SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

AND

DEPARTMENT OF INFORMATION TECHNOLOGY

UNIT – IV – SCSA1603 – BIG DATA ANALYTICS

Mapreduce

Hadoop

• Big Data Technology

• Distributed processing of large data sets

• Open source

• Map Reduce- Simple Programming model

Why Hadoop?

• Handles any data type

➢ Structured/unstructured

➢ Schema/no Schema

➢ High volume/Low volume

➢ All kinds of analytic applications

• Grows with business

• Proven with Petabyte scale

• Capacity & Performance grows

• Leverages commodity hardware to mitigate costs

Hadoop Features

• 100% Apache open source

• No Vendor locking

• Rich Eco system & community development

• To Derive compute value of all data

• More affordable cost-effective platform

Hadoop: It is an open-source software framework used for storing data and running applications on a group of commodity hardware. It has large storage capacity and high processing power. It can manage multiple concurrent processes at the same time. It is used in predictive analysis, data mining and machine learning. It can handle both structured and unstructured form of data. It is more flexible in storing, processing, and managing data than traditional RDBMS. Unlike traditional systems, Hadoop enables multiple analytical processes on the same data at the same time. It supports scalability very flexibly.

Hadoop Key Benefits

1. Scalable
2. Cost effective
3. Flexible
4. Fast
5. Resilient to failure

1. Scalable

Hadoop is a highly scalable storage platform, because it can store and distribute very large data sets across hundreds of inexpensive servers that operate in parallel. Unlike traditional relational database systems (RDBMS) that can't scale to process large amounts of data, Hadoop enables businesses to run applications on thousands of nodes involving thousands of terabytes of data.

2. Cost effective

Hadoop also offers a cost-effective storage solution for businesses' exploding data sets. The problem with traditional relational database management systems is that it is extremely cost prohibitive to scale to such a degree in order to process such massive volumes of data. In an effort to reduce costs, many companies in the past would have had to down-sample data and classify it based on certain assumptions as to which data was the most valuable. The raw data would be deleted, as it would be too costprohibitive to keep. While this approach may have worked in the short term, this meant that when business priorities changed, the complete raw data set was not available, as it was too expensive to store. Hadoop, on the other hand, is designed as a scale-out architecture that can affordably store all of a company's data for later use. The cost savings are staggering: instead of costing thousands to tens of thousands of pounds per terabyte, Hadoop offers computing and storage capabilities for hundreds of pounds per terabyte.

3. Flexible

Hadoop enables businesses to easily access new data sources and tap into different types of data (both structured and unstructured) to generate value from that data. This means businesses can use Hadoop to derive valuable business insights from data sources such as social media, email conversations or click stream data. In addition, Hadoop can be used for a wide variety of purposes, such as log processing, recommendation systems, data warehousing, and market campaign analysis and fraud detection.

4. Fast

Hadoop's unique storage method is based on a distributed file system that basically 'maps' data wherever it is located on a cluster. The tools for data processing are often on the same servers where the data is located, resulting in much faster data processing. If you're dealing with large volumes of unstructured data, Hadoop is able to efficiently process terabytes of data in just minutes, and petabytes in hours.

5. Resilient to failure A key advantage of using Hadoop is its fault tolerance. When data is sent to an individual node, that data is also replicated to other nodes in the cluster, which means that in the event of failure, there is another copy available for use. The Map R distribution goes beyond that by eliminating the Name Node and replacing it with a distributed No Name Node architecture that provides true high availability. Our architecture provides protection from both single and multiple failures. When it comes to handling large data sets in a safe and cost-effective manner, Hadoop has the advantage over relational database management systems, and its value for any size business will continue to increase as unstructured data continues to grow.

Hadoop Infrastructure

The 2 infrastructure models of Hadoop are:

✓ Data Model

✓ Computing Model

Traditional Database Model Vs Hadoop Data Model

DISTRIBUTED DATABASE MODEL	HADOOP DATA MODEL
Deals with tables and relations	Deals with flat files in any format
Must have a schema for data	No schema
Data fragmentation and partitioning	Files are divided automatically into blocks

A distributed database consists of a network of many interconnected physical databases that spread across various geographical locations. The separate databases are periodically synchronized for ensuring all have consistent data.

Mapreduce

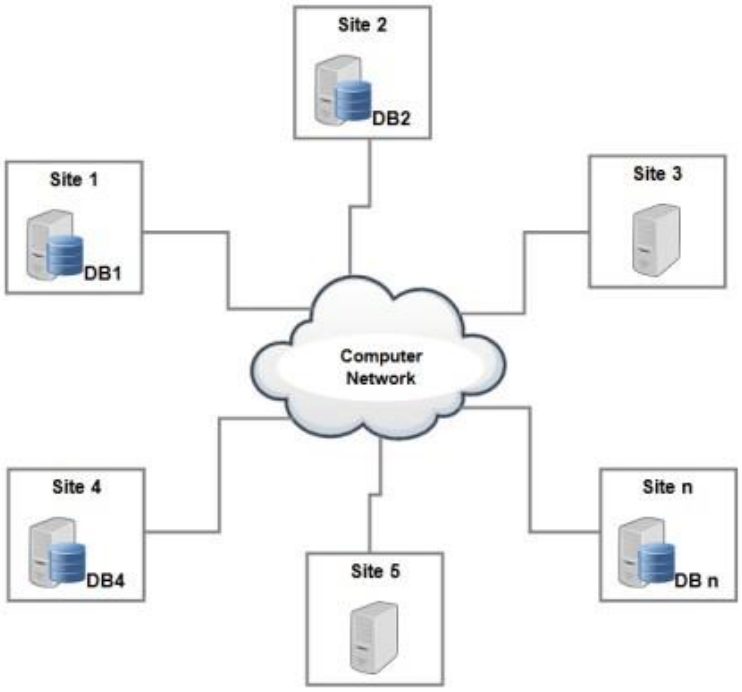


Fig. 2.3 Distributed Database Model

A Hadoop data model consists of a Blocks of Distributed File Systems under a specific Compute Cluster, where the DFS blocks are interconnected with communication channel. This makes sure that the centre is in one place for the data sent to be mapped and reduced.

Traditional Vs Computing Model

DISTRIBUTED DATABASES	HADOOP
Notion of a transaction	Notion of a job divided into tasks
Distributed transaction with ACID properties	MapReduce computing model where every task is either a map or reduce service

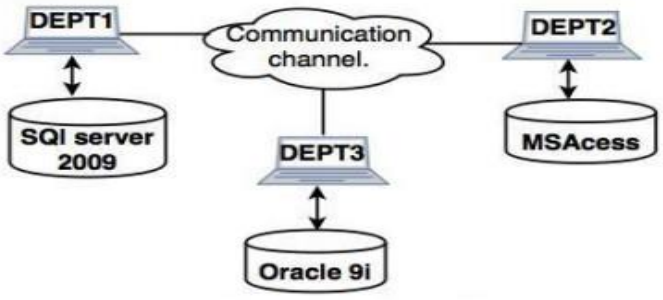


Fig. 2.5 Distributer Databases

Mapreduce

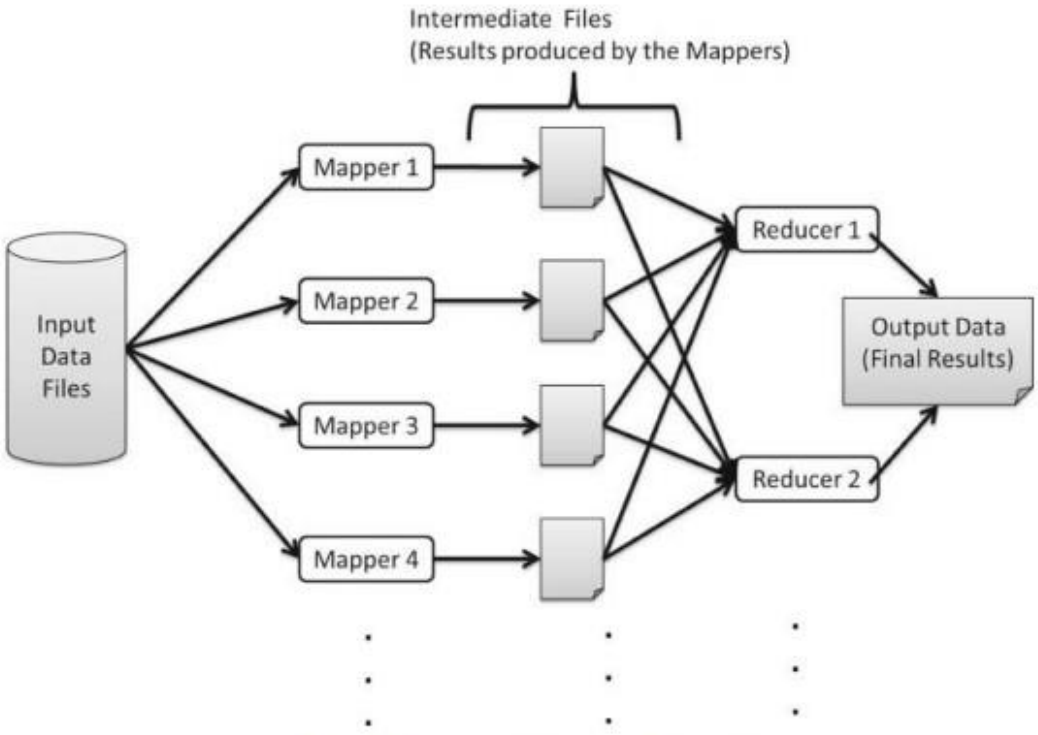


Fig. 2.6 Hadoop – Computing Model

Hadoop Architecture

What is Architecture of Hadoop?

- ✓ Hadoop is the open-source framework of Apache Software Foundation, which is used to store and process large unstructured datasets in the distributed environment.
- ✓ Data is first distributed among different available clusters then it is processed.
- ✓ Hadoop biggest strength is that it is scalable in nature means it can work on a single node to thousands of nodes without any problem.
- ✓ Hadoop framework is based on Java programming and it runs applications with the help of Map Reduce which is used to perform parallel processing and achieve the entire statistical analysis on large datasets.
- ✓ Distribution of large datasets to different clusters is done on the basis of Apache Hadoop software library using easy programming models.
- ✓ Organizations are now adopting Hadoop for the purpose of reducing the cost of data storage.
- ✓ It will lead to the analytics at an economical cost which will maximize the business profitability.

- ✓ For a good Hadoop architectural design, you need to take in considerations – good computing power, storage, and networking

Hadoop Architecture

- ✓ Hadoop ecosystem consists of various components such as Hadoop Distributed File System
- ✓ (HDFS), Hadoop MapReduce, Hadoop Common, HBase, YARN, Pig, Hive, and others.
- ✓ Hadoop components which play a vital role in its architecture are-
- ✓ Hadoop Distributed File System (HDFS)
- ✓ Hadoop MapReduce
- ✓ Hadoop works on the master/slave architecture for distributed storage and distributed computation.
- ✓ NameNode is the master and the DataNodes are the slaves in the distributed storage.
- ✓ The Job Tracker is the master and the Task Trackers are the slaves in the distributed computation.
- ✓ The slave nodes are those which store the data and perform the complex computations.
- ✓ Every slave node comes with a Task Tracker daemon and a Data Node synchronizes the processes with the Job Tracker and Name Node respectively.
- ✓ In Hadoop architectural setup, the master and slave systems can be implemented in the cloud or on-site premise

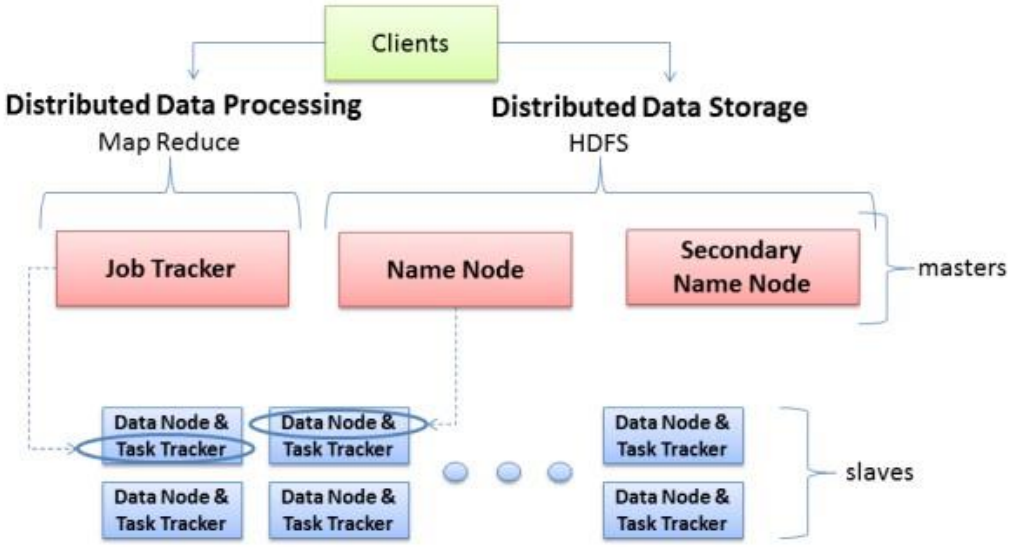


Fig. 2.7 Hadoop Architecture

Hadoop Framework

There are 2 Main layers:

➢ HDFS

➢ Execution Engineer

HDFS

Role of HDFS in Hadoop Architecture

- ✓ HDFS is used to split files into multiple blocks. Each file is replicated when it is stored in Hadoop cluster.
- ✓ The default size of that block of data is 64 MB but it can be extended up to 256 MB as per the requirement.
- ✓ HDFS stores the application data and the file system metadata on two different servers.
- ✓ Name Node is used to store the file system metadata while and application data is stored by the Data Node.
- ✓ To ensure the data reliability and availability to the highest point, HDFS replicates the file content many times.
- ✓ Name Node and Data Node communicate with each other by using TCP protocols.
- ✓ Hadoop architecture performance depends upon Hard-drives throughput and the network speed for the data transfer.

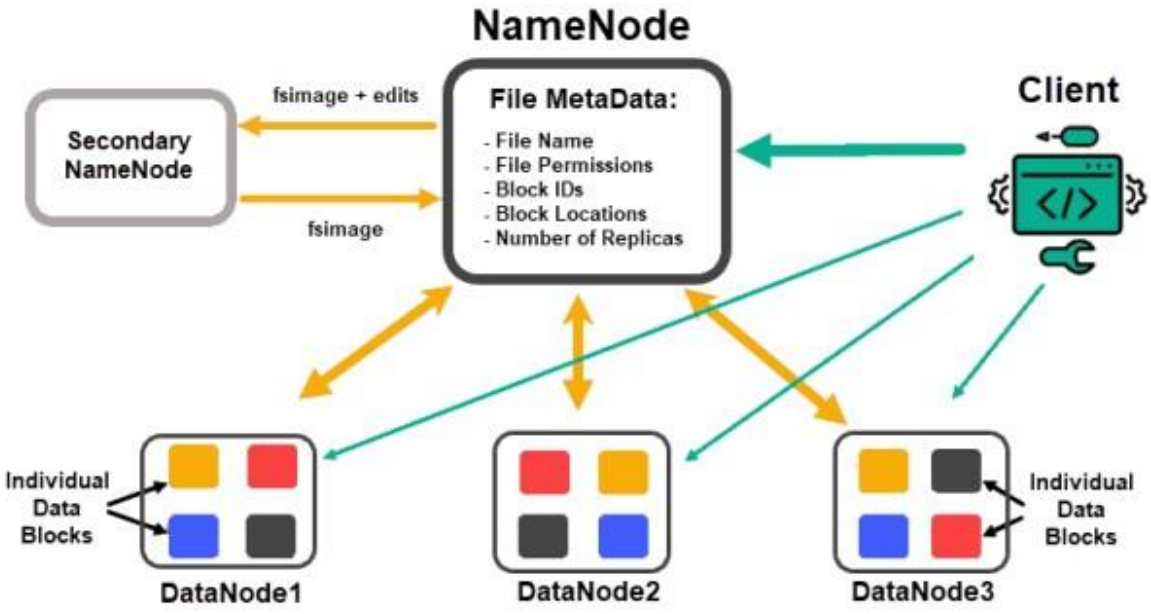


Fig. 2.9 HDFS

What is HDFS in Hadoop?

- The Hadoop Distributed File System is a java-based file, developed by Apache Software
- Foundation with the purpose of providing versatile, resilient, and clustered approach to manage files in a Big Data environment using commodity servers.
- HDFS used to store a large amount of data by placing them on multiple machines as there are hundreds and thousands of machines connected together.
- The goal is to store a smaller number of larger files rather than the greater number of small files
- HDFS provides high reliability of data because it used to replicate data into three different copies- two are saved in one group and third one in another.
- HDFS is scalable in nature as it can be extended to 200 PB of storage where a single cluster contains 4500 servers, supporting billions of blocks and files.

How HDFS Works

- Hadoop works on a master node which is Name Node and multiple slave nodes which are
- Data Nodes on a commodity cluster.
- As all the nodes are present in the same rack in the data center, data is broken into different blocks that are distributed among different nodes for the storage.
- These blocks are replicated across nodes to make data available in case of a failure.

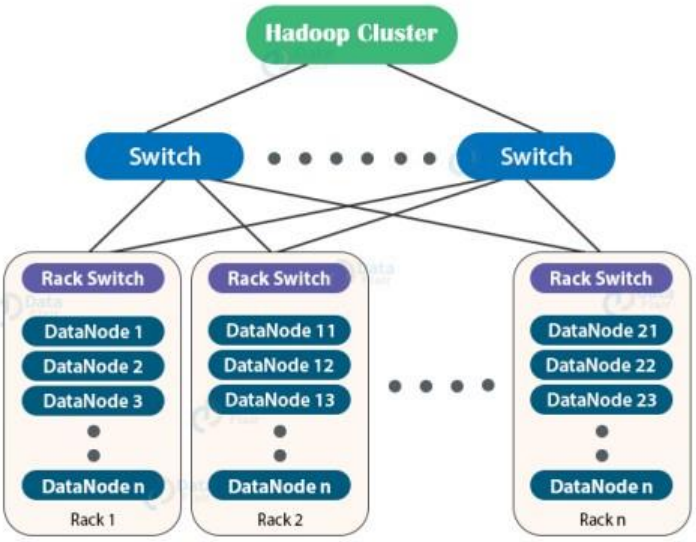


Fig. 2.10 HDFS Working

- The NameNode is known as a smart node in the cluster.

- NameNode knows which data node contains which blocks and where data node has been placed in the clusters.
- The NameNode also contains the access authority which is given to files to perform read, write, create, delete and replication of various blocks between the data nodes.
- The NameNode is connected with the data nodes in the loosely coupled fashion
- This provides the feature of scalability in real time means it can add or delete nodes as per the requirement.
- Data nodes used to communicate every time with the NameNode to check whether the certain task is completed or not.
- Communication between these two ensures that NameNode knows the status of each data node all the time.
- If one of the data nodes is not functioning properly then NameNode can assign that task to another node.
- To operate normally, data nodes in the same block communicate with each other.
- The NameNode is replicated to overcome system failure and is the most critical part of the whole system.
- Data nodes are not considered to be smart, but flexible in nature and data blocks are replicated across various nodes and the NameNode is used to manage the access.
- To gain the maximum efficiency, replication mechanism is used and all the nodes of the cluster are placed into a rack. Rack Id is used by the NameNode to track data nodes in the cluster.
- A heartbeat message is put through to ensure that the data nodes and the NameNode are still connected.
- When the heartbeat is no longer available, then the NameNode detach that data node from the cluster and works in the normal manner.
- When the heartbeat comes, data node is added back to the cluster.
- Transaction log and the checksum are used to maintain the data integrity.
- Transaction log used to keep track of every operation and help them in auditing and rebuilding the file system, in case of an exception.
- Checksum validates the content in HDFS.

- When a user requests a file, it verifies the checksum of that content.
- If checksum validation matches then they can access it.
- If the checksum reports an error, then the file is hidden to avoid tampering.
- The performance also depends on where the data is stored, so it is stored on local disk in the commodity servers.
- To ensure that one server failure doesn't corrupt the whole file, data blocks are replicated in various data nodes.
- The degree of replication and the number of data nodes are managed at a time when the cluster is implemented.

Features of HDFS

✓ Fault-Tolerant
✓ Scalability
✓ Data Availability
✓ Data Reliability
✓ Replication

Description to Features of HDFS

Fault-Tolerant

- HDFS is highly fault-tolerant.
- HDFS replicates and stores data in three different locations.
- So, in the case of corruption or unavailability, data can be accessed from the previous location.

Scalability

- Scalability means adding or subtracting the cluster from HDFS environment.
- Scaling is done by the two ways – vertical or horizontal.
- In vertical scaling, you can add up any number of nodes to the cluster but there is some downtime.
- In horizontal scaling, there is no downtime; you can add any number of nodes in the cluster in real time.

Data Availability

- Data is replicated and stored on different nodes due to which data is available all the time.

- In case of network, node or some hardware failure, data is accessible without any trouble from a different source or from a different node.

Data Reliability

- HDFS provides highly reliable data storage, as data are divided into blocks and each block is replicated in the cluster which made data reliable.
- If one node contains the data is down, it can be accessed from others because HDFS creates three replicas of each block.
- When there is no loss of data in case of failure then it is highly reliable.

Replication

- Data replication is one of the unique and important features of HDFS.
- HDFS used to create replicas of data in the different cluster.
- As if one node goes down it can be accessed from other because every data block has three replicas created.
- This is why; there is no chance of data loss.

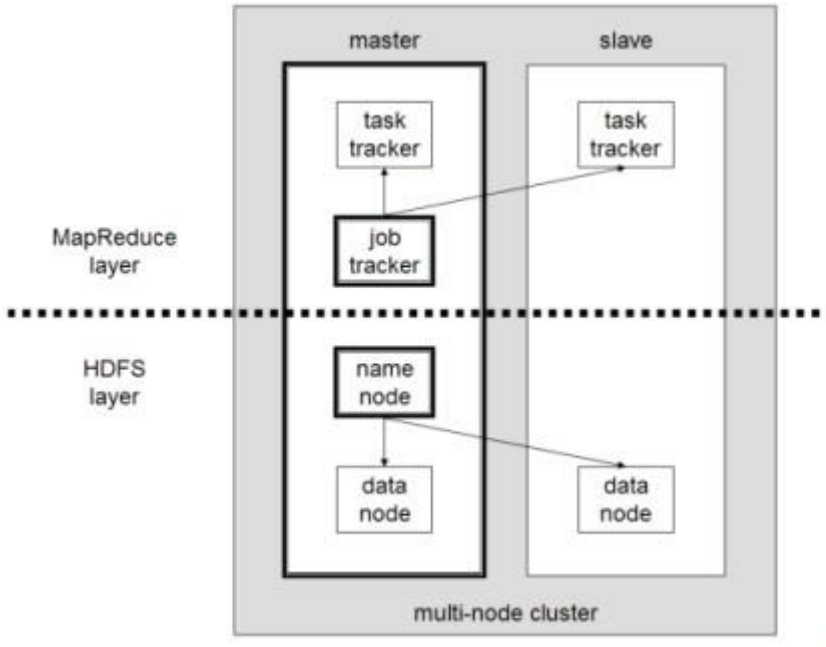


Fig. 2.11 HDFS Layer

Mapreduce

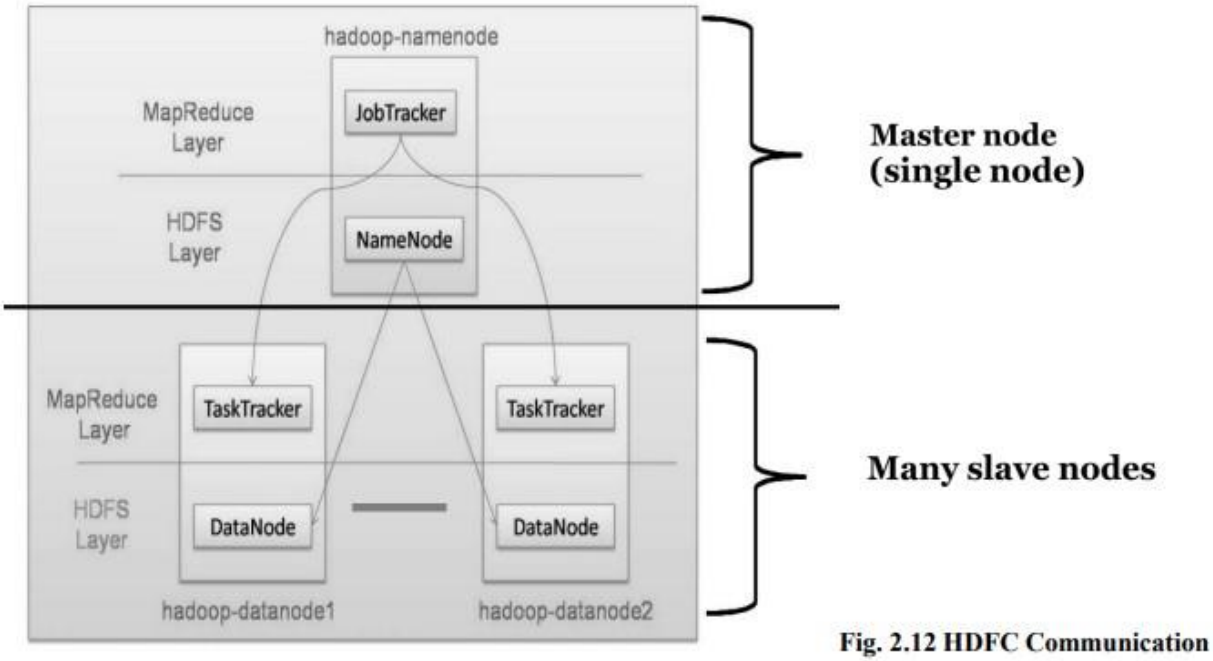


Fig. 2.12 HDFC Communication

Components of HDFS

Name Node

• Single node in cluster

• Brain of HDFS

• Hadoop employs master/slave architecture for both distributed storage and distributed computation

• Distributed Storage system is HDFS

• Name Node is the master of HDFS that directs slave Data Node to perform low level tasks.

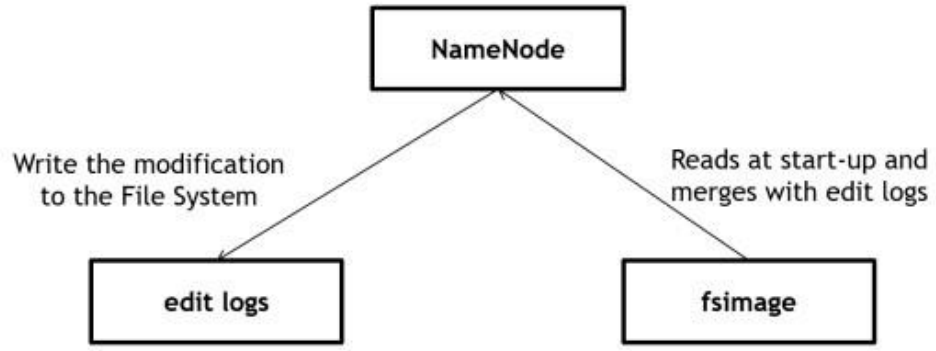


Fig. 2.13 Components of HDFS

Name Node as book Keeper of HDFS

• Name Node list of all blocks of file and list of all data nodes that contains the block

• It keeps track of how files are stored into file blocks, which nodes store these blocks and overall health of HDFS

• The function of Name Node is memory and IO intensive

• The server hosting Name Node doesn't store any user data or perform any computation

Name node – Drawback

• Name node is the Single point of failure in Hadoop cluster

• For other data nodes, if their host node fails due to h/w or s/w reasons, the Hadoop cluster will continue smoothly

Data Nodes

• Each slave machine in the cluster will host a data node to perform grunt work of Distributed File System

• Reading and writing HDFS blocks to actual files on local file system

• During r/w a HDFS file, the file is broken into blocks and NameNode will tell the client which data node each block resides in.

• Client communicates directly with the DataNodes to process local files corresponding to the blocks

• Data node may communicate with other DataNodes to replicate data blocks for redundancy

Data Replication

• HDFS is designed to reliably store very large files across machines in a large cluster.

• It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size.

• The blocks of a file are replicated for fault tolerance.

• The block size and replication factor are configurable per file.

• An application can specify the number of replicas of a file.

• The replication factor can be specified at file creation time and can be changed later.

• Files in HDFS are write-once and have strictly one writer at any time.

• The Name Node makes all decisions regarding replication of blocks.

Heartbeat and Block report of data nodes

• NameNode periodically receives a Heartbeat and a Block report from each of the DataNodes in the cluster

• Receipt of a Heartbeat implies that the DataNode is functioning properly

• A Block report contains a list of all blocks on a DataNode

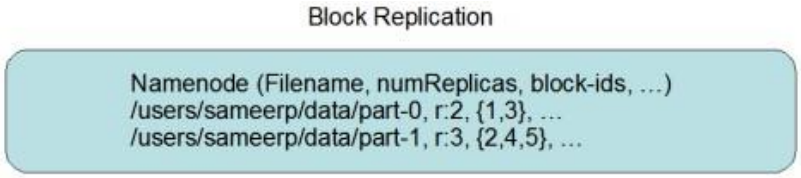


Fig. 2.14 Block Replication

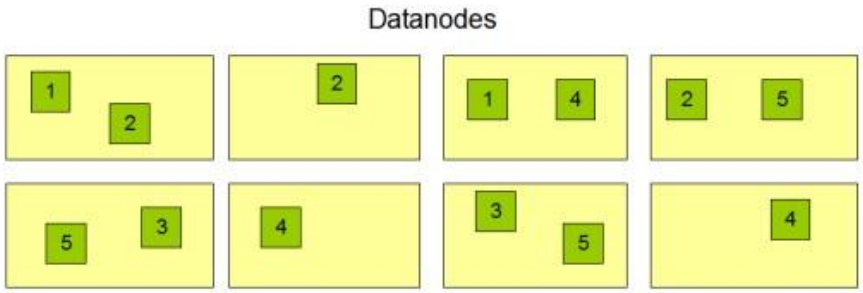


Fig. 2.15 DataNodes

Replication factor

When the replication factor is three, HDFS's placement policy is to put:

1. One replica:
    - On the local machine if the writer is on a DataNode,
    - Otherwise on a random DataNode,
2. Another replica on a node in a different (remote) rack,
3. Last on a different node in the same remote rack.

This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure.

DataNodes and NameNodes

• Data Node constantly report to Name Node

• Upon initialization, each DataNode informs NameNode of the blocks it is currently storing

• After mapping, the DataNode continually poll the NameNode to provide information regarding local changes and receive instructions to create, move or delete blocks from local disk.

Job Tracker

• Job tracker is a liaison between the client application and Hadoop

• Once the client submits the code to the cluster the JobTracker determines the execution plan by determining which files to process

• Assigns nodes to different tasks and monitors all tasks as they are running

• If a task fails, the job tracker will automatically relaunch the task on a different node

Task Tracker

• Job tracker is the master overseeing the overall execution of MapReduce job

• Task trackers manages the execution of individual tasks

• Task tracker can spawn multiple JVMs to handle many map/reduce tasks in parallel

• Main responsibility is to constantly communicate with Job tracker

• If Job tracker fails to receive a heartbeat from TaskTracker within the specified amount of time, it will assume task tracker has crashed and will resubmit the corresponding tasks to other nodes in the cluster.

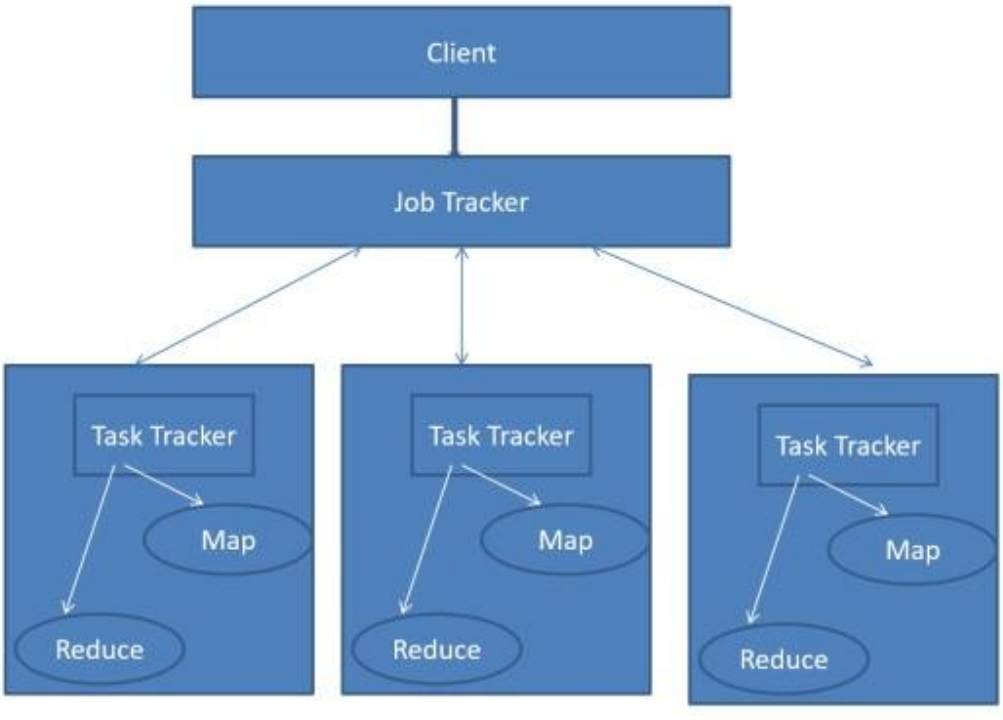


Fig. 2.17 Task Tracker

HDFS

• HDFS is the first building block of Hadoop cluster

• HDFS breaks incoming files into blocks and stores them redundantly across the cluster

• To efficiently process massive amounts of data, it is important to move

• A single large file splits into blocks and blocks are distributed among the nodes of Hadoop cluster

• The blocks used in HDFS are large 128 MB or more compared to small blocks associated with traditional file systems

• This allows the system to scale without increasing the size and complexity of HDFS metadata

• Files in HDFS are write once files

• Input data is loaded into HDFS and processed by MapReduce framework.

Data Loading techniques

HDFS READS

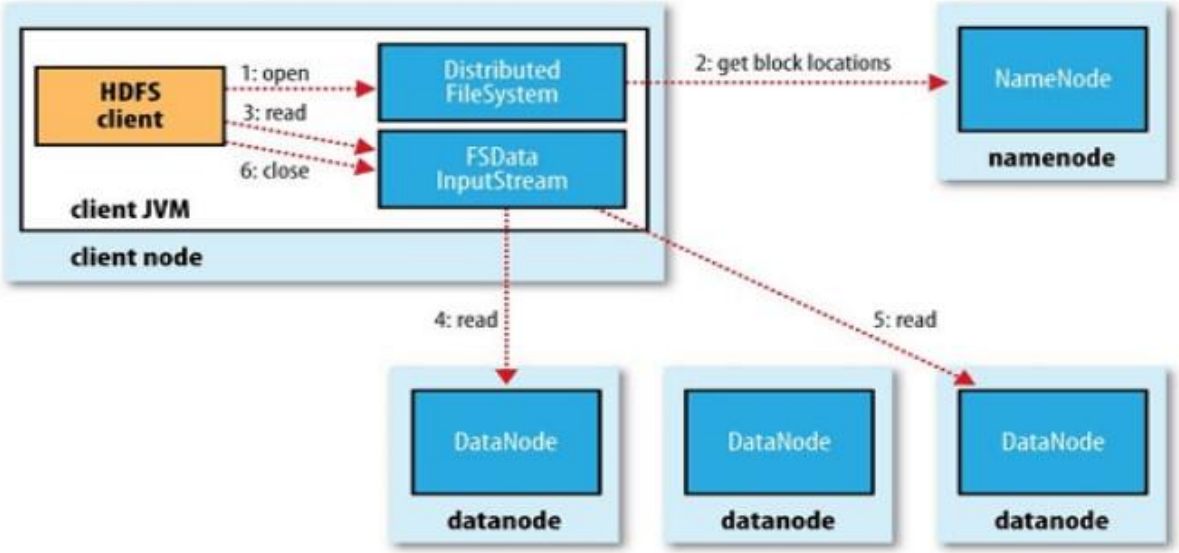


Fig. 2.19 HDFS Reads

HDFS Files and Blocks

• Files are stored as a collection of Blocks

• Blocks: The minimum amount of data that can be read or written

• Typically, 64MB of unit size is default

• Block details are stored on 3 nodes

➢ The Name node – Manages meta-data about files & Blocks

➢ SNN – Holds backup of Name node data

➢ Data Node – Store and serve blocks

➢ Multiple copies of a block are stored

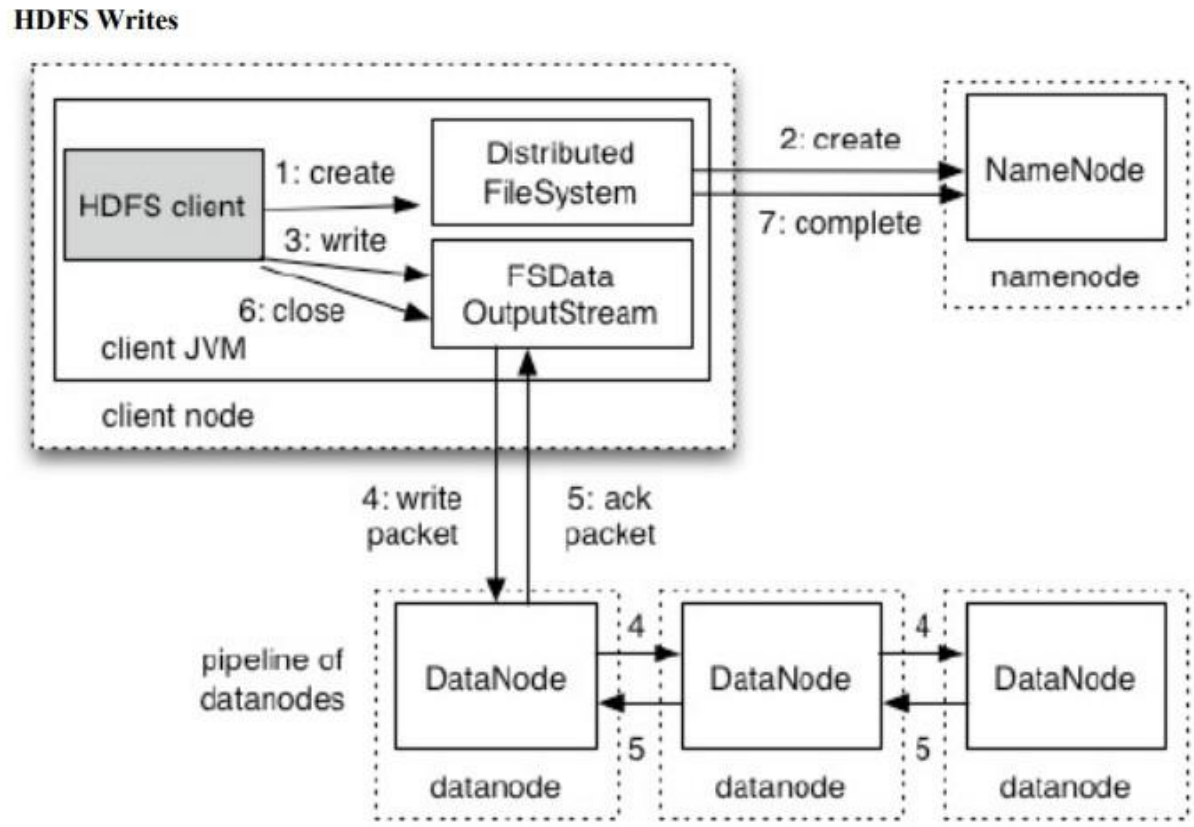


Fig. 2.20 HDFS Write

In conclusion, HDFS empowers Hadoop functionality.

HDFS provides highly reliable data storage despite of any hardware failure.

It is highly fault-tolerant, provides high data availability, and high scalability.

Analysing data with Hadoop

Big Data is unwieldy because of its vast size, and needs tools to efficiently process and extract meaningful results from it. Hadoop is an open source software framework and platform for storing, analysing and processing data. This article is a beginner's guide to how Hadoop can help in the analysis of Big Data.

Big Data is a term used to refer to a huge collection of data that comprises both structured data found in traditional databases and unstructured data like text documents, video and audio. Big

Data is not merely data but also a collection of various tools, techniques, frameworks and platforms. Transport data, search data, stock exchange data, social media data, etc, all come under Big Data.

Technically, Big Data refers to a large set of data that can be analysed by means of computational techniques to draw patterns and reveal the common or recurring points that would help to predict the next step—especially human behaviour, like future consumer actions based on an analysis of past purchase patterns.

Big Data is not about the volume of the data, but more about what people use it for. Many organizations like business corporations and educational institutions are using this data to analyse and predict the consequences of certain actions. After collecting the data, it can be used for several functions like:

Cost reduction

The development of new products

Making faster and smarter decisions

Detecting faults

Today, Big Data is used by almost all sectors including banking, government, manufacturing, airlines and hospitality.

There are many open source software frameworks for storing and managing data, and Hadoop is one of them. It has a huge capacity to store data, has efficient data processing power and the capability to do countless jobs. It is a Java based programming framework, developed by Apache. There are many organisations using Hadoop — Amazon Web Services, Intel, Cloudera, Microsoft, MapR Technologies, Teradata, etc.

The history of Hadoop

Doug Cutting and Mike Cafarella are two important people in the history of Hadoop. They wanted to invent a way to return Web search results faster by distributing the data over several machines and make calculations, so that several jobs could be performed at the same time. At that time, they were working on an open source search engine project called Nutch. But, at the same time, the Google search engine project also was in progress. So, Nutch was divided into two parts—one of the parts dealt with the processing of data, which the duo named Hadoop after the toy elephant that belonged to Cutting's son. Hadoop was released as an open source

project in 2008 by Yahoo. Today, the Apache Software Foundation maintains the Hadoop ecosystem.

Prerequisites for using Hadoop

Linux based operating systems like Ubuntu or Debian are preferred for setting up Hadoop. Basic knowledge of the Linux commands is helpful. Besides, Java plays an important role in the use of Hadoop. But people can use their preferred languages like Python or Perl to write the methods or functions.

There are four main libraries in Hadoop.

1. Hadoop Common: This provides utilities used by all other modules in Hadoop.

2. Hadoop MapReduce: This works as a parallel framework for scheduling and processing the data.

3. Hadoop YARN: This is an acronym for Yet Another Resource Navigator. It is an improved version of MapReduce and is used for processes running over Hadoop.

4. Hadoop Distributed File System – HDFS: This stores data and maintains records over various machines or clusters. It also allows the data to be stored in an accessible format.

HDFS sends data to the server once and uses it as many times as it wants. When a query is raised, NameNode manages all the DataNode slave nodes that serve the given query. Hadoop MapReduce performs all the jobs assigned sequentially. Instead of MapReduce, Pig Hadoop and Hive Hadoop are used for better performances.

Other packages that can support Hadoop are listed below.

Apache Oozie: A scheduling system that manages processes taking place in Hadoop

Apache Pig: A platform to run programs made on Hadoop

Cloudera Impala: A processing database for Hadoop. Originally it was created by the software organization Cloudera, but was later released as open source software

Apache HBase: A non-relational database for Hadoop

Apache Phoenix: A relational database based on Apache HBase

Apache Hive: A data warehouse used for summarisation, querying and the analysis of data

Apache Sqoop: Is used to store data between Hadoop and structured data sources

Apache Flume: A tool used to move data to HDFS

Cassandra: A scalable multi-database system

The importance of Hadoop

Hadoop is capable of storing and processing large amounts of data of various kinds. There is no need to preprocess the data before storing it. Hadoop is highly scalable as it can store and distribute large data sets over several machines running in parallel. This framework is free and uses cost-efficient methods.

Hadoop is used for:

Machine learning

Processing of text documents

Image processing

Processing of XML messages

Web crawling

Data analysis

Analysis in the marketing field

Study of statistical data

Challenges when using Hadoop

Hadoop does not provide easy tools for removing noise from the data; hence, maintaining that data is a challenge. It has many data security issues like encryption problems. Streaming jobs and batch jobs are not performed efficiently. MapReduce programming is inefficient for jobs involving highly analytical skills. It is a distributed system with low level APIs. Some APIs are not useful to developers.

But there are benefits too. Hadoop has many useful functions like data warehousing, fraud detection and marketing campaign analysis. These are helpful to get useful information from the collected data. Hadoop has the ability to duplicate data automatically. So multiple copies of data are used as a backup to prevent loss of data.

Frameworks similar to Hadoop

Any discussion on Big Data is never complete without a mention of Hadoop. But like with other technologies, a variety of frameworks that are similar to Hadoop have been developed. Other frameworks used widely are Ceph, Apache Storm, Apache Spark, DataTorrentRTS, Google BiqQuery, Samza, Flink and HydraDataTorrentRTS.

MapReduce requires a lot of time to perform assigned tasks. Spark can fix this issue by doing in-memory processing of data. Flink is another framework that works faster than Hadoop and Spark. Hadoop is not efficient for real-time processing of data. Apache Spark uses stream processing of data where continuous input and output of data happens. Apache Flink also provides single runtime for the streaming of data and batch processing.

However, Hadoop is the preferred platform for Big Data analytics because of its scalability, low cost and flexibility. It offers an array of tools that data scientists need. Apache Hadoop with YARN transforms a large set of raw data into a feature matrix which is easily consumed. Hadoop makes machine learning algorithms easier.

Streaming

Hadoop streaming is a utility that comes with the Hadoop distribution. This utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer.

Example Using Python

For Hadoop streaming, we are considering the word-count problem. Any job in Hadoop must have two phases: mapper and reducer. We have written codes for the mapper and the reducer in python script to run it under Hadoop. One can also write the same in Perl and Ruby.

Mapper Phase Code

```
!/usr/bin/python
import sys
# Input takes from standard input for myline in sys.stdin:
  # Remove whitespace either side
  myline = myline.strip()
  # Break the line into words
  words = myline.split()
  # Iterate the words list
  for myword in words:
    # Write the results to standard output
    print '%s\t%s' % (myword, 1)
```

Make sure this file has execution permission (chmod +x /home/ expert/hadoop-1.2.1/mapper.py).

Reducer Phase Code

```
#!/usr/bin/python
from operator import itemgetter
import sys
current_word = ""
current_count = 0
word = ""
# Input takes from standard input for myline in sys.stdin:
  # Remove whitespace either side
  myline = myline.strip()
  # Split the input we got from mapper.py word,
  count = myline.split('\t', 1)
  # Convert count variable to integer
  try:
    count = int(count)
  except ValueError:
    # Count was not a number, so silently ignore this line continue
  if current_word == word:
  current_count += count
  else:
    if current_word:
      # Write result to standard output print '%s\t%s' % (current_word, current_count)
      current_count = count
    current_word = word
# Do not forget to output the last word if needed!
if current_word == word:
  print '%s\t%s' % (current_word, current_count)
```

Save the mapper and reducer codes in mapper.py and reducer.py in Hadoop home directory. Make sure these files have execution permission (chmod +x mapper.py and chmod +x reducer.py). As python is indentation sensitive so the same code can be download from the below link.

Execution of WordCount Program

```
$ $HADOOP_HOME/bin/hadoop jar contrib/streaming/hadoop-streaming-1.
2.1.jar \
  -input input_dirs \
  -output output_dir \
  -mapper <path/mapper.py \
  -reducer <path/reducer.py
```

Where "\" is used for line continuation for clear readability.

For Example,

./bin/hadoop jar contrib/streaming/hadoop-streaming-1.2.1.jar -input myinput -output myoutput -mapper /home/expert/hadoop-1.2.1/mapper.py -reducer /home/expert/hadoop-1.2.1/reducer.py

How Streaming Works

In the above example, both the mapper and the reducer are python scripts that read the input from standard input and emit the output to standard output. The utility will create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes.

When a script is specified for mappers, each mapper task will launch the script as a separate process when the mapper is initialized. As the mapper task runs, it converts its inputs into lines and feed the lines to the standard input (STDIN) of the process. In the meantime, the mapper collects the line-oriented outputs from the standard output (STDOUT) of the process and converts each line into a key/value pair, which is collected as the output of the mapper. By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) will be the value. If there is no tab character in the line, then the entire line is considered as the key and the value is null. However, this can be customized, as per one need.

When a script is specified for reducers, each reducer task will launch the script as a separate process, then the reducer is initialized. As the reducer task runs, it converts its input key/values pairs into lines and feeds the lines to the standard input (STDIN) of the process. In the meantime, the reducer collects the line-oriented outputs from the standard output (STDOUT) of the process, converts each line into a key/value pair, which is collected as the output of the reducer. By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) is the value. However, this can be customized as per specific requirements.

Important Commands

Parameters	**Options**	**Description**
-input directory/file-name	Required	Input location for mapper.
-output directory-name	Required	Output location for reducer.
-mapper executable or script or JavaClassName	Required	Mapper executable.
-reducer executable or script or JavaClassName	Required	Reducer executable.
-file file-name	Optional	Makes the mapper, reducer, or combiner executable available locally on the compute nodes.
-inputformat JavaClassName	Optional	Class you supply should return key/value pairs of Text class. If not specified, TextInputFormat is used as the default.
-outputformat JavaClassName	Optional	Class you supply should take key/value pairs of Text class. If not specified, TextOutputformat is used as the default.

-partitioner JavaClassName	Optional	Class that determines which reduce a key is sent to.
-combiner streamingCommand or JavaClassName	Optional	Combiner executable for map output.
-cmdenv name=value	Optional	Passes the environment variable to streaming commands.
-inputreader	Optional	For backwards-compatibility: specifies a record reader class (instead of an input format class).
-verbose	Optional	Verbose output.
-lazyOutput	Optional	Creates output lazily. For example, if the output format is based on FileOutputFormat, the output file is created only on the first call to output.collect (or Context.write).
-numReduceTasks	Optional	Specifies the number of reducers.
-mapdebug	Optional	Script to call when map task fails.
-reducedebug	Optional	Script to call when reduce task fails.

Hadoop Cluster Architecture

• Cluster: Loosely/Tightly connected computers work together as a single system

• Hadoop Cluster: Storing & analysing large amount of unstructured data in distributed environment

• These clusters run on low cost on commodity computers

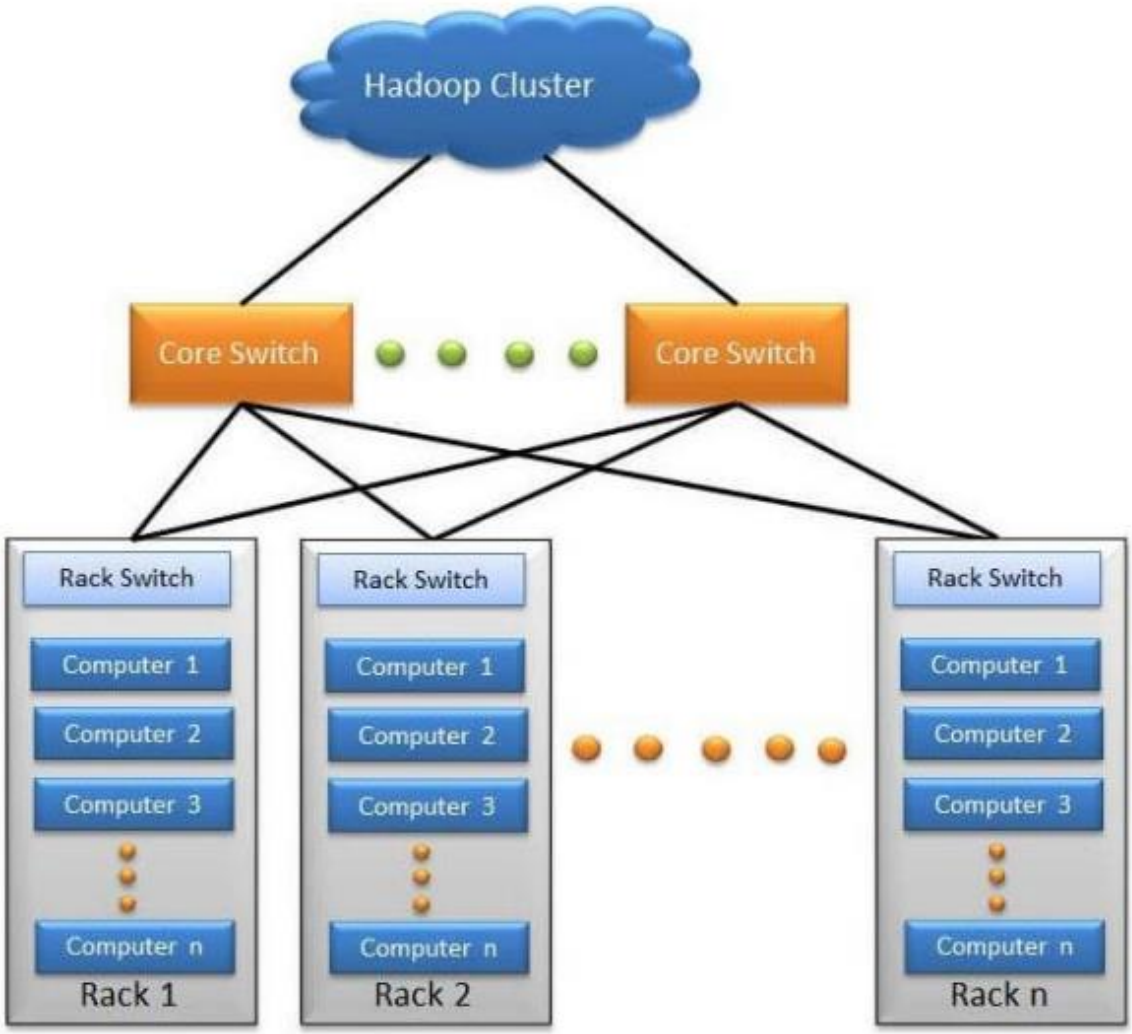


Fig. 2.27 Hadoop Cluster Architecture

Hadoop Cluster architecture as Share nothing systems

• Nodes share only the networks that connects them

• Large Hadoop clusters are arranged in several racks

• Network traffic between different nodes in the same rack is much more desirable than network traffic across the racks

• E.g. Hadoop cluster set up for 4500 nodes

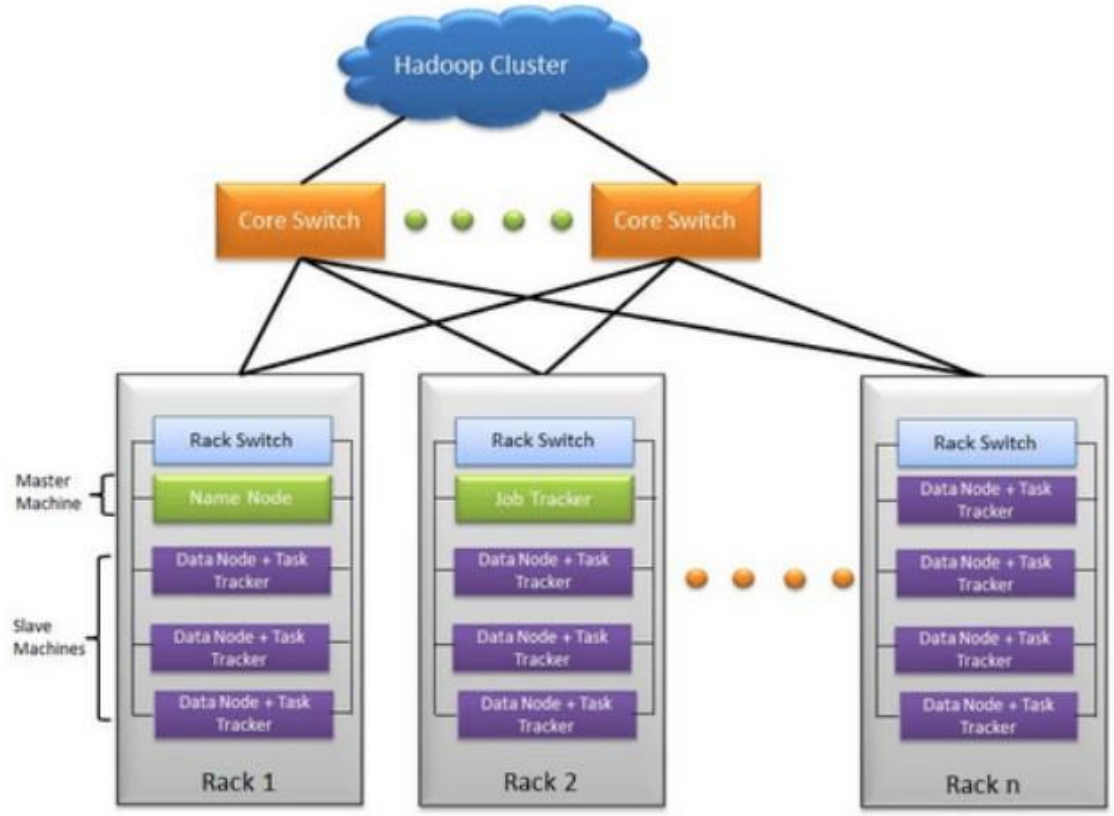


Fig. 2.28 Hadoop Cluster architecture as Share nothing systems

In this Hadoop cluster (Figure 2.28)

➢ 110 different racks

➢ Each rack has 40 slave machines

Rack Description

• At the top of each rack there is a rack switch

• Each slave machine has cables coming out from both the ends

• Cables are connected to rack switch at the top which means that top rack switch will have around 80 ports

• There are global 8 core switches

• The rack which has uplinks connected to core switches and hence connecting all other racks with uniform bandwidth forming the cluster

• In the cluster, few machines act as NameNodes and JobTracker

• They are referred to as Masters

• Masters have different configuration for more DRAM and CPU and less storage

• The majority of machine acts as data node and Task trackers are referred as slaves

• These slave nodes have lots of local disk storage and moderate amounts of DRAM and CPU

Hadoop configuration files

• Hadoop configuration is driven by two types of important configuration files:

➢ Read-only default configuration - src/core/core-default.xml, src/hdfs/hdfsdefault.xml and src/mapred/mapred-default.xml.

➢Site-specific configuration - conf/core-site.xml, conf/hdfs-site.xml and conf/mapredsite.xml.

Hadoop Cluster Modes

• Standalone mode

• Single node cluster/Pseudo distributed mode

• Multi node cluster/ Fully Distributed mode

Standalone mode

• Default mode

• HDFS is not utilized in this mode

• Local file system is used for input and output

• Used for debugging purpose

• No custom configuration is required in 3 Hadoop files

➢ mapred- site.xml

➢ core-site.xml

➢ hdfs-site.xml

• Standalone mode is much faster than pseudo distributed mode

Pseudo distributed mode

• Configuration is required in given 3 files

• Replication factor is one for HDFS

• Here, one node is used as Master node/data node/Job tracker/Task tracker

• Used for real code to test in HDFS

• Pseudo distributed cluster is a cluster where all daemons are running on one node itself

Fully Distributed mode/Multiple node cluster

• Production phase

• This mode involves the code running on an actual Hadoop cluster.

• It is mode in which you see the actual power of Hadoop, when you run your code against a very large input on 1000s of servers.

• It is always difficult to debug a MapReduce program as you have Mappers running on different machine with different piece of input.

• You can never know where the Mappers are going to run eventually.

• Also, with large inputs, it is likely that the data will be irregular in its format.

Working with Hadoop Commands

There are many more commands in "$HADOOP_HOME/bin/hadoop fs" than are demonstrated here, although these basic operations will get you started. Running ./bin/hadoop dfs with no additional arguments will list all the commands that can be run with the FsShell system. Furthermore, $HADOOP_HOME/bin/hadoop fs -help commandName will display a short usage summary for the operation in question, if you are stuck.

A table of all the operations is shown below. The following conventions are used for parameters –

"<path>" means any file or directory name.

"<path>..." means one or more file or directory names.

"<file>" means any filename.

"<src>" and "<dest>" are path names in a directed operation.

"<localSrc>" and "<localDest>" are paths as above, but on the local file system.

Sr. No	Command & Description
1	**-ls <path>** Lists the contents of the directory specified by path, showing the names, permissions, owner, size and modification date for each entry.
2	**-lsr <path>** Behaves like -ls, but recursively displays entries in all subdirectories of path.
3	**-du <path>**

	Shows disk usage, in bytes, for all the files which match path; filenames are reported with the full HDFS protocol prefix.
4	**-dus <path>** Like -du, but prints a summary of disk usage of all files/directories in the path.
5	**-mv <src><dest>** Moves the file or directory indicated by src to dest, within HDFS.
6	**-cp <src> <dest>** Copies the file or directory identified by src to dest, within HDFS.
7	**-rm <path>** Removes the file or empty directory identified by path.
8	**-rmr <path>** Removes the file or directory identified by path. Recursively deletes any child entries (i.e., files or subdirectories of path).
9	**-put <localSrc> <dest>** Copies the file or directory from the local file system identified by localSrc to dest within the DFS.
10	**-copyFromLocal <localSrc> <dest>** Identical to -put
11	**-moveFromLocal <localSrc> <dest>** Copies the file or directory from the local file system identified by localSrc to dest within HDFS, and then deletes the local copy on success.
12	**-get [-crc] <src> <localDest>** Copies the file or directory in HDFS identified by src to the local file system path identified by localDest.

13	**-getmerge <src> <localDest>** Retrieves all files that match the path src in HDFS, and copies them to a single, merged file in the local file system identified by localDest.
14	**-cat <filen-ame>** Displays the contents of filename on stdout.
15	**-copyToLocal <src> <localDest>** Identical to -get
16	**-moveToLocal <src> <localDest>** Works like -get, but deletes the HDFS copy on success.
17	**-mkdir <path>** Creates a directory named path in HDFS. Creates any parent directories in path that are missing (e.g., mkdir -p in Linux).
18	**-setrep [-R] [-w] rep <path>** Sets the target replication factor for files identified by path to rep. (The actual replication factor will move toward the target over time)
19	**-touchz <path>** Creates a file at path containing the current time as a timestamp. Fails if a file already exists at path, unless the file is already size 0.
20	**-test -[ezd] <path>** Returns 1 if path exists; has zero length; or is a directory or 0 otherwise.
21	**-stat [format] <path>** Prints information about path. Format is a string which accepts file size in blocks (%b), filename (%n), block size (%o), replication (%r), and modification date (%y, %Y).
22	**-tail [-f] <file2name>**

	Shows the last 1KB of file on stdout.
23	**-chmod [-R] mode,mode,... <path>...** Changes the file permissions associated with one or more objects identified by path.... Performs changes recursively with R. mode is a 3-digit octal mode, or {augo}+/-{rwxX}. Assumes if no scope is specified and does not apply an umask.
24	**-chown [-R] [owner][:[group]] <path>...** Sets the owning user and/or group for files or directories identified by path.... Sets owner recursively if -R is specified.
25	**-chgrp [-R] group <path>...** Sets the owning group for files or directories identified by path.... Sets group recursively if -R is specified.
26	**-help <cmd-name>** Returns usage information for one of the commands listed above. You must omit the leading '-' character in cmd.

Working with Apache Oozie

What is OOZIE?

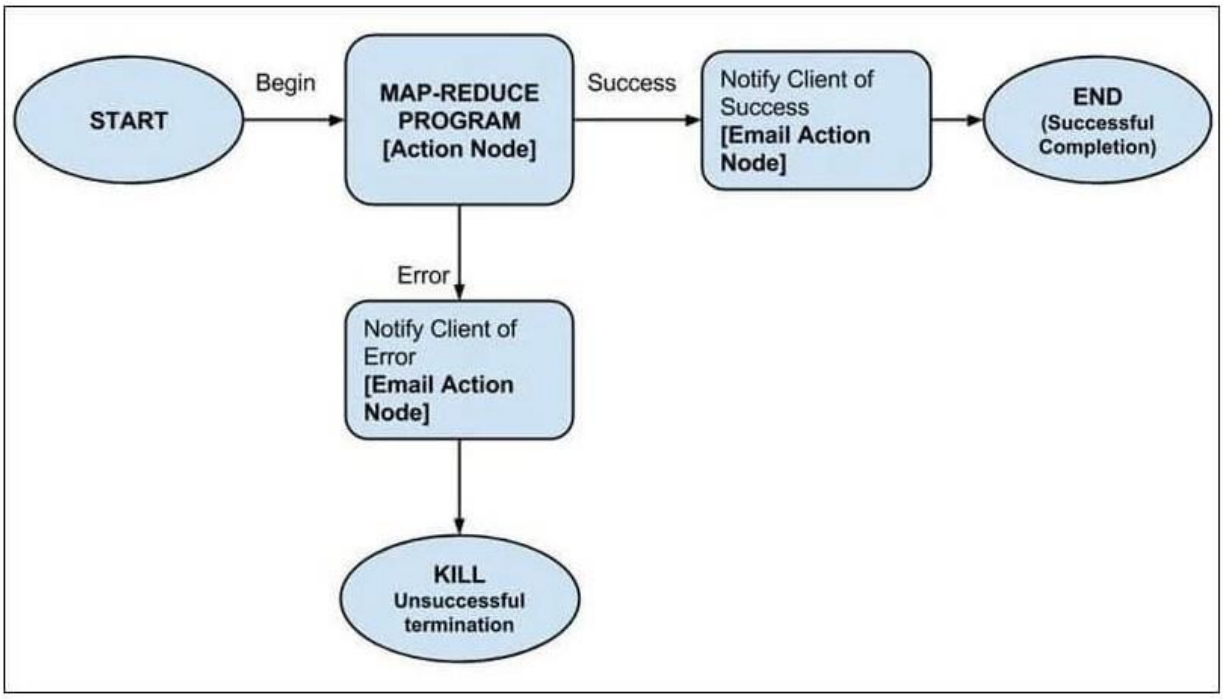
Apache Oozie is a workflow scheduler for Hadoop. It is a system which runs the workflow of dependent jobs. Here, users are permitted to create Directed Acyclic Graphs of workflows, which can be run in parallel and sequentially in Hadoop.

It consists of two parts:

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



Oozie is very much flexible, as well. One can easily start, stop, suspend and rerun jobs. Oozie makes it very easy to rerun failed workflows. One can easily understand how difficult it can be to catch up missed or failed jobs due to downtime or failure. It is even possible to skip a specific failed node.

How does OOZIE work?

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.

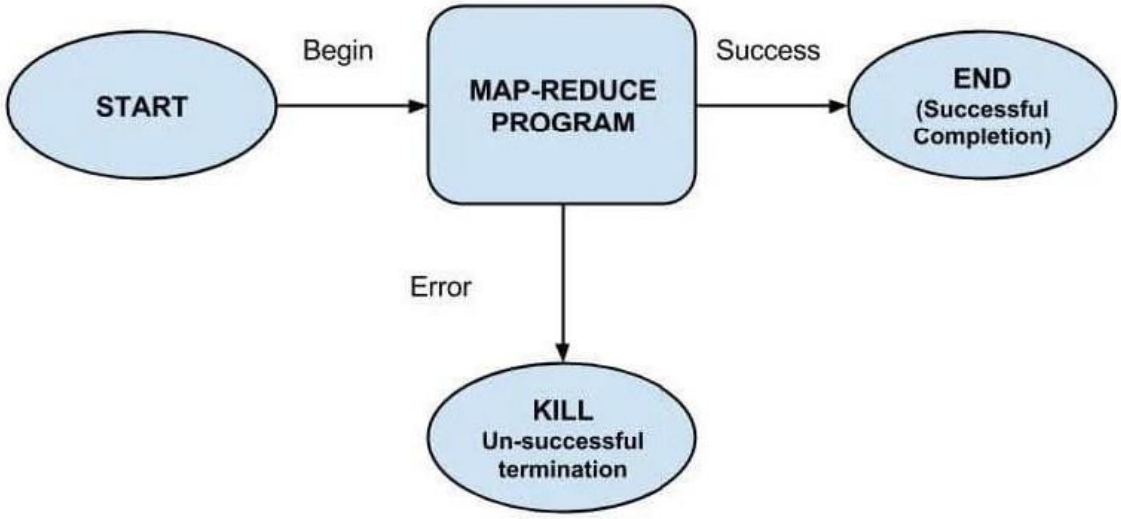
Start Node, designates the start of the workflow job.

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.

Example Workflow Diagram



Packaging and deploying an Oozie workflow application

A workflow application consists of the workflow definition and all the associated resources such as MapReduce Jar files, Pig scripts etc. Applications need to follow a simple directory structure and are deployed to HDFS so that Oozie can access them.

An example directory structure is shown below-

<name of workflow>/</name>

??? lib/

? ??? hadoop-examples.jar

??? workflow.xml

It is necessary to keep workflow.xml (a workflow definition file) in the top level directory (parent directory with workflow name). Lib directory contains Jar files containing MapReduce classes. Workflow application conforming to this layout can be built with any build tool e.g., Ant or Maven.

Such a build need to be copied to HDFS using a command, for example –

% hadoop fs -put hadoop-examples/target/<name of workflow dir> name of workflow

Steps for Running an Oozie workflow job

In this section, we will see how to run a workflow job. To run this, we will use the Oozie command-line tool (a client program which communicates with the Oozie server).

1. Export OOZIE_URL environment variable which tells the oozie command which Oozie server to use (here we're using one running locally):

% export OOZIE_URL="http://localhost:11000/oozie"

2. Run workflow job using-

% oozie job -config ch05/src/main/resources/max-temp-workflow.properties -run

The -config option refers to a local Java properties file containing definitions for the parameters in the workflow XML file, as well as oozie.wf.application.path, which tells Oozie the location of the workflow application in HDFS.

Example contents of the properties file:

nameNode=hdfs://localhost:8020

jobTracker=localhost:8021

oozie.wf.application.path=${nameNode}/user/${user.name}/<name of workflow>

3. Get the status of workflow job-

Status of workflow job can be seen using subcommand 'job' with '-info' option and specifying job id after '-info'.

e.g., % oozie job -info <job id>

Output shows status which is one of RUNNING, KILLED or SUCCEEDED.

4. Results of successful workflow execution can be seen using Hadoop command like-

% hadoop fs -cat <location of result>

Why use Oozie?

The main purpose of using Oozie is to manage different type of jobs being processed in Hadoop system.

Dependencies between jobs are specified by a user in the form of Directed Acyclic Graphs. Oozie consumes this information and takes care of their execution in the correct order as specified in a workflow. That way user's time to manage complete workflow is saved. In addition, Oozie has a provision to specify the frequency of execution of a particular job.

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.