Analyzing Data with Hadoop:

Hadoop is a powerful open-source framework for storing and analyzing big data using clusters of computers. Here's a step-by-step guide to how it works:

1. **HDFS (Hadoop Distributed File System)**
    - ❖ Distributed storage system that handles large files across multiple machines.
    - ❖ Provides high-throughput access to application data.
2. **MapReduce**
    - ❖ Programming model for processing large datasets in parallel.
    - ❖ Divides work into independent tasks processed across the cluster.
3. **YARN (Yet Another Resource Negotiator)**
    - ❖ Manages cluster resources and job scheduling.
4. **Data Ingestion**:

- ❖ Start by loading your data into Hadoop. You can use: Apache Flume for collecting logs (e.g., from web servers) Apache Kafka for handling real-time data streams. These tools help move data into HDFS or other storage systems so it's ready for processing.
- ❖ Ensure that your data is stored in a structured format in HDFS or another suitable storage system.

5. Data Preparation:

- Preprocess and clean the data as needed. This may involve tasks such as data duplication, data normalization, and handling missing values.

6. Choose a Processing Framework:

- Select the appropriate data processing framework based on your requirements. Common choices include:
    - MapReduce: Ideal for batch processing and simple transformations.
    - Apache Spark: Suitable for batch, real-time, and iterative data processing. It offers a wide range of libraries for machine learning, graph processing, and more.

- Apache Hive: If you prefer SQL-like querying, you can use Hive for data analysis.
- Custom Code: You can write custom Java, Scala, or Python code using Hadoop APIs if necessary.

7. Data Analysis:

- ❖ Write code or queries to analyze the data using tools like MapReduce, Spark, Hive, or Pig.
- ❖ Perform tasks like filtering, grouping, aggregating, or transforming the data to get useful insights.

8. Scaling:

- ❖ Hadoop is designed for horizontal scalability. As your data and processing needs grow, you can add more nodes to your cluster to handle larger workloads.

9. Optimization:

- Optimize your code and queries for performance. Tune the configuration parameters of your Hadoop cluster, such as memory settings and resource allocation.
- Consider using data partitioning and bucketing techniques to improve query performance.

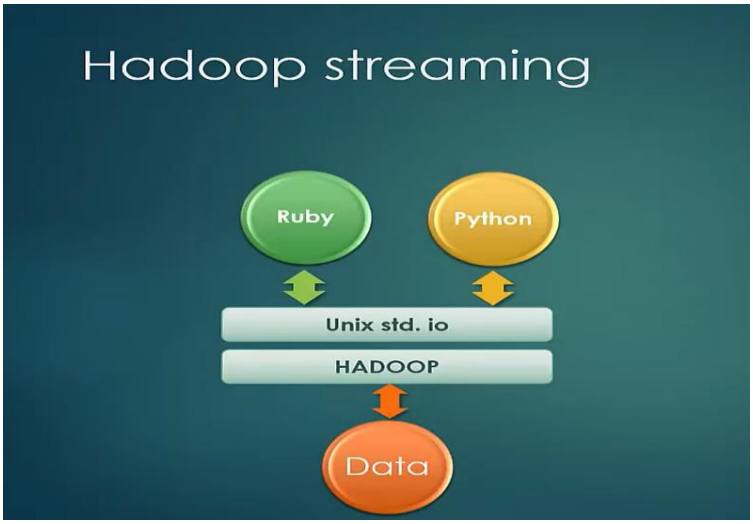
10. Data Visualization:

- Once you have obtained results from your analysis, you can use data visualization tools like Apache Zeppelin, Apache Superset, or external tools like Tableau and Power BI to create meaningful visualizations and reports.

Hadoop Streaming

Hadoop provides an API to MapReduce that allows you to write your map and reduce functions in languages other than Java. Hadoop Streaming uses Unix standard streams as the interface between Hadoop and your program, so you can

use any language that can read standard input and write to standard output to write your MapReduce program.



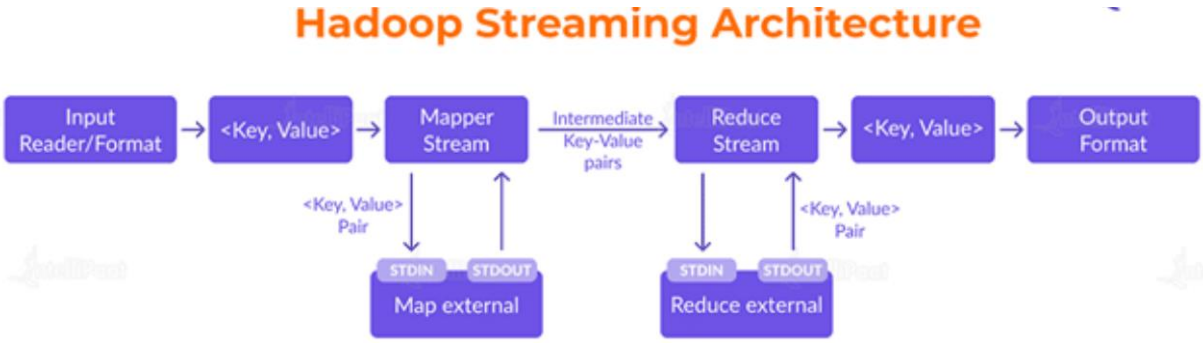
Features of Hadoop Streaming

Some of the key features associated with Hadoop Streaming are as follows:

- Hadoop Streaming is a part of the Hadoop Distribution System.
- It facilitates ease of writing Map Reduce programs and codes.
- Hadoop Streaming supports almost all types of programming languages such as Python, C++, Ruby, Perl etc.
- The entire Hadoop Streaming framework runs on Java. However, the codes might be written in different languages as mentioned in the above point.
- The Hadoop Streaming process uses Unix Streams that act as an interface between Hadoop and Map Reduce programs.
- Hadoop Streaming uses various Streaming Command Options, and the two mandatory ones are – input directoryname or filename and -output directoryname.

Hadoop Streaming architecture

Hadoop Streaming uses Unix pipes to pass data between mappers and reducers written in languages other than Java.



The six steps involved in the working of Hadoop Streaming are:

- **Step 1:** The input data is divided into **chunks** or blocks, typically 64MB to 128MB in size automatically. Each chunk of data is processed by a separate mapper.
- **Step 2:** The mapper reads the input data from standard input (stdin) and generates an intermediate key-value pair based on the logic of the mapper function which is written to standard output (stdout).
- **Step 3:** The intermediate key-value pairs are sorted and partitioned based on their keys ensuring that all values with the same key are directed to the same reducer.
- **Step 4:** The key-value pairs are passed to the reducers for further processing where each reducer receives a set of key-value pairs with the same key.
- **Step 5:** The reducer function, implemented by the developer, performs the required computations or aggregations on the data and generates the final output which is written to the standard output (stdout).
- **Step 6:** The final output generated by the reducers is stored in the specified output location in the HDFS.

The distributed nature of Hadoop enables parallel execution of mappers and reducers across a cluster of machines, providing scalability and fault tolerance. The data processing is efficiently distributed across multiple nodes, allowing for faster processing of large-scale datasets.

Advantages

1. **Flexible**:
    - ❖ You can use any language (Python, Perl, etc.), not just Java.
2. **Easy to Use**:
    - ❖ Just write scripts that read from input and write to output—no need to learn complex Hadoop APIs.
3. **Reusable Code**:
    - ❖ You can reuse your existing scripts and tools for big data processing.

Python:

Streaming supports any programming language that can read from standard input, and write to standard output, so for readers more familiar with Python, here's the same example again. The map script is in Example 2-10, and the reduce script is in Example 2-11.

Example 2-10. Map function for maximum temperature in Python

```
#!/usr/bin/env python
import re
import sys
for line in sys.stdin:     //Read each line from input
 val = line.strip()            // Remove leading/trailing spaces
 (year, temp, q) = (val[15:19], val[87:92], val[92:93])   //  Extract year,
temperature, and quality from fixed positions
```

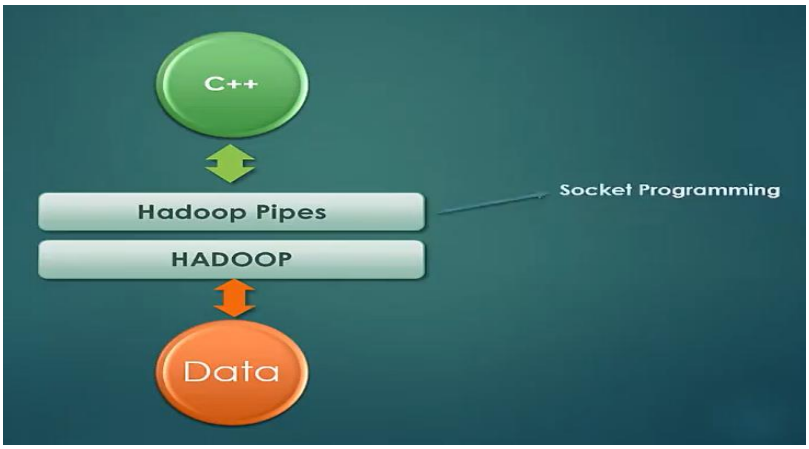
```
if (temp != "+9999" and re.match("[01459]", q)): // Check if temp is valid and quality is acceptable
    print "%s\t%s" % (year, temp) //Output year and temperature
```

Example 2-11. Reduce function for maximum temperature in Python

```
#!/usr/bin/env python
import sys
1.(last_key, max_val) = (None, 0)          // Initialize previous key and max value
2.for line in sys.stdin:                    // Read each line from input
 3.(key, val) = line.strip().split("\t")    // Split key and value
4. if last_key and last_key != key:  // If a new key is found, print result for the old key
  5. print "%s\t%s" % (last_key, max_val)  //Print the maximum value found for the last key.
  6.  (last_key, max_val) = (key, int(val))
 else:
  7. (last_key, max_val) = (key, max(max_val, int(val)))
if last_key:
8. print "%s\t%s" % (last_key, max_val)     // Print the last key's result
```

Hadoop Pipes:

- Hadoop Pipes is a way to write MapReduce programs in C++.
- Unlike Hadoop Streaming (which uses standard input/output), Pipes uses sockets to communicate between Hadoop and your C++ code.
- It does not use JNI (Java Native Interface), making it more efficient for C++ integration.
- Pipes is best for C++ users who want better connection with Hadoop.



Hadoop and Its Ecosystem

- 2002: Doug Cutting and Mike Cafarella started Apache Nutch, an open-source web search engine.
- 2003–2004: They read Google's papers on Google File System (GFS) and MapReduce, which showed how to store and process large data.
- Inspired, they added these ideas into Nutch, but it couldn't scale well beyond 40 machines.
- 2006: Doug Cutting, working at Yahoo!, split off the storage and processing parts into a new project called Hadoop. The name "Hadoop" came from his son's toy elephant.

Hadoop allows for:

- Distributed storage using HDFS (Hadoop Distributed File System)
- Distributed processing using MapReduce

It is designed to:

- Scale from a single machine to thousands
- Handle hardware failures automatically
- Work with structured, semi-structured, and unstructured data

THE ARCHITECTURE OF BIG DATA: HADOOP ECOSYSTEM AND ARCHITECTURE

The core of Apache Hadoop consists of:

1. A storage part, known as Hadoop Distributed File System (HDFS), and

2. A processing part called MapReduce.

Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed. This approach takes advantage of data locality- nodes manipulating the data they have access to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

Thus, base Apache Hadoop framework is composed of the following modules:

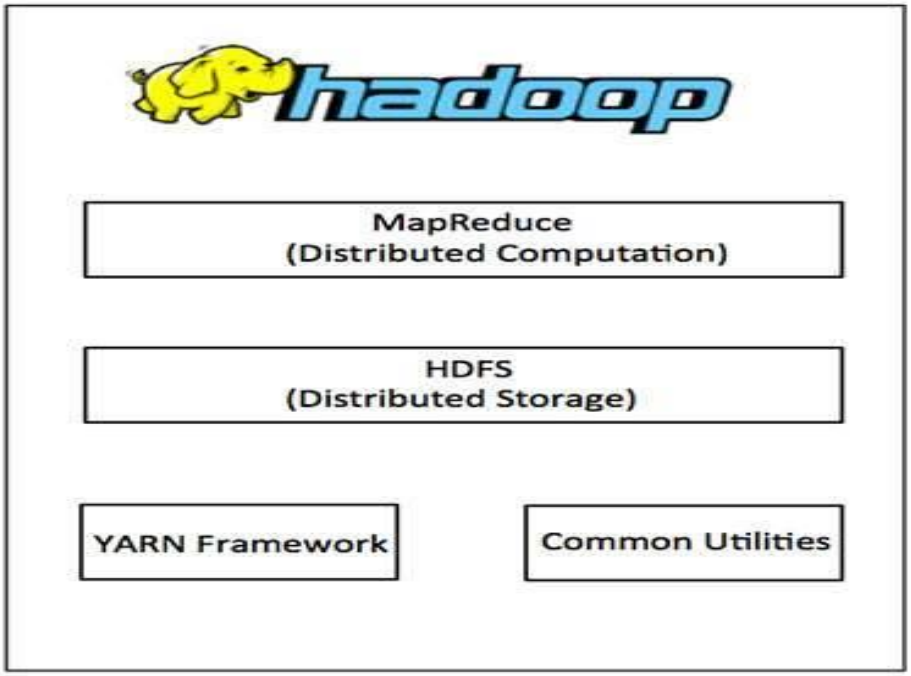
1. Hadoop Common - Hadoop Common includes the core Java libraries and utilities needed by other Hadoop modules. It provides basic functions like file system and operating system access, and includes the scripts and files required to start Hadoop.

2. Hadoop Distributed File System (HDFS) - a distributed filesystem that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;

3. Hadoop YARN is a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications.

4. Hadoop MapReduce (either MapReduce/MR, or YARN/MR, - an implementation of the MapReduce programming model for large scale data processing.

Fig. 2.1 depicts these four components available in Hadoop framework.



Since 2012, the term Hadoop has come to refer not just to the base modules above, but also to the ecosystem, or collection of additional software packages that can be installed on top of or alongside Hadoop, such as Apache Pig, Apache Hive, Apache HBase, Apache Phoenix, Apache ZooKeeper, Cloudera Impala, Apache Flume, Apache Sqoop, Apache Oozie, Apache Storm Apache Spark etc.

There are a number of supporting projects that leverage HDFS and MapReduce as detailed below.

1. MapReduce: This is the programming model for Hadoop. There are two phases, not surprisingly called Map and Reduce. To impress your friends tell them there is a shuffle-sort between the Map and Reduce phase. The JobTracker manages the 4000+ components of your MapReduce job. The TaskTrackers take orders from the JobTracker. If you like Java then code in Java. If you like SQL or other non-Java languages you are still in luck, you can use a utility called Hadoop Streaming.

2. HDFS: If you want 4000+ computers to work on your data, then you'd better spread your data across 4000+ computers. HDFS does this for you. HDFS has a

few moving parts. The Datanodes store your data, and the Namenode keeps track of where stuff is stored. There are other pieces, but you have enough to get started.

3. Jaql: A language query designed for use with data in a JSON format and designed to work with semi-structured data. It has High-level queries system that results in low- level MapReduce jobs.

4. Hive: A data warehouse system for Hadoop that uses an SQL-like language to access its data. If you like SQL, you can write SQL and have Hive convert it to a MapReduce job although you don't get a full ANSI-SQL environment, but you do get 4000 notes and multi-Petabyte scalability. Hue gives you a browser-based graphical interface to do your Hive work.

5. Pig: It uses a language called Pig Latin. The names may sound funny, but Pig helps you process large data quickly and reliably at a low cost.

6. Sqoop: Provides bi-directional data transfer between Hadoop and your favorite relational database. Transfers bulk data between Hadoop and structured datastores.

7. Flume: *Flume is a distributed service used to collect, aggregate, and transfer large volumes of log data from various sources (like servers, applications) to Hadoop's HDFS.* It is especially useful for real-time log data ingestion.

8. Oozie: Manages Hadoop workflow. This doesn't replace your scheduler or BPM (Business Process Management.) tooling, but it does provide if-then-else branching and control within your Hadoop jobs.

9. HBase: HBase is a NoSQL database built on top of Hadoop's HDFS. Stores large amounts of data in a non-relational (NoSQL) format. Designed for real-time read/write access to huge datasets. Works well when you need to store billions of rows and millions of columns.

10. Mahout: Machine learning for Hadoop. Used for predictive analytics and other advanced analysis.

12. Zookeeper: A centralized system for maintaining configuration information, Naming and providing distributed synchronization and group services. Used to

manage synchronization for the cluster. You won't be working much with Zookeeper, but it is working hard for you.

13. Apache Giraph : Apache Giraph is a powerful tool for analyzing large networks, such as social media connections(i.e Facebook, Yahoo, and Twitter), and is used by big tech companies for graph-based data processing.

14. Apache Tez is focused on improving the performance when working with graphs. This makes the development easier and reduces the number of MapReduce jobs that are executed underneath it significantly. Apache Tez highly increases the performance against typical MapReduce queries and optimizes the resource management.

Hadoop

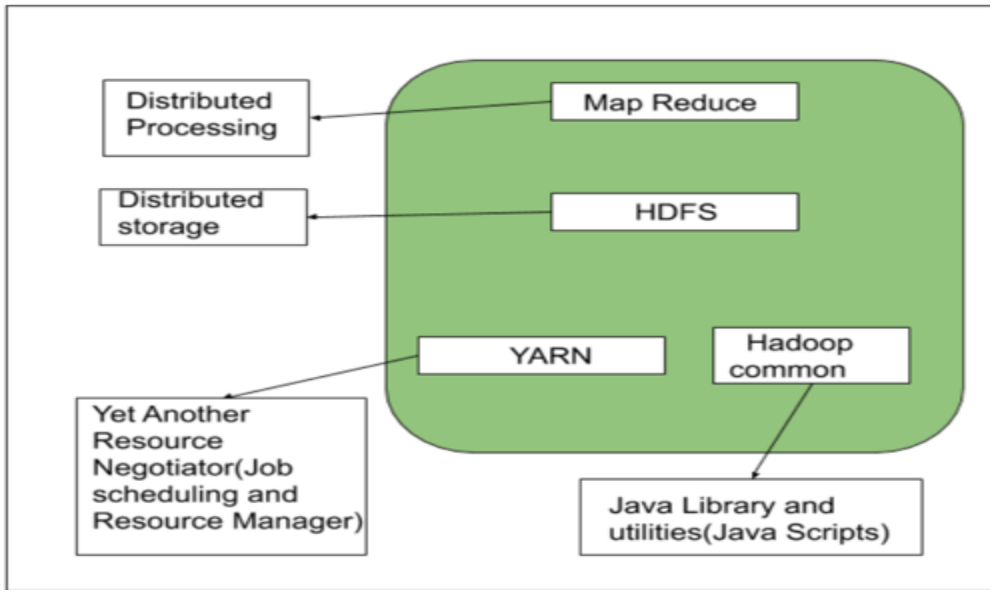
What is Hadoop?

- A software framework that supports data-intensive (i.e involve processing, storing, or analyzing very large amounts of data) distributed applications.
- It enables applications to work with thousands of nodes and petabytes of data.
- Hadoop was inspired by Google's MapReduce and Google File System (GFS).
- Yahoo! has been the largest contributor to the project, and uses Hadoop extensively across its businesses.

Hadoop – Architecture

As we all know Hadoop is a framework written in Java that utilizes a large cluster of commodity hardware to maintain and store big size data. Hadoop works on MapReduce Programming Algorithm that was introduced by Google. Today lots of Big Brand Companies are using Hadoop in their Organization to deal with big data, eg. Facebook, Yahoo, Netflix, eBay, etc. The Hadoop Architecture Mainly consists of 4 components.

- MapReduce
- HDFS(Hadoop Distributed File System)
- YARN(Yet Another Resource Negotiator)
- Common Utilities or Hadoop Common



Let's understand the role of each one of this component in detail.

1. MapReduce:
2. *HDFS*

HDFS(Hadoop Distributed File System) is utilized for storage permission. It is mainly designed for working on commodity Hardware devices(inexpensive devices), working on a distributed file system design. HDFS is designed in such a way that it believes more in storing the data in a large chunk of blocks rather than storing small data blocks.

HDFS in Hadoop provides Fault-tolerance and High availability to the storage layer and the other devices present in that Hadoop cluster. Data storage Nodes in HDFS.

- NameNode(Master)
- DataNode(Slave)

NameNode:

- ❖ NameNode is the master in a Hadoop cluster that manages and controls all the DataNodes (slaves).
- ❖ It stores metadata (data about data), such as file names, file sizes, and where each block of a file is stored.
- ❖ Metadata also includes block IDs and locations, helping NameNode find the nearest DataNode for faster access.

- NameNode gives instructions to DataNodes to create, delete, or replicate data blocks.
- The NameNode is like a traffic controller – it doesn't carry the data itself, but it tells the DataNodes what to do and where to go for faster and safer data handling.

DataNode:

- DataNodes are slaves in a Hadoop cluster that store the data (file blocks).
- A Hadoop cluster can have 1 to 500+ DataNodes — more DataNodes mean more storage capacity.
- DataNodes should have large storage space to hold many file blocks efficiently.
- NameNode manages the DataNodes, telling them what to do (store, delete, replicate data).
- In Hadoop 2.0 and later, older components like JobTracker and TaskTracker were replaced by ResourceManager (i.e Responsible for global resource management and job scheduling.) and NodeManager (i.e It reports node health and resource availability to the RM.) under the new YARN architecture.

3. YARN(Yet Another Resource Negotiator)

YARN is a Framework on which MapReduce works. YARN performs 2 operations that are Job scheduling and Resource Management. The Purpose of Job schedular is to divide a big task into small jobs so that each job can be assigned to various slaves in a Hadoop cluster and Processing can be Maximized. Job Scheduler also keeps track of which job is important, which job has more priority, dependencies between the jobs and all the other information like job timing, etc. The Resource Manager manages all the resources available in a Hadoop cluster for running jobs.

Features of YARN

- **Multi-Tenancy**: Supports running multiple applications or workloads on the same cluster at the same time.
- **Scalability**: Can handle large clusters with thousands of nodes and scale efficiently.
- **Compatibility**: Works with existing Hadoop applications and frameworks, ensuring easy integration.

4. Hadoop common or Common Utilities

- Hadoop Common provides essential Java libraries and scripts used by all components in a Hadoop cluster.
- These utilities are required by HDFS, YARN, and MapReduce for the cluster to run smoothly.
- Hadoop Common ensures the cluster can handle hardware failures by automatically solving issues through the Hadoop framework.
- It acts as the foundation for all other Hadoop components, making sure they work together efficiently.

Hadoop Daemons

Hadoop consist of five daemons.

  - NameNode
  - DataNode
  - Secondary nameNode
  - Job tracker
  - Task tracker

- "Running Hadoop" means running a set of daemons, or resident programs, on the different servers in your network.
- These daemons have specific roles; some exist only on one server, some exist across multiple servers.

NameNode

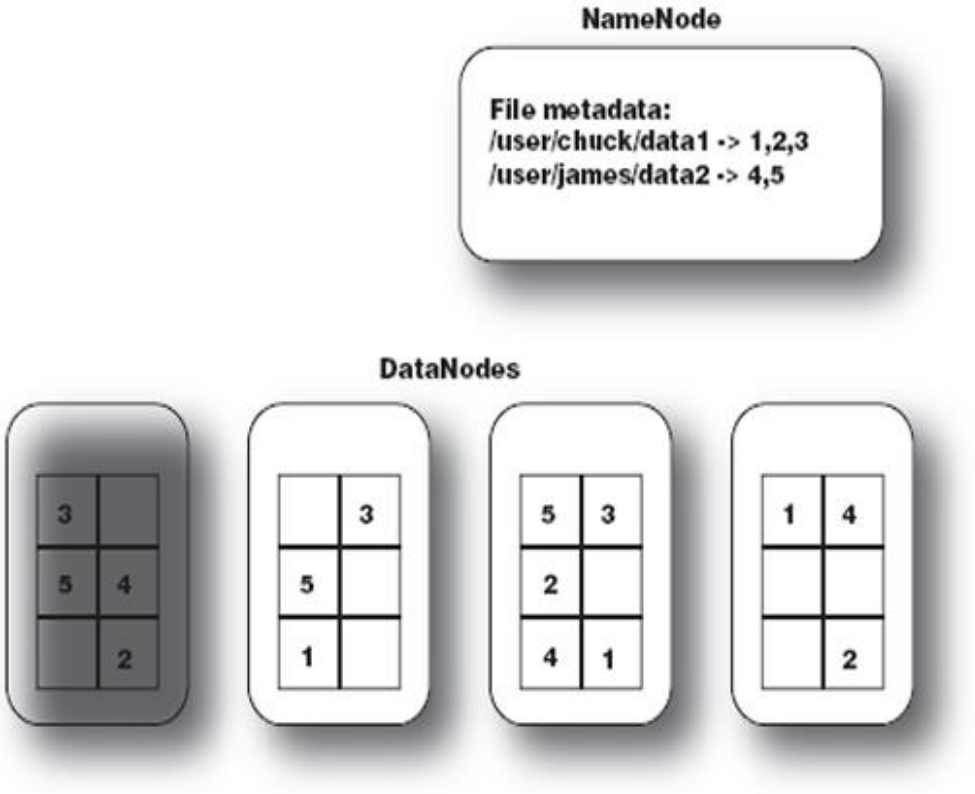
- Hadoop employs a master/slave architecture for both distributed storage and distributed computation.
- The distributed storage system is called the Hadoop File System, or HDFS.
- The NameNode is the master of HDFS that directs the slave DataNode daemons to perform the low-level I/O tasks.
- The NameNode is the bookkeeper of HDFS; it keeps track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed file system.

- The function of the NameNode is memory and I/O intensive. As such, the server hosting the NameNode typically doesn't store any user data or perform any computations for a MapReduce program to lower the workload on the machine.
- it's a single point of failure of your Hadoop cluster.
- For any of the other daemons, if their host nodes fail for software or hardware reasons, the Hadoop cluster will likely continue to function smoothly or you can quickly restart it. Not so for the NameNode.

DataNode

- Each slave machine in cluster host a Data Node daemon to perform work of the distributed file system, reading and writing HDFS blocks to actual files on the local file system.
- Sends heartbeats to the NameNode to signal it's alive.
- Performs read/write operations as per client or NameNode instructions.
- Each file is split into blocks, and blocks are replicated on multiple DataNodes for fault tolerance.
- Read or write a HDFS file, the file is broken into blocks and the NameNode will tell your client which Data Node each block resides in.
- Job communicates directly with the DataNode daemons to process the local files corresponding to the blocks.
- Furthermore, a DataNode may communicate with other DataNodes to replicate its data blocks for redundancy.

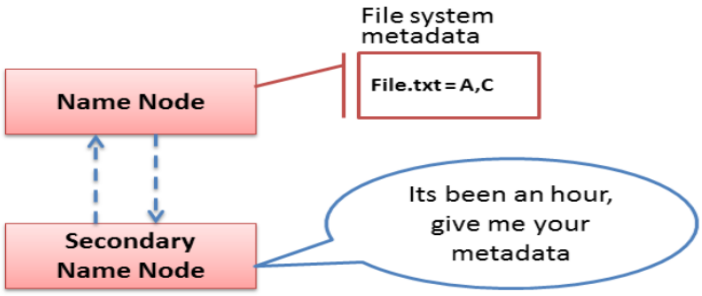
DataNode



DataNode

- Upon initialization, each of the DataNodes informs the NameNode of the blocks it's currently storing.
- After this mapping is complete, the DataNodes continually poll the NameNode to provide information regarding local changes as well as receive instructions to create, move, or delete blocks from the local disk.

Secondary NameNode



- The Secondary NameNode (SNN) is an assistant daemon for monitoring the state of the cluster HDFS.
- The SNN differs from the NameNode, it doesn't receive or record any real-time changes to HDFS.
- As mentioned earlier, the NameNode is a single point of failure for a Hadoop cluster, and the SNN snapshots help minimize the downtime and loss of data.
- If the NameNode fails, a person must manually set up the Secondary NameNode to take over.
- You cannot make the secondary namenode as the primary namenode

Job Tracker

- The JobTracker daemon is the liaison (mediator) between your application and Hadoop.
- After you submit your code, the JobTracker decides how to run it: it picks the files to process, assigns tasks to nodes, and keeps track of everything while it runs.
- If a task fails, the JobTracker will try to run it again on the same or a different node, up to a set number of times.
- It usually runs on a server as the main node of the cluster.

Tasktracker

- Task Trackers manage the execution of individual tasks on each slave node.
- Each Task Tracker is responsible for executing the individual tasks that the Job Tracker assigns.
- The TaskTracker sends regular heartbeat messages to the JobTracker.

Hadoop I/O:

Hadoop provides basic tools for input and output (I/O).

- Some tools, like data integrity and compression, are general techniques used in many systems.
- These are especially important when handling very large datasets (in terabytes).
- Other tools are specific to Hadoop, like: Serialization frameworks (for converting data into a format for storage or transmission).
- These tools help in building reliable and efficient distributed systems.

Data Integrity:

Hadoop ensures that data is not lost or changed during storage or processing. However, when working with large amounts of data, there's a higher risk of errors during disk or network operations. To detect errors, Hadoop uses checksums:
A checksum is a small code created from the data when it first enters the system. When the data is moved or used later, a new checksum is created and compared to the original. If they don't match, the data is considered corrupted.

Note: This method only detects errors it cannot fix them.
That's why it's better to use reliable hardware, like ECC (Error-Correcting Code) memory, which can detect and correct some types of errors.

Hadoop commonly uses CRC (Cyclic Redundancy Check)-32, which creates a 32-bit checksum to check for data errors.

Introduction of compression:

- In Hadoop, compression is an essential technique used in the I/O (Input/Output) operations to reduce the size of data, which improves storage efficiency and speeds up data transfer across the network.

- Since Hadoop deals with very large datasets, compressing data helps reduce the amount of space required to store it and decreases the time required for data processing by lowering disk I/O and network traffic.

Why Compression Is Important in Hadoop

i. Reduced Storage Costs
ii. Improved Data Transfer Speed
iii. Faster Processing
iv. Optimized Resource Usage

Types of Compression techniques in Hadoop

i. Gzip (GNU zip)
ii. Bzip2
iii. Snappy
iv. LZO (Lempel-Ziv-Oberhumer)

Splittable vs. Non-Splittable Compression Formats

- Splittability refers to the ability to split a compressed file into chunks for parallel processing, which is a key aspect of how Hadoop processes large datasets in a distributed manner.
- Splittable Compression Formats (e.g., Bzip2, LZO with indexing) allow large files to be processed in parallel, making Hadoop's MapReduce framework more efficient.

- Non-Splittable Compression Formats (e.g., Gzip, Snappy) don't allow splitting, which can be a bottleneck for large files. This means that the entire compressed file has to be handled by a single mapper, reducing parallelism.

Serialization

- Serialization in Hadoop I/O refers to the process of converting data objects (like records, values, or structures) into a stream of bytes that can be efficiently stored or transmitted over a network.

- In Hadoop, serialization is critical because it allows data to be written to and read from the Hadoop Distributed File System (HDFS) and enables communication between nodes during distributed processing tasks like MapReduce.

Serialization is Used in Hadoop:

1. Interprocess Communication: Hadoop uses RPC (Remote Procedure Call) for communication between nodes (like between client and NameNode).

2. Persistent Storage: Hadoop stores data in file-based formats like SequenceFile, MapFile, and Avro. Data in these files is stored in a serialized format for efficient I/O and space saving.

Good Serialization Format Should Be:

1.Compact

Uses minimal space, saving network bandwidth.

2.Fast

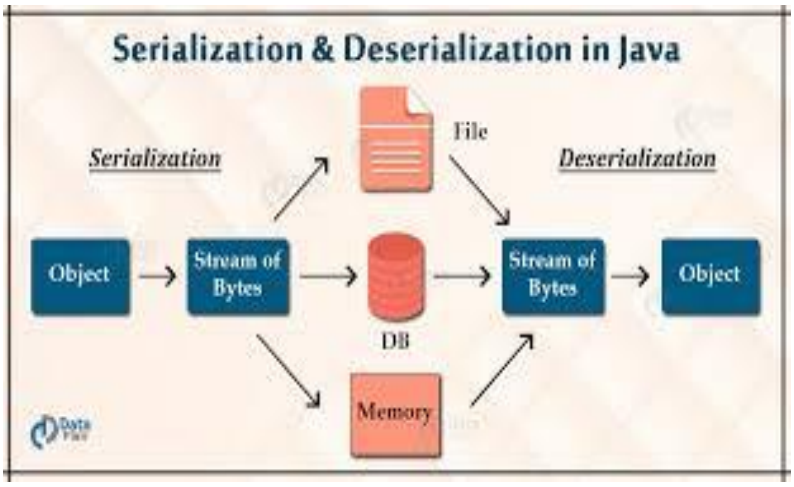
Adds very little delay during data transmission.

3.Extensible

Easy to update new versions can work with old ones (e.g., adding new fields without breaking old clients).

3.Interoperable

Can be used across different programming languages (e.g., Java client and Python server).



Example:

```
import java.io.*;  // Import necessary classes

class Student implements Serializable {
   int stuRollNo;
   String stuName;
}

public class Main {
public static void main(String[] args) {   // Main method entry point of the program
Student s1 = new Student();
s1.stuRollNo= 5;
s1.stuName = "Pradeep";
String fileName = "D:\\Automation TestData\\TestFile.txt";
try        // Try block to handle file operations (serialization)
 {
FileOutputStream fos = new FileOutputStream(fileName);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(s1);              // Write (serialize) the Student object to the file
oos.close();                      // Close the ObjectOutputStream to release resources
fos.close();
System.out.println("Object saved in file.");
}
catch (IOException e) {    // Catches input/output errors that occur during file operations
          e.printStackTrace();  //Prints detailed error information to help debug the exception
       }
   }
}
```

De-Serialization:

- ❖ Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.
- ❖ The ObjectInputStream class contains readObject() method for deserializing an object.

Example:

```
package com.org.java;       // Declares the package name
import java.io.FileInputStream;  // For reading bytes from a file
import java.io.ObjectInputStream;     // For reading serialized objects from a file
import java.io.IOException;          // For handling input/output exceptions
import java.io.ClassNotFoundException; // For handling class loading issues
public class DerializationDemo
{
public static void main(String[] args)
 {
String fileName = "D:\\Automation TestData\\TestFile.txt";
try
{
FileInputStream fis = new FileInputStream(fileName); // Create a stream to read raw bytes from the file
ObjectInputStream ois = new ObjectInputStream(fis);
Student obj = (Student)ois.readObject();
System.out.println(obj.stuRollNo);
System.out.println(obj.stuName);
ois.close();
fis.close();
}
// Handle input/output errors (e.g., file not found)
       catch (IOException e) {
          e.printStackTrace();  // Print error details to console
       }
       // Handle error if Student class is not found during deserialization
       catch (ClassNotFoundException e) {
          e.printStackTrace();  // Print error details to console
       }
   }
}
```

Types of Serialization Frameworks

1. Writable (Hadoop-Specific Serialization)

- ❖ Writable is a custom data type used to represent common data types like integers, strings, and more. For example, you may have types like IntWritable, Text, LongWritable, etc.
- ❖ Very fast and compact.
- ❖ It is commonly used in MapReduce programs because of its performance benefits.
- ❖ Handled by the WritableSerialization class in Hadoop, which is optimized for high throughput and low overhead.

Example:

- ❖ IntWritable (represents an integer)
- ❖ Text (represents a string)

2. Java Serialization

- Uses Java's built-in Serializable interface.
- Allows us to use Java types like Integer, String easily.
- Slower and bigger than Writable.
- Handled by JavaSerialization class.

3.Binary/Compact Serialization:

- Binary serialization refers to converting objects into a binary format (a compact byte stream). It's efficient in terms of speed and memory usage, making it great for large-scale systems where performance is important.

i) Apache Thrift

- Created by Facebook.
- It allows efficient data serialization and is used in Hadoop when cross-language data exchange and high-performance network communication are needed.
- Used inside Hadoop for network communication (RPC).

ii) Google Protocol Buffers (Protobuf**)**

- Created by Google.
- Google Protocol Buffers is another serialization framework used in Hadoop for structured data. Like Avro, Protobuf is schema-based and language-neutral.
- Very compact and fast.
- Also used for RPC and internal communication in Hadoop.

Key Characteristics

- ❖ Compact binary format: Data serialized with Protobuf is extremely compact.
- ❖ Schema-based: Like Avro, Protobuf uses a schema to describe the structure of the data.
- ❖ Language-neutral: Supports multiple programming languages (e.g., Java, C++,Python).ABALEUNIVERN

iii) Avro (Best for Hadoop)

- Specifically designed for big data processing.
- Avro is a popular serialization framework used in Hadoop for working with complex data types.
- It stores data in a compact binary format and also includes a schema with the data, which makes it self-describing.
- Supports evolution schema (e.g., you can add or remove fields later).
- Works well with Hive, Pig, Kafka, etc.

File-Based Data Structures

Sometimes, we need better ways to store data. In Hadoop MapReduce, saving many small pieces of data in separate files is not efficient. To solve this, Hadoop uses special file formats that group data in a smarter and more scalable way.

1.Sequence File

- ❖ A SequenceFile is a special binary file format used in Hadoop to store data as key-value pairs.
- ❖ It is mostly used when you want to save large amounts of data efficiently, especially during MapReduce jobs.

Key Features:

- Stores data in (key, value) format
- Data is stored in a binary (compact) format
- Both key and value must be Writable types (e.g., Text, IntWritable, etc.)
- Designed for fast and efficient I/O

- Often used for intermediate data storage in MapReduce

Example:

Imagine you are processing website logs and want to store:

- User ID → Last login time

You can store:

(1001, "2024-01-01 09:30")

(1002, "2024-01-01 10:45")

This would be stored in a SequenceFile with keys as IntWritable and values as Text.

2.MapFile

A MapFile is a sorted SequenceFile with an index to permit lookups by key. MapFile can be thought of as a persistent form of java.util.Map (although it doesn't implement this interface), which is able to grow beyond the size of a Map that is kept in memory.

MapFile variants :

1. SetFile

- ❖ Used to store a set of unique keys.
- ❖ The keys are stored in sorted order.
- ❖ This means when you insert a new key, it must be placed at the correct position to maintain the sorting order.
- ❖ When you only care about storing keys (not values) and need to check if a key exists in a sorted collection.

Example: Imagine you're storing a list of user IDs and need to check if a specific user exists. SetFile is efficient for that.

2. ArrayFile

- Stores data like an array, where each item has an index (0, 1, 2…).
- Useful when you want to access data by position.
- The value is a Writable object associated with each index.
- This variant is a direct mapping of indices to values.
- When you have an ordered collection of data, where each data element can be accessed by its position (index), much like an array in programming.

Example: If you are storing a list of user names, you can use their position (index) in the list as the key, and the username as the value.

3. BloomMapFile

- A MapFile that is optimized for sparse datasets where not all possible keys are used.
- A smart version of MapFile for sparse data (where many keys are missing).
- Use a Bloom filter to quickly check if a key might be present.
- Faster because it avoids unnecessary disk lookups.
- Can sometimes say “maybe” when the key isn’t really there (false positive).
- If “maybe”, it checks the MapFile to confirm.

Example: In an application where you need to check whether a **particular word** is present in a large dictionary (but only a small subset of words are actually used), the **BloomMapFile** will quickly tell you "maybe" with minimal memory usage, and then it will perform a regular lookup if needed.