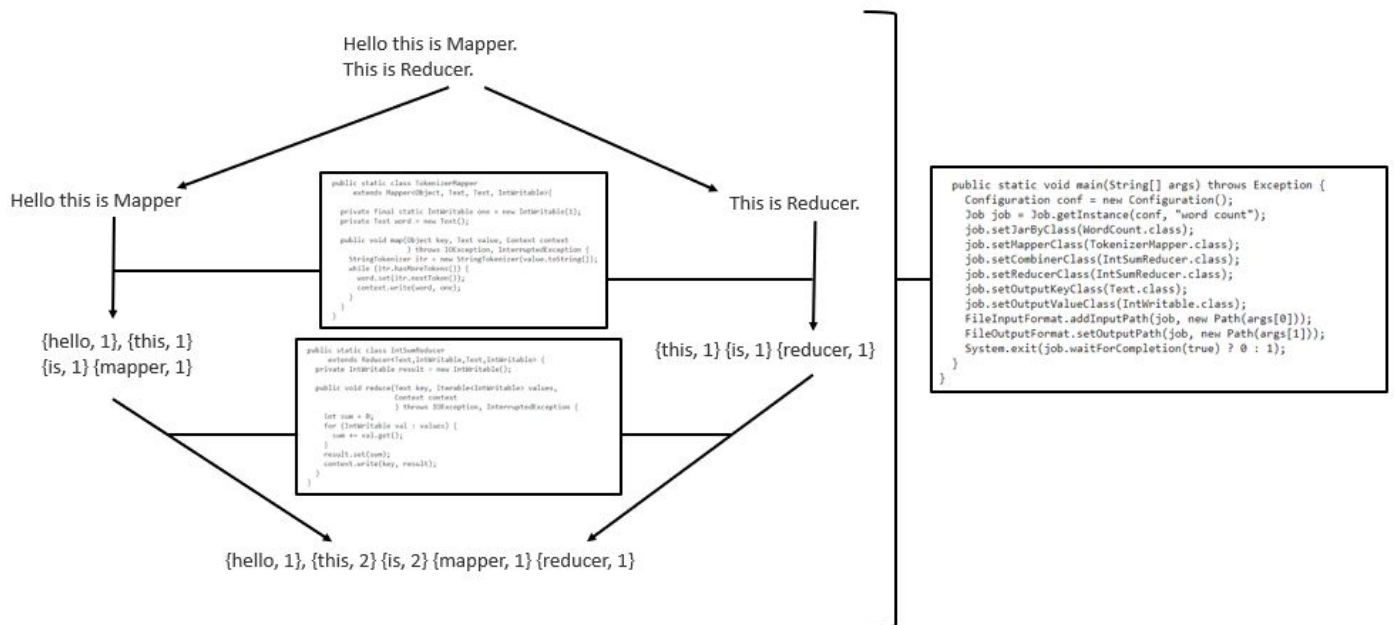
YARN stands for Yet Another Resource Negotiator. It has four major components, resource manager, node manager, application master and containers. The resource manager is located on a master machine, whereas the node manager, the application master and containers are located on the slave machines.



1. The client submits the job to the resource manager. The application manager present in the resource manager accepts the job. The application manager checks whether the task requested can be fulfilled with the resources available in the cluster. If not, then it rejects the task. In case the request is accepted, it moves ahead with the task of launching the application master for that particular task with the help of the scheduler.
2. It is always advised to keep the code file close to the data for faster processing. Hence, the scheduler contacts the name node to get the location of relevant data blocks. The name node replies with the locations of the data nodes where the data required by the code is stored.
3. The scheduler interacts with the respective node manager for available resources to launch an application master in the same node. If resources are not available in that node, then it will provide the application manager with another node to launch the application master.
4. The status of the task is reported by the application master. The main task of the application master is to negotiate resources (memory/CPU/network, etc.) with the resource manager and then execute the task assigned to it using those resources.
5. The resources provided to an application master to execute a task are known as containers. The application master is provided with these containers to run an application/task. There is exactly one node manager that manages all the resources in a node. The node manager reports the condition of the node, specifically, the CPU and RAM usage of that node, the node's health, etc, to the resource manager continuously.

6. Remember that a client submits a MapReduce job to the application manager. A MapReduce job can have one or more mapper and reducer codes. Let us assume that a MapReduce job has four sub-jobs, i.e., tasks. In this case, the node manager will launch exactly four application masters to process these codes, with each application master taking care of exactly one application. The application masters then negotiate for containers and execute the tasks allotted to them. After completion, the requested containers and the application masters free the occupied resources.

Running a MapReduce Program



Summary

EMR Cluster

In this session, you learnt how to work with the Hadoop framework on the Amazon EMR cluster.

Advantages of AWS over On-Premise Hardware

The advantages of an AWS-managed service over an on-premise cluster are as follows:

1. **Cost:** Setting up an on-premise cluster involves a huge investment in hardware, although in the case of cloud, it is pay as you go. In other words, you do not buy any hardware; you rent the necessary hardware from the cloud and then pay as per your usage.
2. **Services:** While handling any big data task, you may require a lot of services based on the business use case. It is tricky to install these services and manage the software in an on-premise cluster, although this is not the case with cloud where AWS manages all of that for you.
3. **Quick upgrades:** All the associated services would undergo frequent updates and improvements. To have these upgrades be reflected in your on-premise cluster, you need to update your entire infrastructure; however, in the case of a cloud cluster, you can choose the updated version of services and launch a new cluster.
4. **Quick prototyping:** To build some quick prototypes, you may need to make changes to your existing environment. For this, you need to have a similar test cluster where you can experiment; but in the case of an on-premise cluster, this would require additional resources and would need you to go through the hassle of setting up the cluster. This can be handled by cloning an existing cluster in the case of cloud services.
5. **Security:** Network intrusions are a common phenomenon when handling a distributed system. In the case of cloud services, these are handled by the cloud provider.

Hadoop Ecosystem

Some commonly used components of the Hadoop stack are as follows:

1. **Sqoop:** This is an application that provides a command-line interface for transferring data between Hadoop and relational databases.
2. **Flume:** This is a distributed, reliable and available service for efficiently collecting, aggregating and moving large amounts of streaming event data. It is also used for ingesting massive quantities of event data, such as network traffic data, social-media-generated data and data from email messages.

3. **Oozie and Airflow:** These are workflow scheduler systems for Hadoop, providing control over complex multistage Hadoop jobs.
4. **Hive and Presto:** These are data warehouses that provide SQL-like interfaces to query databases and file systems integrated with Hadoop.
5. **Pig:** This is a high-level language for writing programs that run on Apache Hadoop. It allows creating user-defined functions, thereby making programming much simpler than writing MapReduce codes.
6. **HBase:** This is a column-based NoSQL database that provides high throughput and low latency on Hadoop platforms. It provides a fault-tolerant method to store large quantities of sparse data and perform real-time analytics.
7. **Spark:** This is a fast and general computer engine for Hadoop data. It provides a simple and expressive programming model to support machine learning.

Introduction to EMR (Elastic MapReduce)

1. EMR is a managed service provided by AWS; it is built on top of Hadoop. It is referred to as a managed service because, unlike an on-premise cluster, you do not have to install Hadoop manually and perform the necessary configuration; AWS manages all of that for you.
2. All the services, such as Spark for running big data processing jobs; Oozie to schedule those jobs, Presto or Hive to query the HDFS data using SQL; and a lot more, can be installed on the same EMR cluster by just selecting the list of services that you need at the time of installation.
3. EMR enables on-demand utilisation of resources. In other words, you can simply terminate the cluster if you no longer require it and then clone it back whenever you need it in the future.
4. EMR gives you an additional benefit of being able to use all other AWS services, as they are tightly integrated with each other. For example, instead of storing your HDFS on a disk, you can leverage S3, and EMR will still work just fine.

You can go to this [link](#) for the EMR Documentation.

Working with the EMR Cluster

Launching and Connecting to the EMR Cluster



You can explore more about these tabs [here](#).

Working with the HDFS

<code>sudo initctl list</code>	Using this command, you can view the different services running on your Hadoop cluster. These services might go down frequently and stop responding. The first step to follow in such a scenario is to connect to the master node and restart the service.
<code>sudo start hadoop-yarn-resourcemanager</code>	Using this command, you can start a service. We have used the 'start' command to start the resourcemanager.
<code>hadoop fs -ls /</code>	This command is used to list the contents of the HDFS.

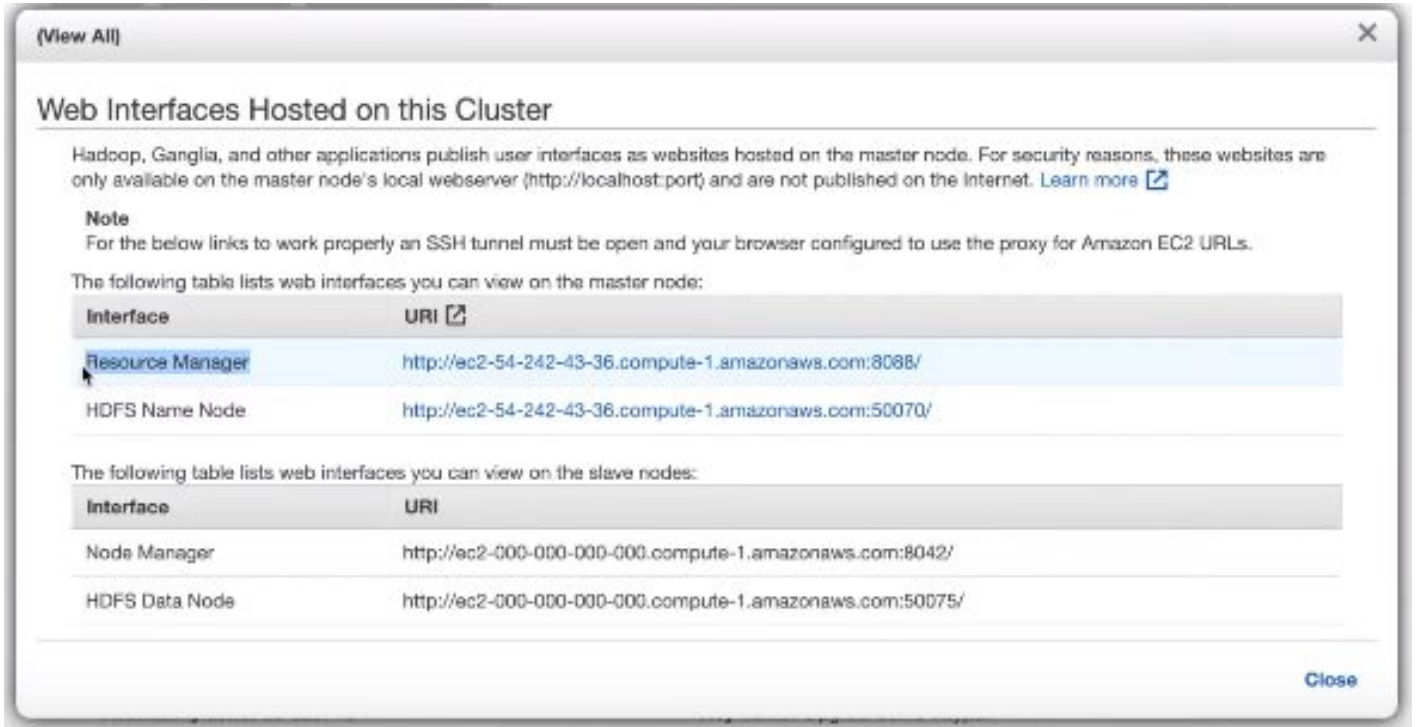
hadoop fs -mkdir /input1	This command is used to create a new directory in the present working directory.
vi text file	This command is used to open the vi editor
hadoop fs -put test.txt /test_data/	This command is used to copy the file in the present working directory to a new directory named test_data.
hdfs fs -help	Using the help command, you can see all related HDFS commands.
hdfs fs -rm	This command is used to delete directories and files.
hadoop fs -distcp	This command allows the user to perform intercluster/intracluster copying.

Running a MapReduce Job on EMR

1. Once you start your EMR cluster, go to the Steps tab and click on 'Add Step'. Here, you have the option of selecting a custom JAR (Java program), Hive program, Pig program or Spark application. In case you want to execute a code that does not belong to any of the previous programs, you can select 'streaming programs'.
2. After that, select a name for your program. Provide the path for the mapper and reducer codes in the S3 bucket or the Hadoop location. EMR also supports the aggregate keyword. Next, add the input and output S3 locations, and the action (Continue/Cancel and Wait/Terminate Cluster) taken by the cluster if the job fails.
3. Once you click on 'Add', the job will be submitted to the YARN.
4. After the job is completed, you can check the output in the output location in the S3 bucket provided to the EMR.

Running a MapReduce Job on EMR

You can refer to the web URLs of the resource manager and the HDFS NameNode to conduct a quick analysis of the cluster.



(View All) X

Web Interfaces Hosted on this Cluster

Hadoop, Ganglia, and other applications publish user interfaces as websites hosted on the master node. For security reasons, these websites are only available on the master node's local webserver (<http://localhost:port>) and are not published on the internet. [Learn more](#)

Note
For the below links to work properly an SSH tunnel must be open and your browser configured to use the proxy for Amazon EC2 URLs.

The following table lists web interfaces you can view on the master node:

Interface	URI
Resource Manager	http://ec2-54-242-43-36.compute-1.amazonaws.com:8088/
HDFS Name Node	http://ec2-54-242-43-36.compute-1.amazonaws.com:50070/

The following table lists web interfaces you can view on the slave nodes:

Interface	URI
Node Manager	http://ec2-000-000-000-000.compute-1.amazonaws.com:8042/
HDFS Data Node	http://ec2-000-000-000-000.compute-1.amazonaws.com:50075/

Close

Since the cost associated with EMR is very high, you cannot leave it idle and should terminate the cluster if you are not using it for a while. At times, if you are running a single MapReduce job or a simple Hive query, then you can choose to terminate the cluster while specifying the job under the Steps tab.

You can also clone the terminated EMR cluster by selecting the Clone option. However, in this case, you might lose all the data associated with the cluster. Therefore, avoid putting data in the EMR cluster; instead, store the data in S3.

Disclaimer: All content and material on the upGrad website is copyrighted material, either belonging to upGrad or its bonafide contributors, and is purely for the dissemination of education. You are permitted to access, print and download extracts from this site purely for your own education only and on the following basis:

- You can download this document from the website for self-use only.
- Any copy of this document, in part or full, saved to disk or to any other storage medium, may only be used for subsequent, self-viewing purposes or to print an individual extract or copy for non-commercial personal use only.
- Any further dissemination, distribution, reproduction, copying of the content of the document herein or the uploading thereof on other websites or use of content for any other commercial/unauthorised purposes in any way which could infringe the intellectual property rights of upGrad or its contributors, is strictly prohibited.
- No graphics, images or photographs from any accompanying text in this document will be used separately for unauthorised purposes.
- No material in this document will be modified, adapted or altered in any way.
- No part of this document or upGrad content may be reproduced or stored in any other website or included in any public or private electronic retrieval system or service without upGrad's prior written permission.
- Any right not expressly granted in these terms are reserved.