UNIT-III, Part-A

Introduction to Hadoop: Hadoop: History of Hadoop, the Hadoop Distributed File System, Components of Hadoop Analysing the Data with Hadoop, Scaling Out, Hadoop Streaming, Design of HDFS, Java interfaces to HDFS Basics

Hadoop: History of Hadoop,the Hadoop Distributed File System:

1. How node failure can be handled in Hadoop? Discuss. **[7M-R20-SET-1-July 2023] [Remembering]**
2. List and explain the important features of Hadoop? **[7M-R20-SET-2-July 2023] [Understanding]**
3. Differentiate between HDFS and GFS? **[7M-R20-SET-3-July 2023] [Create]**
4. What are the Advantages and Disadvantages of Hadoop?**[Remembering]**

Components of Hadoop Analysing the Data with Hadoop, Scaling Out, Hadoop Streaming:

5. What are components of Hadoop analysing the data with hadoop?**[Remembering]**
6. Explain about the Scaling Out and Hadoop Streaming?**[Understanding]**

Design of HDFS, Java interfaces to HDFS Basics:

7. Describe the Design of HDFS?-**[Create]**
8. Explain about the Java interface to basics of HDFS.**[Understanding]**
9. What are the Data Formats in Hadoop file systems? **[Remembering]**

UNIT-III, PART-B

Developing a Map Reduce Application, How Map Reduce Works, Anatomy of a Map Reduce Job run, Failures, Job Scheduling, Shuffle and Sort, Task execution, Map Reduce Types and Formats, Map Reduce Features Hadoop environment.

Developing a Map Reduce Application, How Map Reduce Works, Anatomy of a Map Reduce Job run:

1. Discuss the various types of map reduce & its formats. **[7M-R20-SET-1-July 2023] [Create]**
2. Explain various phases of Map Reduce job with an example. **[8M-R20-SET-1-July 2023][Understanding]**
3. Write in detail the concept of developing the Map Reduce Application? **[7M-R20-SET-2-July 2023] [Create]**

Failures, Job Scheduling, Shuffle and Sort, Task execution, Map Reduce Types and Formats:

4. Describe the anatomy of Map Reduce program and discuss various types of failures in running a Map Reduce job. **[7M-R20-SET-1-July 2023] [Create]**
5. Explain about the Map Reduce Types and Formats. **[Understanding]**

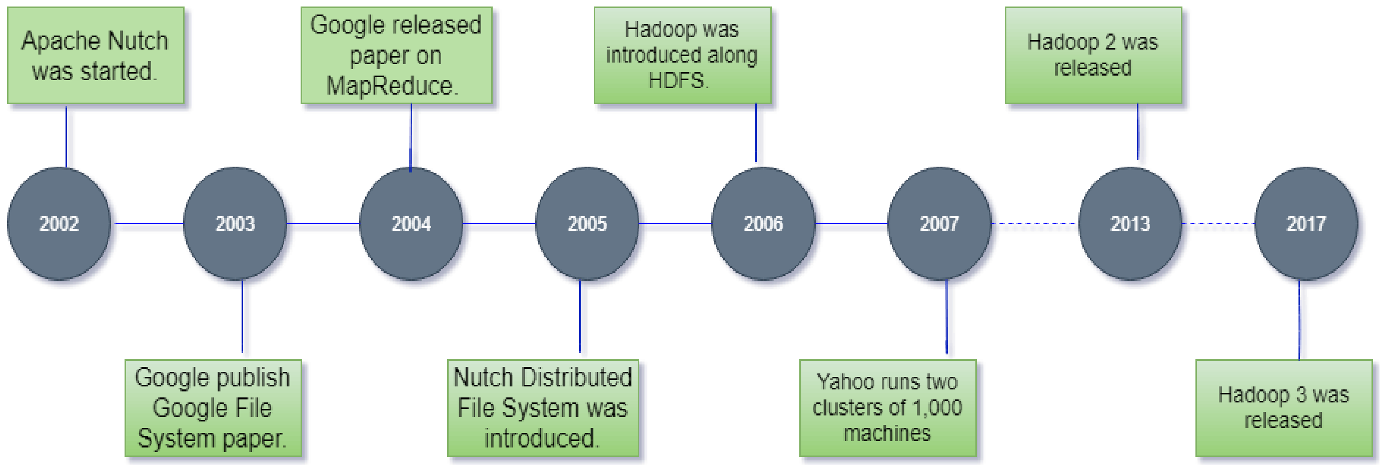
Map Reduce Features Hadoop environment:

6. Explain about the Map Reduce features of hadoop environment. **[Understanding]**

Hadoop is an open source framework from Apache and is used to store process and analyze data which are very huge in volume. Hadoop is written in Java and is not OLAP (online analytical processing). It is used for batch/offline processing. It is being used by Facebook, Yahoo, Google, Twitter, LinkedIn and many more. Moreover it can be scaled up just by adding nodes in the cluster.

History of Hadoop

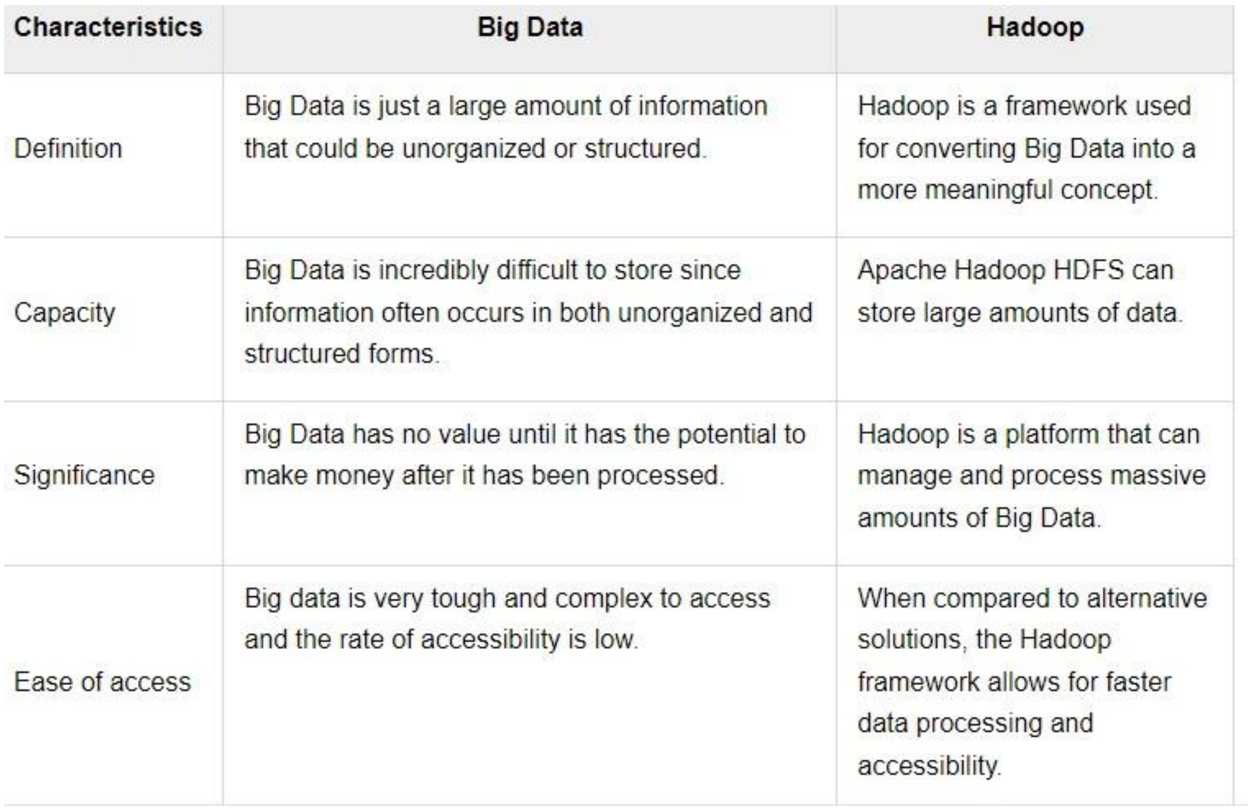
The Hadoop was started by Doug Cutting and Mike Cafarella in 2002. Its origin was the Google File System paper, published by Google.


Apache Nutch was started.
Google released paper on MapReduce.
Hadoop was introduced along HDFS.
Hadoop 2 was released
2002
2003
2004
2005
2006
2007
2013
2017
Google publish Google File System paper.
Nutch Distributed File System was introduced.
Yahoo runs two clusters of 1,000 machines
Hadoop 3 was released

- In 2002, Doug Cutting and Mike Cafarella started to work on a project, **Apache Nutch.** It is an open source web crawler software project.
- While working on Apache Nutch, they were dealing with big data. To store that data they have to spend a lot of costs which becomes the consequence of that project. This problem becomes one of the important reason for the emergence of Hadoop.
- In 2003, Google introduced a file system known as GFS (Google file system). It is a proprietary distributed file system developed to provide efficient access to data.
- In 2004, Google released a white paper on Map Reduce. This technique simplifies the data processing on large clusters.

- In 2005, Doug Cutting and Mike Cafarella introduced a new file system known as NDFS (Nutch Distributed File System). This file system also includes Map reduce.
- In 2006, Doug Cutting quit Google and joined Yahoo. On the basis of the Nutch project, Dough Cutting introduces a new project Hadoop with a file system known as HDFS (Hadoop Distributed File System). Hadoop first version 0.1.0 released in this year.
- Doug Cutting gave named his project Hadoop after his son's toy elephant.
- In 2007, Yahoo runs two clusters of 1000 machines.
- In 2008, Hadoop became the fastest system to sort 1 terabyte of data on a 900 node cluster within 209 seconds.
- In 2013, Hadoop 2.2 was released.
- In 2017, Hadoop 3.0 was released.

Differences between Big Data and Hadoop

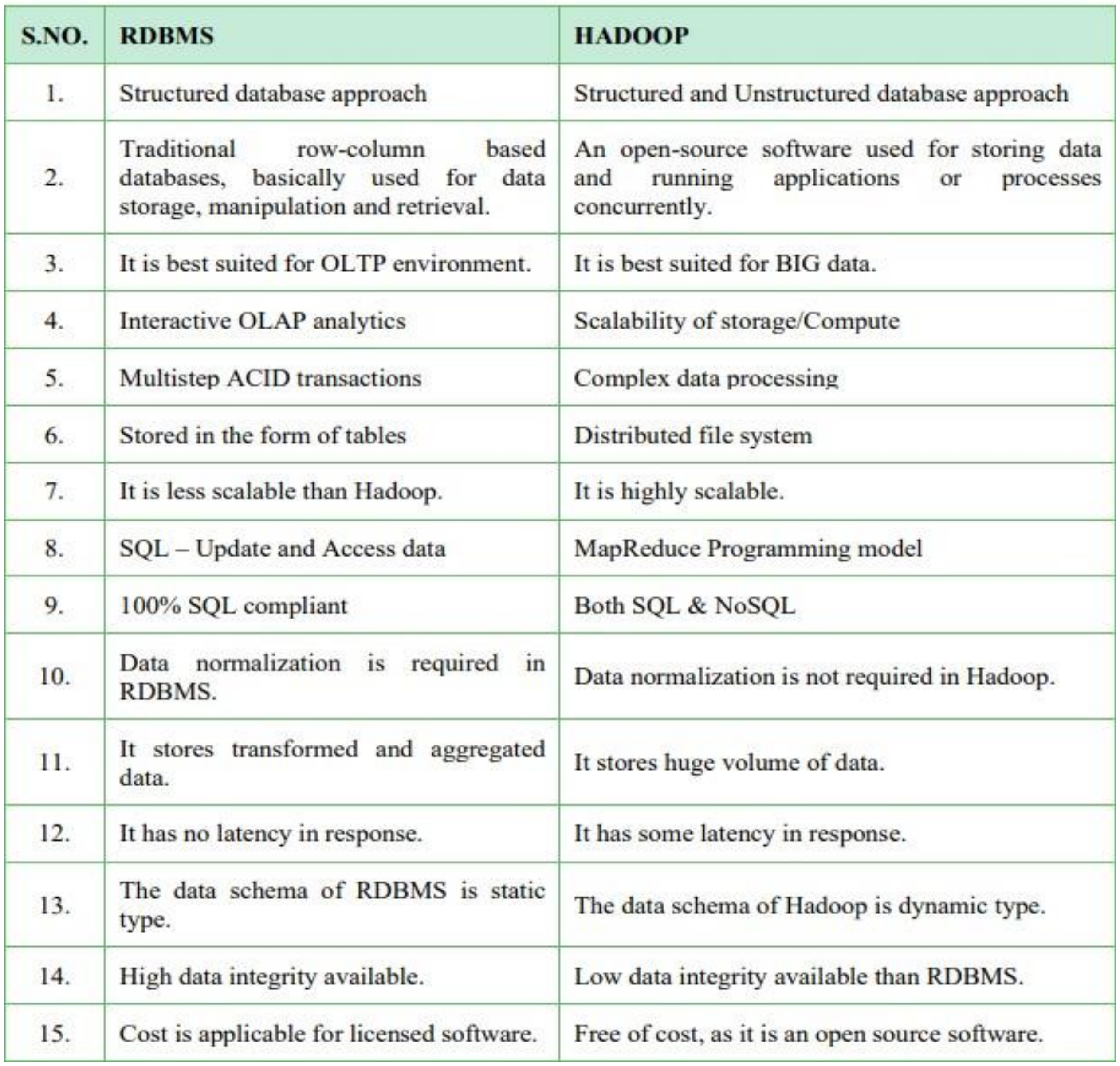


Characteristics	Big Data	Hadoop
Definition	Big Data is just a large amount of information that could be unorganized or structured.	Hadoop is a framework used for converting Big Data into a more meaningful concept.
Capacity	Big Data is incredibly difficult to store since information often occurs in both unorganized and structured forms.	Apache Hadoop HDFS can store large amounts of data.
Significance	Big Data has no value until it has the potential to make money after it has been processed.	Hadoop is a platform that can manage and process massive amounts of Big Data.
Ease of access	Big data is very tough and complex to access and the rate of accessibility is low.	When compared to alternative solutions, the Hadoop framework allows for faster data processing and accessibility.

Q) Explain the differences between Hadoop and RDBMS

Parameters	RDBMS	Hadoop
System	Relational Database Management system	Node based flat structure
Data	Suitable for structured data	Suitable for Structured, unstructured data, supports variety of formats(xml, json)
Processing	OLTP	Analytical, big data processing Hadoop clusters, node require any consistent relationships between data
Choice	When the data needs consistent relationship	Big data processing, which does not require any consistent relationships between data
Processor	Needs expensive hardware or high-end processors to store huge volumes of data	In commodity hardware less configure hardware.
Cost	Cost around \$10,000 to \$14,000 per terabytes of storage	Cost around \$4000 per terabytes of storage.

Table 2.1 RDBMS Vs Hadoop



S.NO.	RDBMS	HADOOP
1.	Structured database approach	Structured and Unstructured database approach
2.	Traditional row-column based databases, basically used for data storage, manipulation and retrieval.	An open-source software used for storing data and running applications or processes concurrently.
3.	It is best suited for OLTP environment.	It is best suited for BIG data.
4.	Interactive OLAP analytics	Scalability of storage/Compute
5.	Multistep ACID transactions	Complex data processing
6.	Stored in the form of tables	Distributed file system
7.	It is less scalable than Hadoop.	It is highly scalable.
8.	SQL – Update and Access data	MapReduce Programming model
9.	100% SQL compliant	Both SQL & NoSQL
10.	Data normalization is required in RDBMS.	Data normalization is not required in Hadoop.
11.	It stores transformed and aggregated data.	It stores huge volume of data.
12.	It has no latency in response.	It has some latency in response.
13.	The data schema of RDBMS is static type.	The data schema of Hadoop is dynamic type.
14.	High data integrity available.	Low data integrity available than RDBMS.
15.	Cost is applicable for licensed software.	Free of cost, as it is an open source software.

Hadoop is an open source framework that is meant for storage and processing of big data in a distributed manner.

It is the best solution for handling big data challenges.

Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes.

In short, Hadoop framework is capable enough to develop applications capable of running on clusters of computers and they could perform complete statistical analysis for a huge amount of data.

It runs applications on large clusters of commodity hardware and it processes thousands of terabytes of data on thousands of the nodes. Hadoop is inspired from Google's MapReduce and Google File System (GFS)papers.

The major advantage of Hadoop framework is that it provides reliability and high availability.

Some important features of Hadoop are

Open Source – Hadoop is an open source framework which means it is available free of cost. Also, the users are allowed to change the source code as per their requirements.

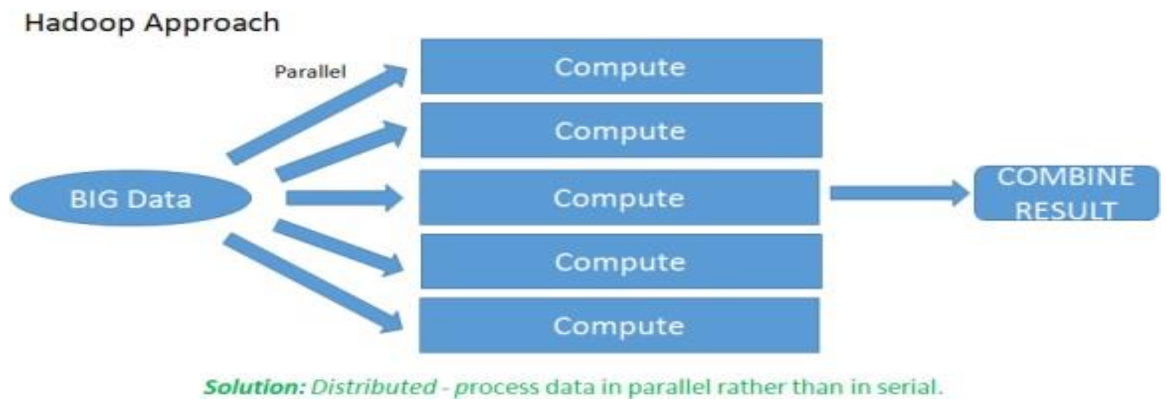
Distributed Processing – Hadoop supports distributed processing of data i.e. faster processing. The ata in Hadoop HDFS is stored in a distributed manner and MapReduce is responsible for the parallel processing of data.

Fault Tolerance – Hadoop is highly fault-tolerant. It creates three replicas for each block (default) at different nodes.

Reliability – Hadoop stores data on the cluster in a reliable manner that is independent of machine. So, the data stored in Hadoop environment is not affected by the failure of the machine

Scalability – It is compatible with the other hardware and we can easily add/remove the new hardware to the nodes.

High Availability – The data stored in Hadoop is available to access even after the hardware failure. In case of hardware failure, the data can be accessed from another node.



<u>**The core components of Hadoop are**</u> –

1. HDFS: (Hadoop Distributed File System) – HDFS is the basic storage system of Hadoop. The large data files running on a cluster of commodity hardware are stored in HDFS. It can store data in a reliable manner even when hardware fails. The key aspects of HDFS are:

a. Storage component

b. Distributes data across several nodes
c. Natively redundant.
2. Map Reduce: MapReduce is the Hadoop layer that is responsible for data processing. It writes an application to process unstructured and structured data stored in HDFS.

It is responsible for the parallel processing of high volume of data by dividing data into independent tasks. The processing is done in two phases Map and Reduce.

The Map is the first phase of processing that specifies complex logic code and the Reduce is the second phase of processing that specifies lightweight operations.

The key aspects of Map Reduce are:

a. Computational frame work
b. Splits a task across multiple nodes
c. Processes data in parallel
3.Namenode: Name node is the heart of the Hadoop system. The NameNode manages the file system namespace. It stores the metadata information of the data blocks. The Name Node also knows the location of the data blocks on the data node. If the NameNode crashes, then the entire Hadoop system goes down.
4.Secondary Namenode:The responsibility of secondary name node is to periodically copy and merge the namespace image and editlog . Incase if the name node crashes, then the namespace image stored in secondary NameNode can be used to restart the NameNode.

5.DataNode: It stores the blocks of data and retrieves them. The DataNodes also reports the blocks information to the NameNode periodically.

6.Job Tracker: Job Tracker responsibility is to schedule the client's jobs. Job tracker creates map and reduce tasks and schedules them to run on the DataNodes. Tracker also checks for any failed tasks and reschedules the failed tasks on another DataNode. Job tracker can be run on the NameNode or a separatenode.

7.Task Tracker: Task tracker runs on the DataNodes. Task trackers responsibility is to run the map or reduce tasks assigned by the NameNode and to report the status of the tasks to the NameNode

Modules of Apache Hadoop framework: There are four basic or core components:

Hadoop Common: It is a set of common utilities and libraries which handle other Hadoop modules. It makes sure that the hardware failures are managed by Hadoop cluster automatically.

Hadoop YARN: It allocates resources which in turn allow different users to execute various applications without worrying about the increased workloads.

HDFS: It is a Hadoop Distributed File System that stores data in the form of small memory blocks and distributes them across the cluster. Each data is replicated multiple times to ensure data availability. Hadoop

MapReduce: It executes tasks in a parallel fashion by distributing the data as small blocks

Use of Hadoop : There are many advantages of using Hadoop:

1. **Robust and Scalable** – We can add new nodes as needed as well modify them.

2. **Affordable and Cost Effective**–We do not need any special hardware for running Hadoop. We can just use commodity server.

3. **Adaptive and Flexible** – Hadoop is built keeping in mind that it will handle structured and unstructured data.

4. **Highly Available and Fault Tolerant** – When a node fails, the Hadoop framework automatically fails over to another node

<u>Advantages of Hadoop:</u>

1. **Scalable:** Hadoop is a highly scalable storage platform, because it can stores and distribute very large data sets across hundreds of inexpensive servers that operate in parallel.

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.

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.

4. **Speed of Processing:** 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.

<u>Hadoop Distributed File System(HDFS)</u>

HDFS: (Hadoop Distributed File System) – HDFS is the basic storage system of Hadoop. The large data files running on a cluster of commodity hardware are stored in HDFS. It can store data in a reliable manner even when hardware fails.

The key aspects of HDFS are:

- HDFS is developed by the inspiration of Google File System(GFS).
- Storage component: Stores data in hadoop
- Distributes data across several nodes: divides large file into blocks and stores in various data nodes.

Natively redundant: replicates the blocks in various data nodes.
High Throughput Access: Provides access to data blocks which are nearer to the client. Re-replicates the nodes when nodes are failed.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines.

These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

Hadoop runs code across a cluster of computers. This process includes the following core tasks that Hadoop performs −

- Data is initially divided into directories and files. Files are divided into uniform sized Blocks of 128M and 64M (preferably 128M).
- These files are then distributed across various cluster nodes for further processing.
- HDFS, being on top of the local file system, supervises the processing.
- Blocks are replicated for handling hardware failure.
- Checking that the code was executed successfully.
- Performing the sort that takes place between the map and reduce stages.
- Sending the sorted data to a certain computer.
- Writing the debugging logs for each job.

Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

HDFS (Hadoop Distributed File System) is a unique design that provides storage for *extremely large files* with streaming data access pattern and it runs on *commodity hardware*. Let's elaborate the terms:

- ***Extremely large files***: Here we are talking about the data in range of petabytes(1000 TB).
- ***Streaming Data Access Pattern***: HDFS is designed on principle of *write-once and read-many-times*. Once data is written large portions of dataset can be processed any number times.
- ***Commodity hardware:*** Hardware that is inexpensive and easily available in the market. This is one of feature which specially distinguishes HDFS from other file system.

Nodes: Master-slave nodes typically forms the HDFS cluster.

1. **NameNode(MasterNode):**

- Manages all the slave nodes and assign work to them.
- It executes file system namespace operations like opening, closing, renaming files and directories.

- It should be deployed on reliable hardware which has the high config. not on commodity hardware.

2. **DataNode(SlaveNode):**

- Actual worker nodes, who do the actual work like reading, writing, processing etc.
- They also perform creation, deletion, and replication upon instruction from the master.
- They can be deployed on commodity hardware.

3. Job Tracker

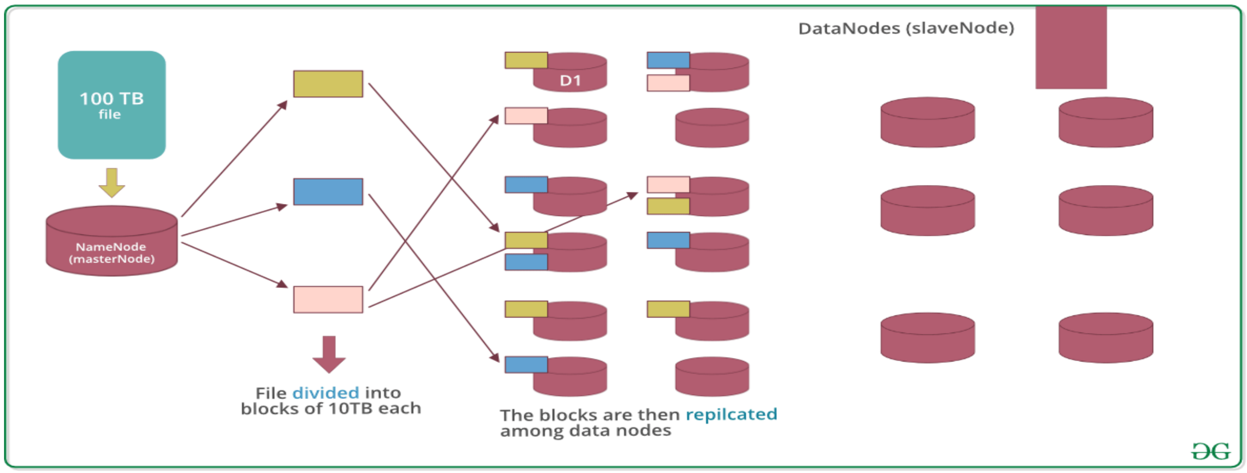
- The role of Job Tracker is to accept the MapReduce jobs from client and process the data by using NameNode.In response, NameNode provides metadata to Job Tracker.

4. Task Tracker

- It works as a slave node for Job Tracker. o It receives task and code from Job Tracker and applies that code on the file. This process can also be called as a Mapper.

5. MapReduce Layer

- The MapReduce comes into existence when the client application submits the MapReduce job to Job Tracker.
- In response, the Job Tracker sends the request to the appropriate Task Trackers. Sometimes, the TaskTracker fails or time out. In such a case, that part of the job is rescheduled.



Lets assume that 100TB file is inserted, then masternode(namenode) will first divide the file into blocks of 10TB (default size is 128 MB in Hadoop 2.x and above). Then these blocks are stored across different datanodes(slavenode). Datanodes(slavenode) replicate the blocks among themselves and the information of what blocks they contain is sent to the master.

File Read operation: The steps involved in the File Read are as follows:

1. The client opens the file that it wishes to read from by calling open() on the DFS.

2. The DFS communicates with the NameNode to get the location of data blocks. NameNode returns with the addresses of the DataNodes that the data blocks are stored on. Subsequent to this, the DFS returns an FSD to client to read from the file.

3. Client then calls read() on the stream DFSInputStream, which has addresses of DataNodes for the first few block of the file.

4. Client calls read() repeatedly to stream the data from the DataNode.

5.When the end of the block is reached, DFSInputStream closes the connection with the DataNode. It repeats the steps to find the best DataNode for the next block and subsequent blocks.

6. When the client completes the reading of the file, it calls close() on the FSInputStream to the connection

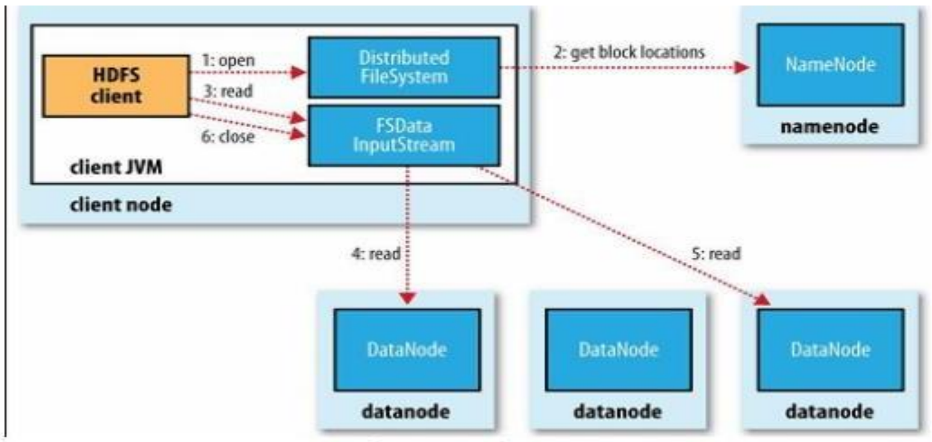


Fig. File Read Anatomy

File Write operation:

1. The client calls create() on DistributedFileSystem to create a file.

2. An RPC call to the namenode happens through the DFS to create a new file.

3. As the client writes data, data is split into packets by DFSOutputStream, which is then writes to an internal queue, called data queue. Data streamer consumes the data queue.

4. Data streamer streams the packets to the first DataNode in the pipeline. It stores packet and forwards it to the second DataNode in the pipeline.

5. In addition to the internal queue, DFSOutputStream also manages on “Ackqueue” of the packets that are waiting for acknowledged by DataNodes.

6. When the client finishes writing the file, it calls close() on the stream.

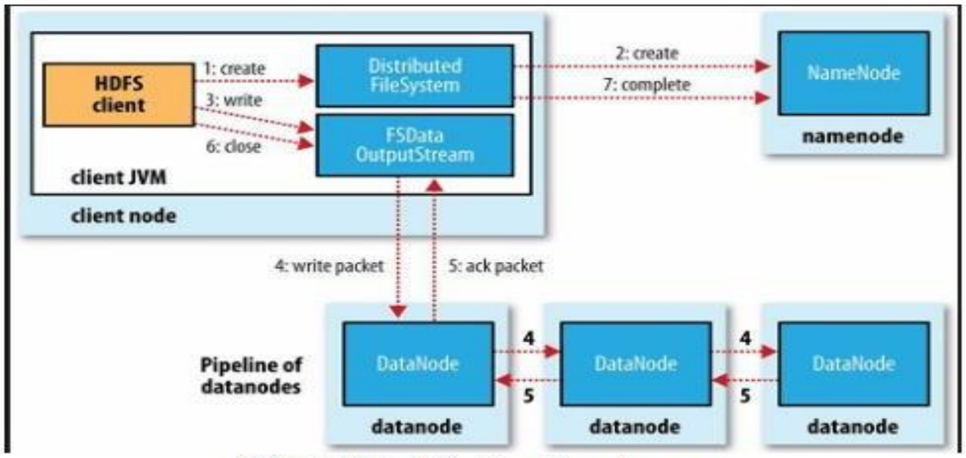


Fig. File Write Anatomy

Data format

When processing Big data, the cost required to store such data is more (Hadoop stores data redundantly to achieve fault tolerance). Along with the storage cost, processing the data comes with CPU, Network, IO costs, etc.. As the data increases, the cost for processing and storage increases too.

Choosing an appropriate file format can have some significant benefits:

1. Faster read times
2. Faster write times
3. Splittable files
4. Schema evolution support
5. Advanced compression support

There are mainly **7 file formats supported by Hadoop**. We will see each one in detail here-

1. **Text/CSV Files**
2. **JSON Records**
3. **Avro Files**
4. **Sequence Files**
5. **RC Files**
6. **ORC Files**
7. **Parquet Files**

Text/CSV Files

Text and CSV files are quite common and frequently Hadoop developers and data scientists received text and CSV files to work upon.

However CSV files do not support block compression, thus compressing a CSV file in Hadoop often comes at a significant read performance cost.

Each line in these files should have a record and so there is no metadata stored in these files.

JSON Records: JSON records contain JSON files where each line is its own JSON datum. In the case of JSON files, metadata is stored and the file is also splittable but again it also doesn't support block compression.

The only issue is there is not much support in Hadoop for JSON file but thanks to the third party tools which helps a lot. Just do the experiment and get your work done.

AVRO File Format

Avro format is a row-based storage format for Hadoop, which is widely used as a serialization platform.

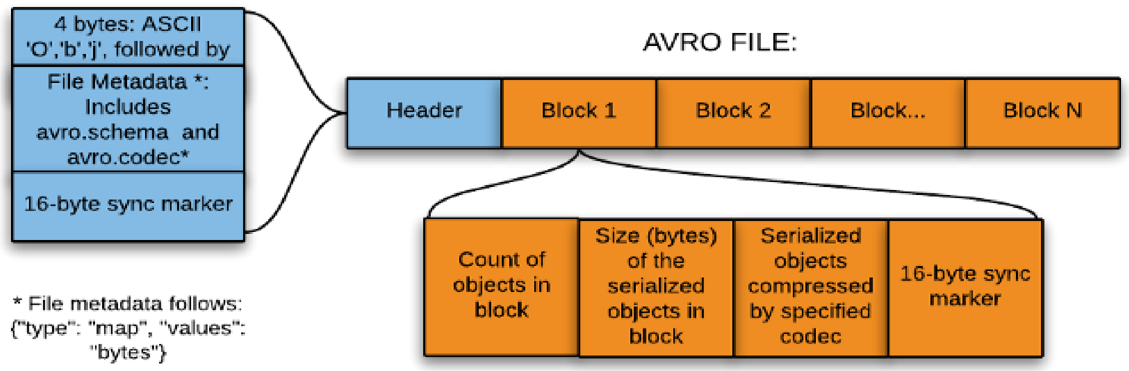
Avro format stores the schema in JSON format, making it easy to read and interpret by any program.

The data itself is stored in a binary format making it compact and efficient in Avro files.

Avro format is a language-neutral data serialization system. It can be processed by many languages (currently C, C++, C#, Java, Python, and Ruby).

A key feature of Avro format is the robust support for data schemas that changes over time, i.e., schema evolution. Avro handles schema changes like missing fields, added fields, and changed fields.

Avro format provides rich data structures. For example, you can create a record that contains an array, an enumerated type, and a sub-record.


4 bytes: ASCII 'O','b','j', followed by
File Metadata *: Includes avro.schema and avro.codec*
16-byte sync marker
* File metadata follows: {"type": "map", "values": "bytes"}
AVRO FILE:
Header
Block 1
Block 2
Block...
Block N
Count of objects in block
Size (bytes) of the serialized objects in block
Serialized objects compressed by specified codec
16-byte sync marker

PARQUET File Format

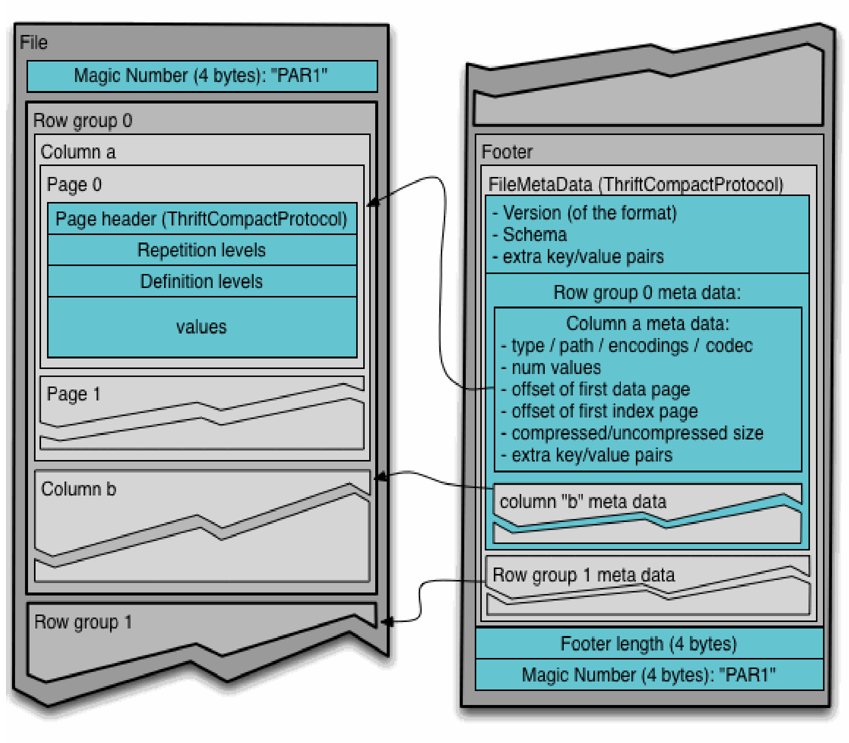
Parquet, an open-source file format for Hadoop, stores **nested data structures** in a flat **columnar format**.

Compared to a traditional approach where data is stored in a row-oriented approach, Parquet file format is more efficient in terms of storage and performance.

It is especially good for queries that read particular columns from a “wide” (with many columns) table since only needed columns are read, and IO is minimized.

One of the unique features of Parquet is that it can store data with nested structures in a columnar fashion too. This means that in a Parquet file format, even the nested fields can be read individually without reading all the fields in the nested structure.

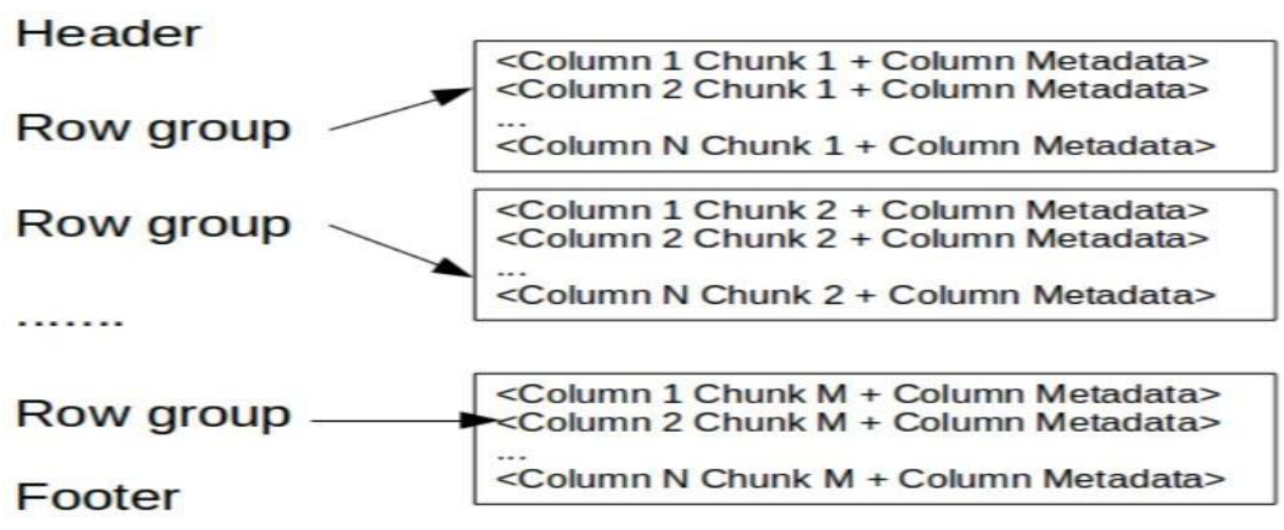
Parquet format uses the record shredding and assembly algorithm for storing nested structures in a columnar fashion.



File
Magic Number (4 bytes): "PAR1"
Row group 0
Column a
Page 0
Page header (ThriftCompactProtocol)
Repetition levels
Definition levels
values
Page 1
Column b
Row group 1
Footer
FileMetaData (ThriftCompactProtocol)
- Version (of the format)
- Schema
- extra key/value pairs
Row group 0 meta data:
Column a meta data:
- type / path / encodings / codec
- num values
- offset of first data page
- offset of first index page
- compressed/uncompressed size
- extra key/value pairs
column "b" meta data
Row group 1 meta data
Footer length (4 bytes)
Magic Number (4 bytes): "PAR1"

To understand the Parquet file format in Hadoop, you should be aware of the following terms-

- **Row group**: A logical horizontal partitioning of the data into rows. A row group consists of a column chunk for each column in the dataset.
- **Column chunk**: A chunk of the data for a particular column. These column chunks live in a particular row group and are guaranteed to be contiguous in the file.
- **Page**: Column chunks are divided up into pages written back to back. The pages share a standard header and readers can skip the page they are not interested in.



Header
Row group
Row group
....
Row group
Footer
<Column 1 Chunk 1 + Column Metadata>
<Column 2 Chunk 1 + Column Metadata>
...
<Column N Chunk 1 + Column Metadata>
<Column 1 Chunk 2 + Column Metadata>
<Column 2 Chunk 2 + Column Metadata>
...
<Column N Chunk 2 + Column Metadata>
<Column 1 Chunk M + Column Metadata>
<Column 2 Chunk M + Column Metadata>
...
<Column N Chunk M + Column Metadata>

ORC File Format

The Optimized Row Columnar(ORC) file format provides an effective way to store the data.

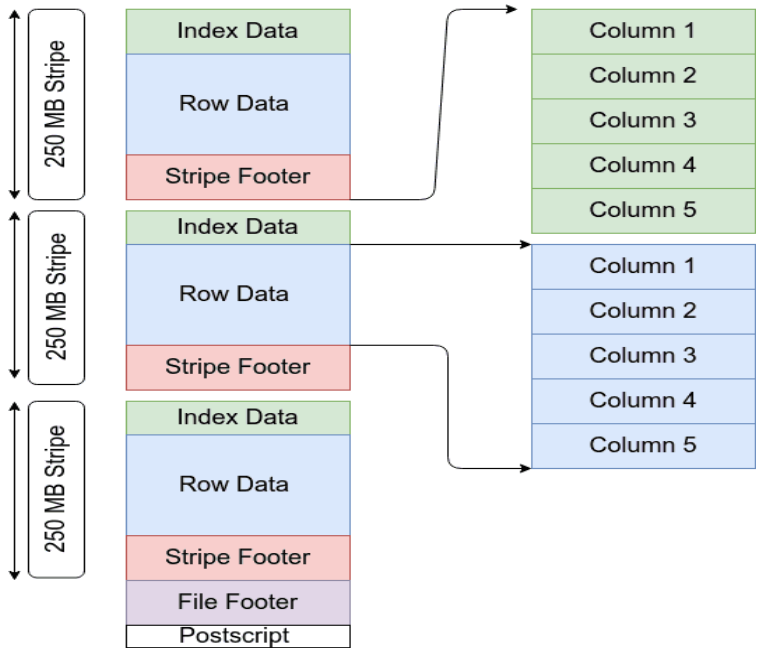
It is a column-oriented data storage format same as Parquet.

The advantages of ORC file formats are:-

- The compression ratio is very high, it reduces the size of the original data by up to 75%.
- It generates a single file as output for each of the tasks, by doing this it reduces the load from the NameNodes.

It provides three levels of indexing with each file:-

- **File-level** – ORC provides the statistics about the values in each column across the entire file.
- **Stripe level** – It provides statistics about the values in each column for each stripe.
- **Row-level** – In this, it provides the statistics about the values in each column for each set of 10,000 rows within a stripe.



Sequence File

The sequencefile format can be used to store an image in the binary format. They store key-value pairs in a binary container format and are more efficient than a text file. However, sequence files are not human- readable.

RC File (Record Columnar Files):

RC file was the first columnar file in Hadoop and has significant compression and query performance benefits.
But it doesn’t support schema evaluation and if you want to add anything to RC file you will have to rewrite the file. Also, it is a slower process.

Hadoop Streaming and Pipes

Streaming and Pipes Both Streaming and Pipes run special map and reduce tasks for the purpose of launching the user-supplied executable and communicating with it.

In the case of Streaming, the Streaming task communicates with the process (which may be written in any language) using standard input and output streams.

The Pipes task, on the other hand, listens on a socket and passes the C++ process a port number in its environment, so that on startup, the C++ process can establish a persistent socket connection back to the parent Java Pipes task.

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.

Hadoop provides an API to MapReduce that allows you to write your map and reduce functions in languages other than Java.

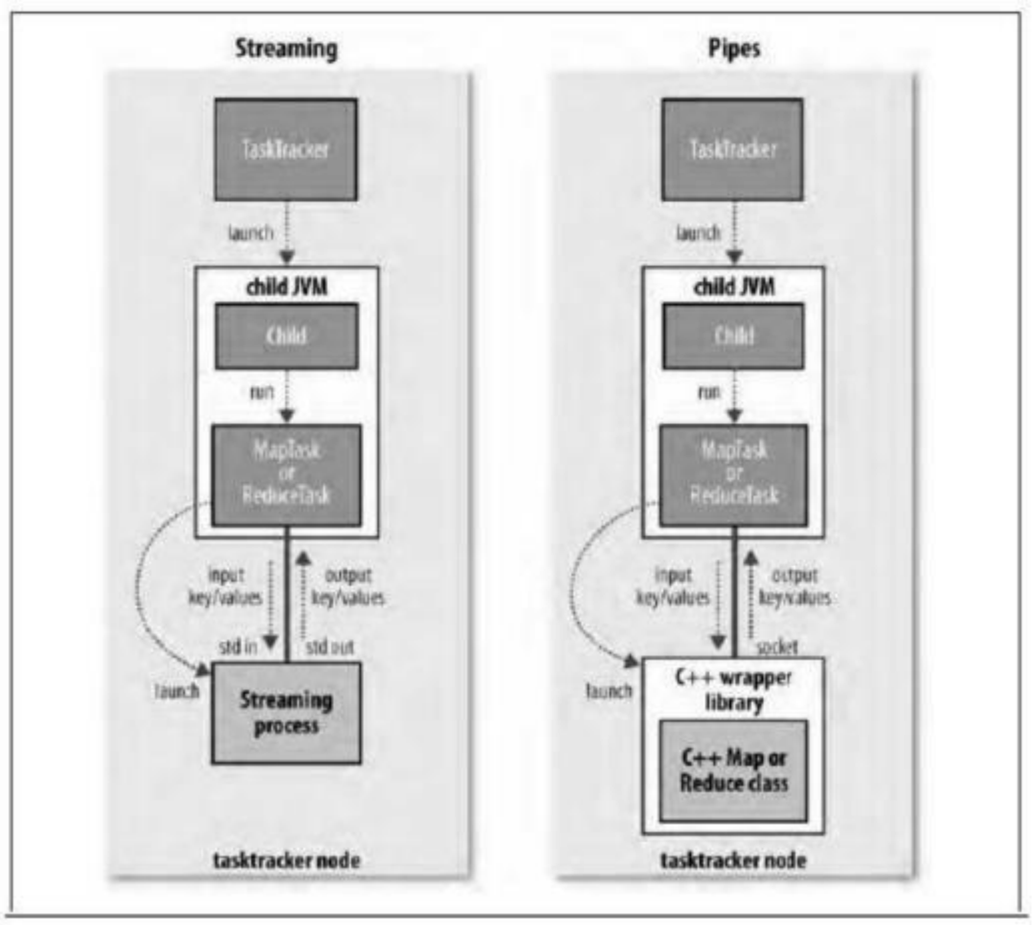
Hadoop Streaming uses Unix standard stream as the interface between Hadoop and your program, so you can use any language that can read standard input and write to standard output to write your MapReduce program.

Streaming is naturally suited for text processing (although, as of version 0.21.0, it can handle binary streams, too), and when used in text mode, it has a line-oriented view of data.

Map input data is passed over standard input to your map function, which processes it line by line and writes lines to standard output. A map output key-value pair is written as a single tabdelimited line

Input to the reduce function is in the same format a tab- -value pair passed over separated key standard input.

The reduce function reads lines from standard input, which the framework guarantees are sorted by key, and writes its results to standard output.



Hadoop Pipes

Hadoop Pipes is the name of the C++ interface to Hadoop MapReduce. Unlike treaming, which uses standard input and output to communicate with the map and reduce code, Pipes uses sockets as the channel over which the tasktracker communicates with the process running the C++ map or reduce function. JNI is not used.

Hadoop Pipes is the name of the C++ interface to Hadoop MapReduce. Unlike Streaming, which uses standard input and output to communicate with the map and reduce code, Pipes uses sockets as the channel over which the tasktracker communicates with the process running the C++ map or reduce function. JNI is not used.

```
class MaxTemperatureMapper : public HadoopPipes::Mapper
{
public:
        MaxTemperatureMapper(HadoopPipes::TaskContext& context) { }
          void map(HadoopPipes::MapContext& context)
          {
          // statement to convert the input data into string
          // statement to obtain year and temp using the substring method
          // statement to place the year and temp into a set
          }

  };

  class MapTemperatureReducer : public HadoopPipes::Reducer
  {
  public:
          MapTemperatureReducer(HadoopPipes::TaskContext& context) { }
          void reduce(HadoopPipes::ReduceContext& context)
          {
          // statement to find the maximum temperature of a each year
          // statement to put the max. temp and its year in a set
          }
  };

  int main(int argc, char *argv[])
  {
  return HadoopPipes::runTask( HadoopPipes:: TemplateFactory < MaxTemperatureMapper,
   MapTemperatureReducer>());
  }
```

```
int main(int argc, char *argv[])
{
return HadoopPipes::runTask( HadoopPipes:: TemplateFactory < MaxTemperatureMapper,
 MapTemperatureReducer>());
}
```

The map and reduce functions are defined by extending the Mapper and Reducer classes defined in the HadoopPipes namespace and providing implementations of the map() and reduce() methods in each case.

These methods take a context object (of type MapContext or ReduceContext), which provides the means for reading input and writing output, as well as accessing job configuration information via the JobConf class.

The main() method is the application entry point. It calls HadoopPipes::runTask, which connects to the Java parent process and marshals data to and from the Mapper or Reducer. The runTask() method is passed a Factory so that it can create instances of the Mapper or Reducer.

Hadoop Ecosystem

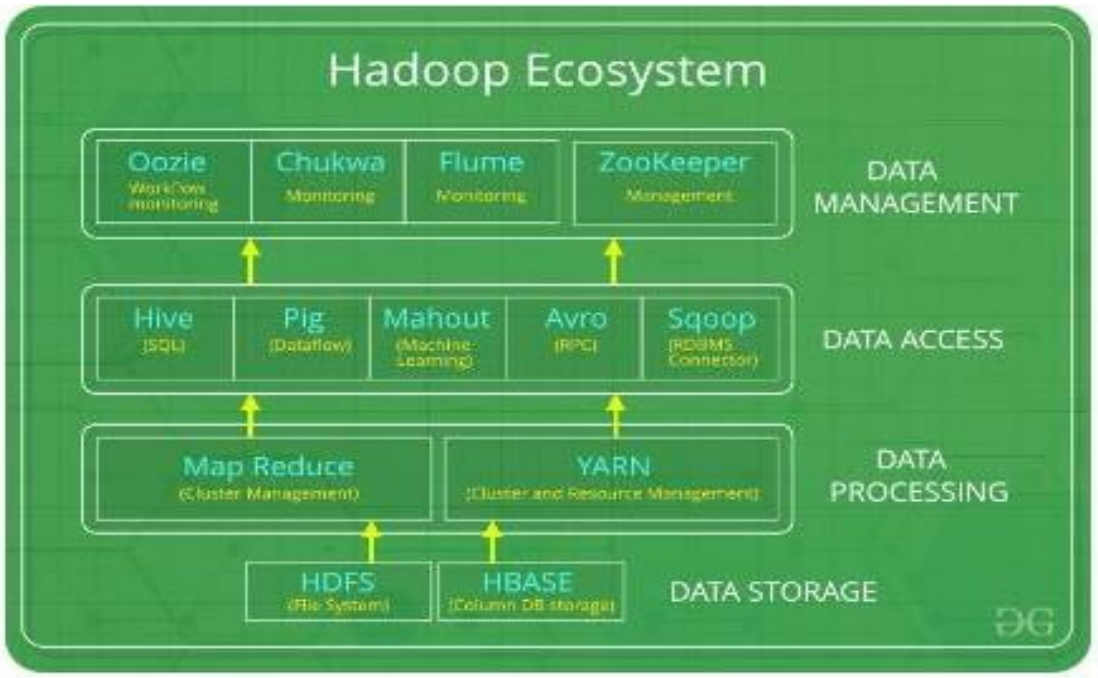
Hadoop Ecosystem is a platform or a suite which provides various services to solve the big data problems. It includes Apache projects and various commercial tools and solutions.

There are *four major elements of Hadoop* i.e. **HDFS, MapReduce, YARN, and Hadoop Common**. Most of the tools or solutions are used to supplement or support these major elements.

All these tools work collectively to provide services such as absorption, analysis, storage and maintenance of data etc.

Following are the components that collectively form a Hadoop ecosystem:

- **HDFS:** Hadoop Distributed File System
- **YARN:** Yet Another Resource Negotiator
- **MapReduce:** Programming based Data Processing
- **Spark:** In-Memory data processing
- **PIG, HIVE:** Query based processing of data services
- **HBase:** NoSQL Database
- **Mahout, Spark MLLib:** Machine Learning algorithm libraries
- **Solar, Lucene:** Searching and Indexing
- **Zookeeper:** Managing cluster
- **Oozie:** Job Scheduling



HDFS:

- HDFS is the primary or major component of Hadoop ecosystem and is responsible for storing large data sets of structured or unstructured data across various nodes and thereby maintaining the metadata in the form of log files.
- HDFS consists of two core components i.e.
  1. Name node
  2. Data Node
- Name Node is the prime node which contains metadata (data about data) requiring comparatively fewer resources than the data nodes that stores the actual data. These data nodes are commodity hardware in the distributed environment. Undoubtedly, making Hadoop cost effective.
- HDFS maintains all the coordination between the clusters and hardware, thus working at the heart of the system.

YARN:

- Yet Another Resource Negotiator, as the name implies, YARN is the one who helps to manage the resources across the clusters. In short, it performs scheduling and resource allocation for the Hadoop System.
- Consists of three major components i.e.
  1. Resource Manager
  2. Nodes Manager
  3. Application Manager
- Resource manager has the privilege of allocating resources for the applications in a system whereas Node managers work on the allocation of resources such as CPU, memory, bandwidth per machine and later on acknowledges the resource manager. Application manager works as an

interface between the resource manager and node manager and performs negotiations as per the requirement of the two.

MapReduce:

- By making the use of distributed and parallel algorithms, MapReduce makes it possible to carry over the processing's logic and helps to write applications which transform big data sets into a manageable one.
- MapReduce makes the use of two functions i.e. Map() and Reduce() whose task is:
    1. Map() performs sorting and filtering of data and thereby organizing them in the form of group. Map generates a key-value pair based result which is later on processed by the Reduce() method.
    2. Reduce(), as the name suggests does the summarization by aggregating the mapped data. In simple, Reduce() takes the output generated by Map() as input and combines those tuples into smaller set of tuples.

PIG:

Pig was basically developed by Yahoo which works on a pig Latin language, which is Query based language similar to SQL.

- It is a platform for structuring the data flow, processing and analyzing huge data sets.
- Pig does the work of executing commands and in the background, all the activities of MapReduce are taken care of. After the processing, pig stores the result in HDFS.
- Pig Latin language is specially designed for this framework which runs on Pig Runtime. Just the way Java runs on the JVM.
- Pig helps to achieve ease of programming and optimization and hence is a major segment of the Hadoop Ecosystem.

HIVE:

With the help of SQL methodology and interface, HIVE performs reading and writing of large data sets. However, its query language is called as HQL (Hive Query Language).

- It is highly scalable as it allows real-time processing and batch processing both. Also, all the SQL datatypes are supported by Hive thus, making the query processing easier.
- Similar to the Query Processing frameworks, HIVE too comes with two components: JDBC Drivers and HIVE Command Line.
- JDBC, along with ODBC drivers work on establishing the data storage permissions and connection whereas HIVE Command line helps in the processing of queries.

Mahout:

- Mahout, allows Machine Learnability to a system or application. Machine Learning, as the name suggests helps the system to develop itself based on some patterns, user/environmental interaction or on the basis of algorithms.
- It provides various libraries or functionalities such as collaborative filtering, clustering, and classification which are nothing but concepts of Machine learning. It allows invoking algorithms as per our need with the help of its own libraries.

Apache Spark:

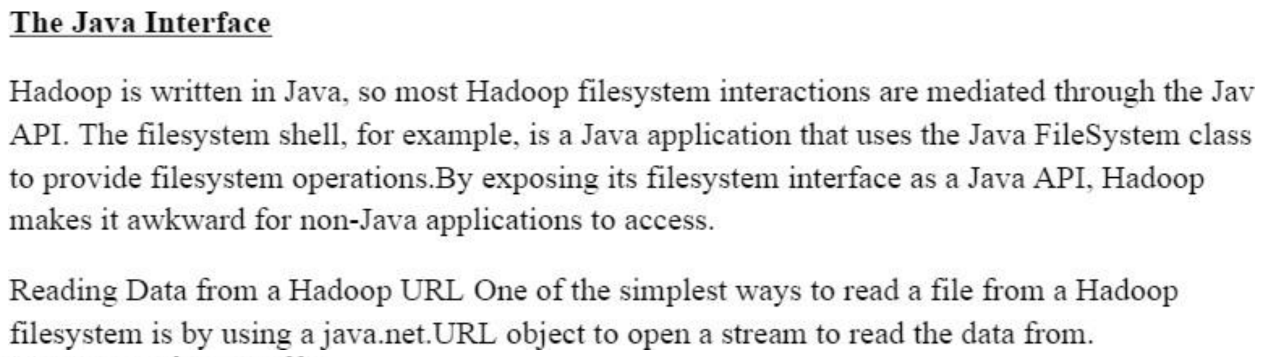
- It's a platform that handles all the process consumptive tasks like batch processing, interactive or iterative real-time processing, graph conversions, and visualization, etc.

- It consumes in memory resources hence, thus being faster than the prior in terms of optimization.
- Spark is best suited for real-time data whereas Hadoop is best suited for structured data or batch processing, hence both are used in most of the companies interchangeably.

ApacheHBase:

- It's a NoSQL database which supports all kinds of data and thus capable of handling anything of Hadoop Database. It provides capabilities of Google's BigTable, thus able to work on Big Data sets effectively.
- At times where we need to search or retrieve the occurrences of something small in a huge database, the request must be processed within a short quick span of time. At such times, HBase comes handy as it gives us a tolerant way of storing limited data

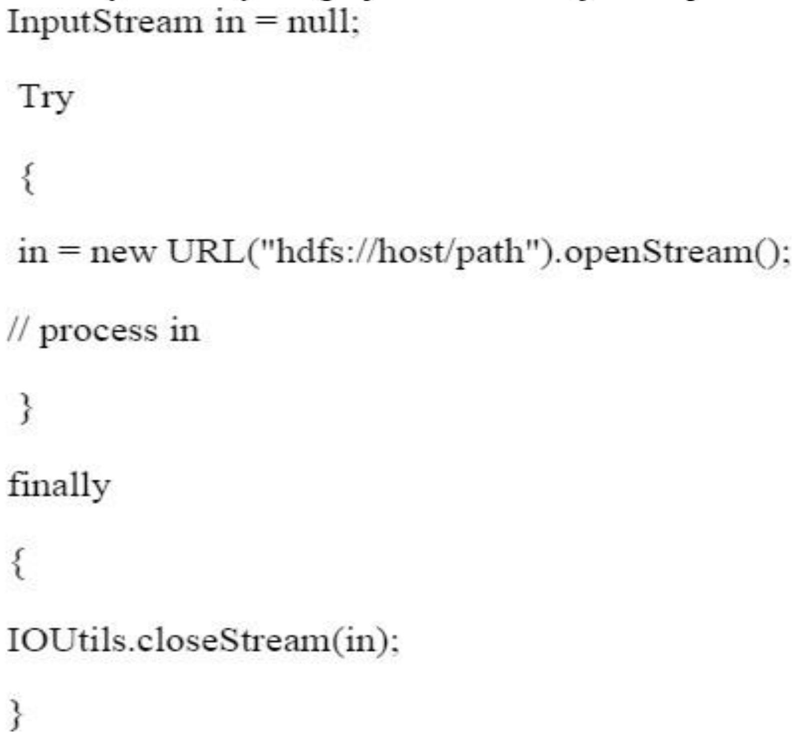
Java interfaces to HDFS Basics



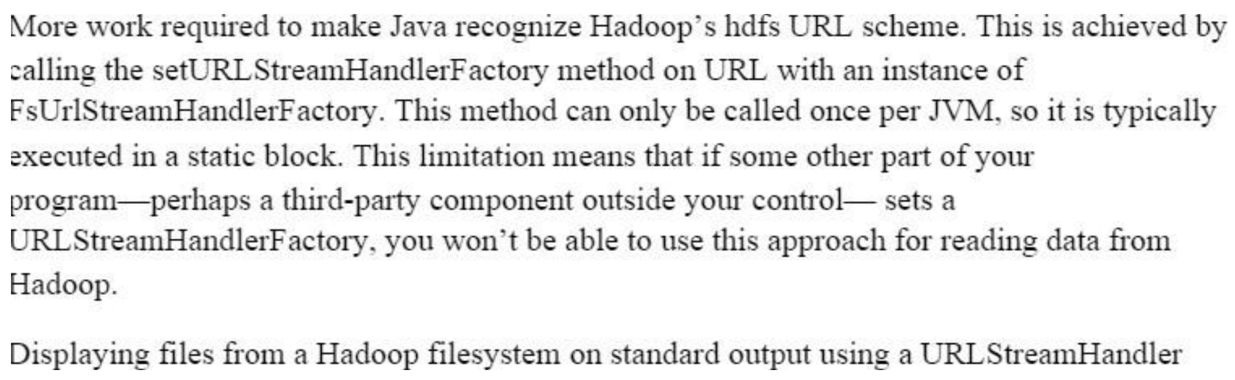
The Java Interface

Hadoop is written in Java, so most Hadoop filesystem interactions are mediated through the Jav API. The filesystem shell, for example, is a Java application that uses the Java FileSystem class to provide filesystem operations.By exposing its filesystem interface as a Java API, Hadoop makes it awkward for non-Java applications to access.

Reading Data from a Hadoop URL One of the simplest ways to read a file from a Hadoop filesystem is by using a java.net.URL object to open a stream to read the data from.



```
InputStream in = null;

 Try

 {

 in = new URL("hdfs://host/path").openStream();

// process in

 }

finally

{

IOUtils.closeStream(in);

}
```


More work required to make Java recognize Hadoop's hdfs URL scheme. This is achieved by calling the setURLStreamHandlerFactory method on URL with an instance of FsUrlStreamHandlerFactory. This method can only be called once per JVM, so it is typically executed in a static block. This limitation means that if some other part of your program—perhaps a third-party component outside your control— sets a URLStreamHandlerFactory, you won't be able to use this approach for reading data from Hadoop.

Displaying files from a Hadoop filesystem on standard output using a URLStreamHandler

Developing a Map Reduce Application

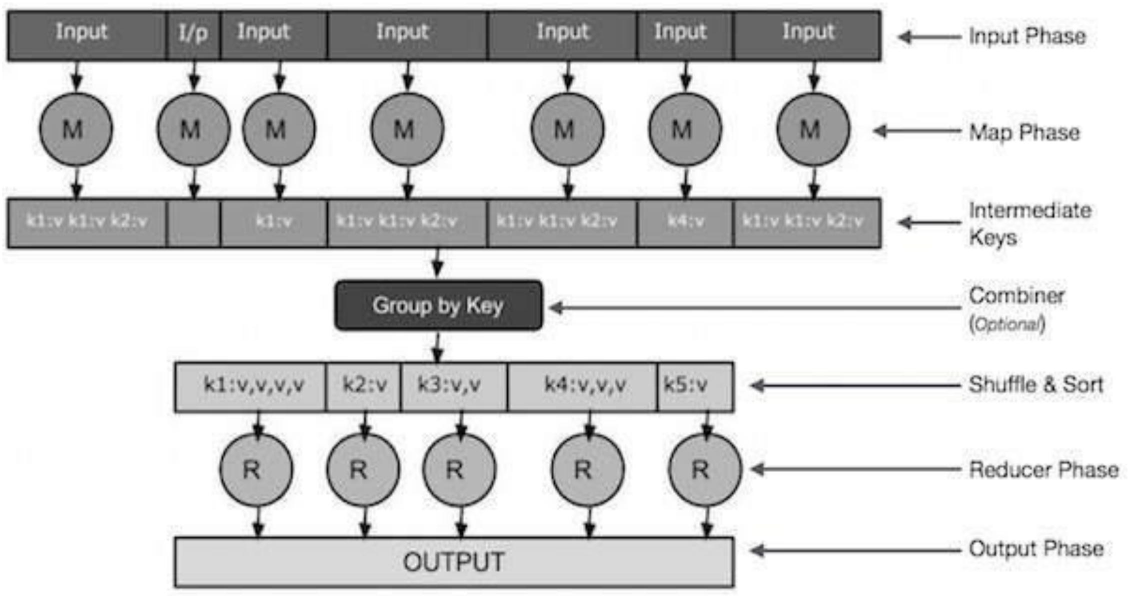
Entertainment: To discover the most popular movies, based on what you like and what you watched in this case Hadoop MapReduce help you out. It mainly focuses on their logs and clicks.
E-commerce: Numerous E-commerce suppliers, like Amazon, Walmart, and eBay, utilize the MapReduce programming model to distinguish most loved items dependent on clients' inclinations or purchasing behavior.
It incorporates making item proposal Mechanisms for E-commerce inventories, examining website records, buy history, user interaction logs, etc.

Data Warehouse: We can utilize MapReduce to analyze large data volumes in data warehouses while implementing specific business logic for data insights.
Fraud Detection: Hadoop and MapReduce are utilized in monetary enterprises, including organizations like banks, insurance providers, installment areas for misrepresentation recognition, pattern distinguishing proof, or business metrics through transaction analysis.

How does MapReduce Works?

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.



Input Phase − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
Map − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
Intermediate Keys − The key-value pairs generated by the mapper are known as intermediate keys.
Combiner − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
Shuffle and Sort − The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.
Reducer − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.
Output Phase − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

Anatomy of a MapReduce Job Run

Hadoop MapReduce jobs are divided into a set of map tasks and reduce tasks that run in a distributed fashion on a cluster of computers. Each task work on a small subset of the data it has been assigned so that the load is spread across the cluster.

The input to a MapReduce job is a set of files in the data store that are spread out over the HDFS. In Hadoop, these files are split with an input format, which defines how to separate a files into input split. You can assume that input split is a byte-oriented view of a chunk of the files to be loaded by a map task.

You can run a MapReduce job with a single method call: submit() on a Job object (you can also call waitForCompletion(), which submits the job if it hasn't been submitted already, then waits for it to finish).[51] This method call conceals a great deal of processing behind the scenes. This section uncovers the steps Hadoop takes to run a job.

The whole process is illustrated in Figure 7-1. At the highest level, there are five independent entities:[52]

- The client, which submits the MapReduce job.
- The YARN resource manager, which coordinates the allocation of compute resources on the cluster.
- The YARN node managers, which launch and monitor the compute containers on machines in the cluster.
- The MapReduce application master, which coordinates the tasks running the MapReduce job. The application master and the MapReduce tasks run in containers that are scheduled by the resource manager and managed by the node managers.
- The distributed filesystem (normally HDFS, covered in Chapter 3), which is used for sharing job files between the other entities.

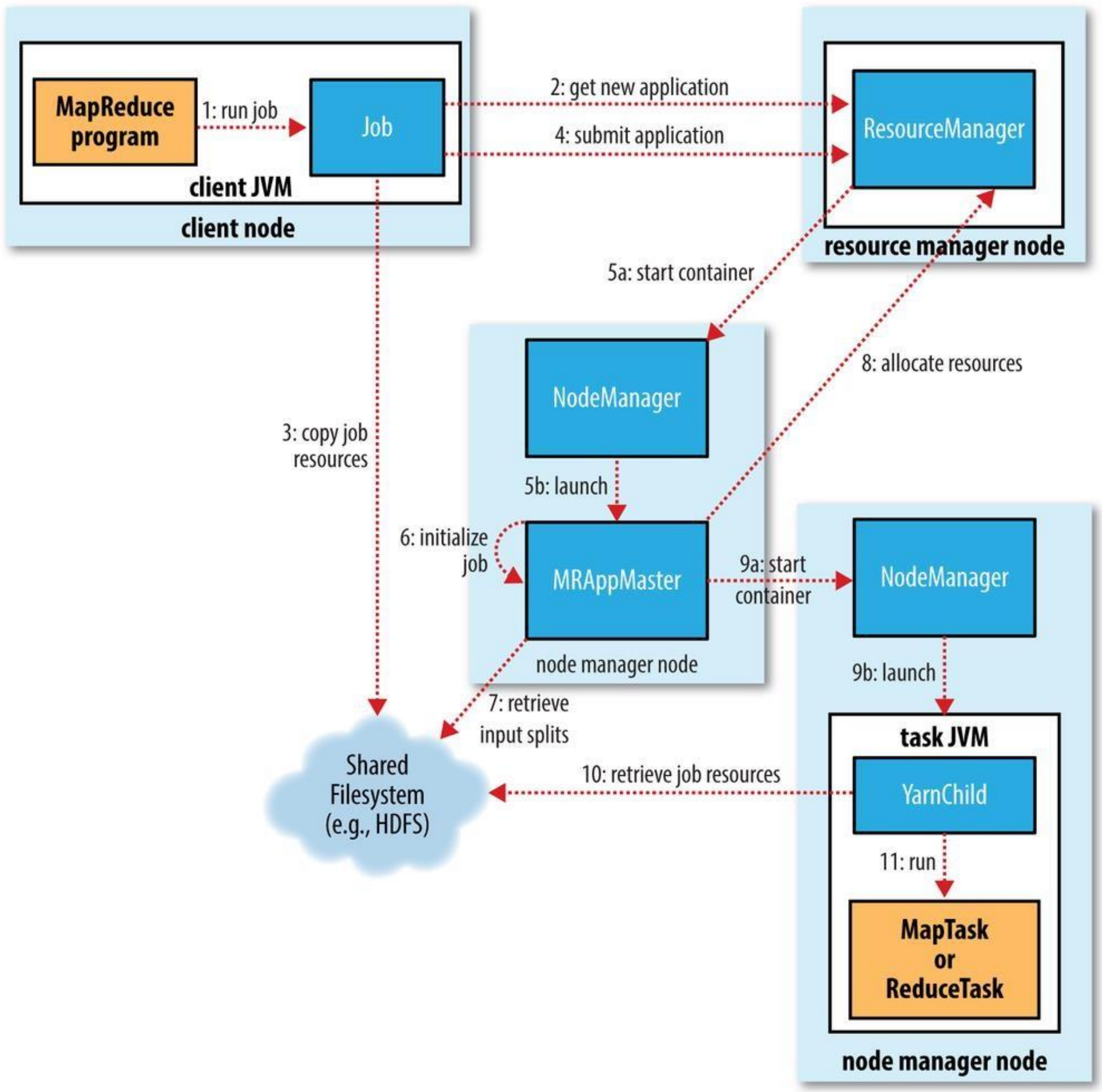


Figure 7-1. How Hadoop runs a MapReduce job

Map Reduce Features Hadoop environment.

1. Scalability

Apache Hadoop is a highly scalable framework. This is because of its ability to store and distribute huge data across plenty of servers. All these servers were inexpensive and can operate in parallel. We can easily scale the storage and computation power by adding servers to the cluster.

Hadoop MapReduce programming enables organizations to run applications from large sets of nodes which could involve the use of thousands of terabytes of data.

Hadoop MapReduce programming enables business organizations to run applications from large sets of nodes. This can use thousands of terabytes of data.

2. Flexibility
MapReduce programming enables companies to access new sources of data. It enables companies to operate on different types of data. It allows enterprises to access structured as well as unstructured data, and derive significant value by gaining insights from the multiple sources of data.

Additionally, the MapReduce framework also provides support for the multiple languages and data from sources ranging from email, social media, to clickstream.

The MapReduce processes data in simple key-value pairs thus supports data type including meta-data, images, and large files. Hence, MapReduce is flexible to deal with data rather than traditional DBMS.

3. Security and Authentication
The MapReduce programming model uses HBase and HDFS security platform that allows access only to the authenticated users to operate on the data. Thus, it protects unauthorized access to system data and enhances system security.

4. Cost-effective solution
Hadoop's scalable architecture with the MapReduce programming framework allows the storage and processing of large data sets in a very affordable manner.

5. Fast
Hadoop uses a distributed storage method called as a Hadoop Distributed File System that basically implements a mapping system for locating data in a cluster.

The tools that are used for data processing, such as MapReduce programming, are generally located on the very same servers that allow for the faster processing of data.

So, Even if we are dealing with large volumes of unstructured data, Hadoop MapReduce just takes minutes to process terabytes of data. It can process petabytes of data in just an hour.

6. Simple model of programming
Amongst the various features of Hadoop MapReduce, one of the most important features is that it is based on a simple programming model. Basically, this allows programmers to develop the MapReduce programs which can handle tasks easily and efficiently.

The MapReduce programs can be written in Java, which is not very hard to pick up and is also used widely. So, anyone can easily learn and write MapReduce programs and meet their data processing needs.

7. Parallel Programming
One of the major aspects of the working of MapReduce programming is its parallel processing. It

divides the tasks in a manner that allows their execution in parallel.
The parallel processing allows multiple processors to execute these divided tasks. So the entire program is run in less time.

8. Availability and resilient nature

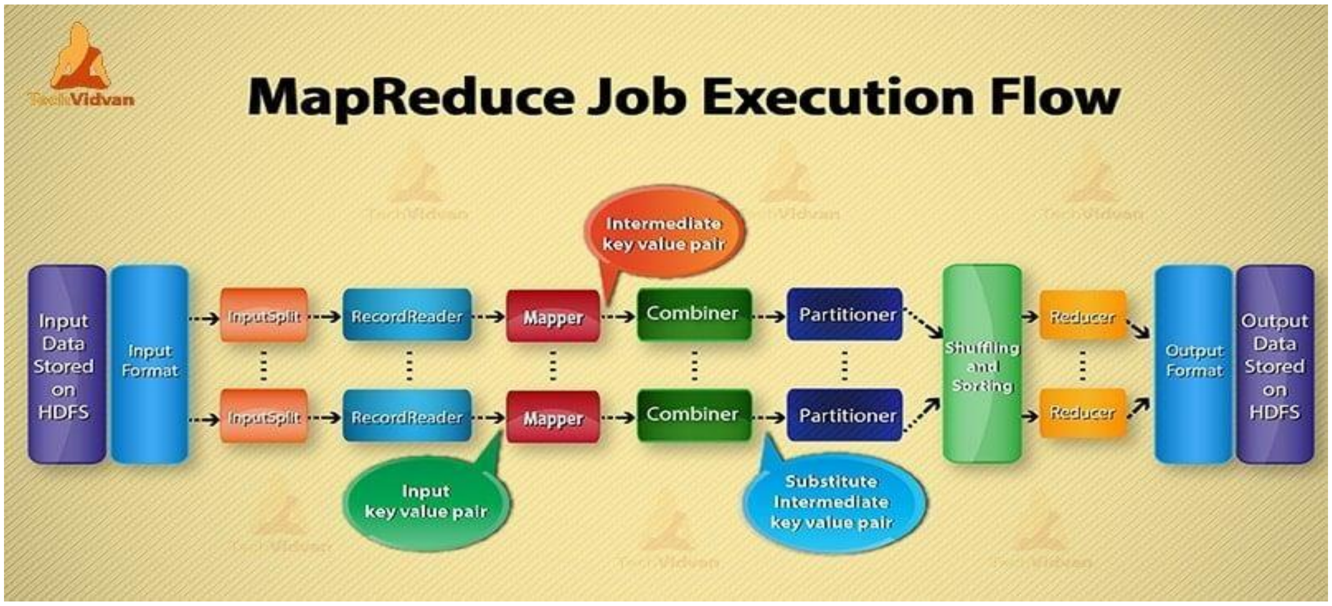
Whenever the data is sent to an individual node, the same set of data is forwarded to some other nodes in a cluster. So, if any particular node suffers from a failure, then there are always other copies present on other nodes that can still be accessed whenever needed. This assures high availability of data.

One of the major features offered by Apache Hadoop is its fault tolerance. The Hadoop MapReduce framework has the ability to quickly recognizing faults that occur.

It then applies a quick and automatic recovery solution. This feature makes it a game-changer in the world of big data processing.

Task execution

MapReduce processess the data in various phases with the help of different components. Let's discuss the steps of job execution in Hadoop.


TechVidvan
MapReduce Job Execution Flow
Intermediate key value pair
Input Data Stored on HDFS
Input Format
InputSplit
RecordReader
Mapper
Combiner
Partitioner
Shuffling and Sorting
Reducer
Output Format
Output Data Stored on HDFS
Input key value pair
Substitute Intermediate key value pair

1. Input Files

In input files data for MapReduce job is stored. In HDFS, input files reside. Input files format is arbitrary. Line-based log files and binary format can also be used.

2. InputFormat

After that InputFormat defines how to split and read these input files. It selects the files or other objects for input. InputFormat creates InputSplit.

3. InputSplits

It represents the data which will be processed by an individual Mapper. For each split, one map task is created. Thus the number of map tasks is equal to the number of InputSplits. Framework divide split into records, which mapper process.

4. RecordReader

It communicates with the inputSplit. And then converts the data into key-value pairs suitable for reading by the Mapper. RecordReader by default uses TextInputFormat to convert data into a key-value pair.

It communicates to the InputSplit until the completion of file reading. It assigns byte offset to each line present in the file. Then, these key-value pairs are further sent to the mapper for further processing.

5. Mapper

It processes input record produced by the RecordReader and generates intermediate key-value pairs. The intermediate output is completely different from the input pair. The output of the mapper is the full collection of key-value pairs.

Hadoop framework doesn't store the output of mapper on HDFS. It doesn't store, as data is temporary and writing on HDFS will create unnecessary multiple copies. Then Mapper passes the output to the combiner for further processing.

6. Combiner

Combiner is Mini-reducer which performs local aggregation on the mapper's output. It minimizes the data transfer between mapper and reducer. So, when the combiner functionality completes, framework passes the output to the partitioner for further processing.

7. Partitioner

Partitioner comes into the existence if we are working with more than one reducer. It takes the output of the combiner and performs partitioning.

Partitioning of output takes place on the basis of the key in MapReduce. By hash function, key (or a subset of the key) derives the partition.

On the basis of key value in MapReduce, partitioning of each combiner output takes place. And then the record having the same key value goes into the same partition. After that, each partition is sent to a reducer.

Partitioning in MapReduce execution allows even distribution of the map output over the reducer.

8. Shuffling and Sorting

After partitioning, the output is shuffled to the reduce node. The shuffling is the physical movement of the data which is done over the network. As all the mappers finish and shuffle the output on the reducer nodes.

Then framework merges this intermediate output and sort. This is then provided as input to reduce phase.

9. Reducer

Reducer then takes set of intermediate key-value pairs produced by the mappers as the input. After that runs a reducer function on each of them to generate the output.

The output of the reducer is the final output. Then framework stores the output on HDFS.

10. RecordWriter

It writes these output key-value pair from the Reducer phase to the output files.

11. OutputFormat

OutputFormat defines the way how RecordReader writes these output key-value pairs in output files. So, its instances provided by the Hadoop write files in HDFS. Thus OutputFormat instances write the final output of reducer on HDFS.

Conclusion

We have learned step by step MapReduce job execution flow. I hope this blog helps you a lot to understand the MapReduce working.

If still, you have any query related to MapReduce job execution flow, so you can share with us in the comment section given below. We will try our best to solve them.

Write a java Program to implement Word Count With Map Reduce

In Hadoop, MapReduce is a computation that decomposes large manipulation jobs into individual tasks that can be executed in parallel across a cluster of servers. The results of tasks can be joined together to compute final results.

MapReduce consists of 2 steps:

Map Function – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

Example – (Map function in Word Count)

Set of data	Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN,BUS, buS, caR, CAR, car, BUS, TRAIN	
Output	Convert into another set of data (Key,Value)	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Work Flow of the Program

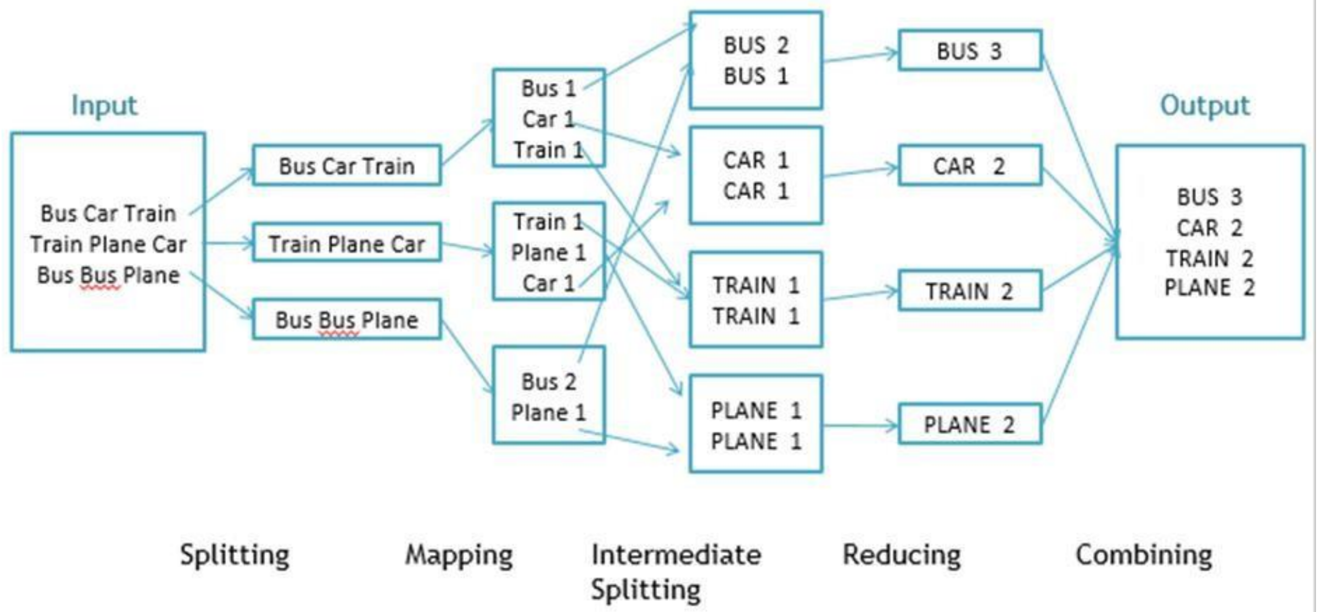


Fig. WorkFlow of MapReducing

```
package PackageDemo;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

public static void main(String [] args) throws Exception

{

Configuration c=new Configuration();

String[] files=new GenericOptionsParser(c,args).getRemainingArgs();

Path input=new Path(files[0]);

Path output=new Path(files[1]);

Job j=new Job(c,"wordcount");

j.setJarByClass(WordCount.class);

j.setMapperClass(MapForWordCount.class);

j.setReducerClass(ReduceForWordCount.class);

j.setOutputKeyClass(Text.class);
```

```
j.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(j, input);
FileOutputFormat.setOutputPath(j, output);
System.exit(j.waitForCompletion(true)?0:1);
}
public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{
public void map(LongWritable key, Text value, Context con) throws IOException, InterruptedException
{
String line = value.toString();
String[] words=line.split(",");
for(String word: words )
{
    Text outputKey = new Text(word.toUpperCase().trim());
  IntWritable outputValue = new IntWritable(1);
  con.write(outputKey, outputValue);
}
}
}
public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>
{
public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException
{
int sum = 0;
  for(IntWritable value : values)
  {
```

```
  sum += value.get();

  }

  con.write(word, new IntWritable(sum));

}


}


}
```

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.

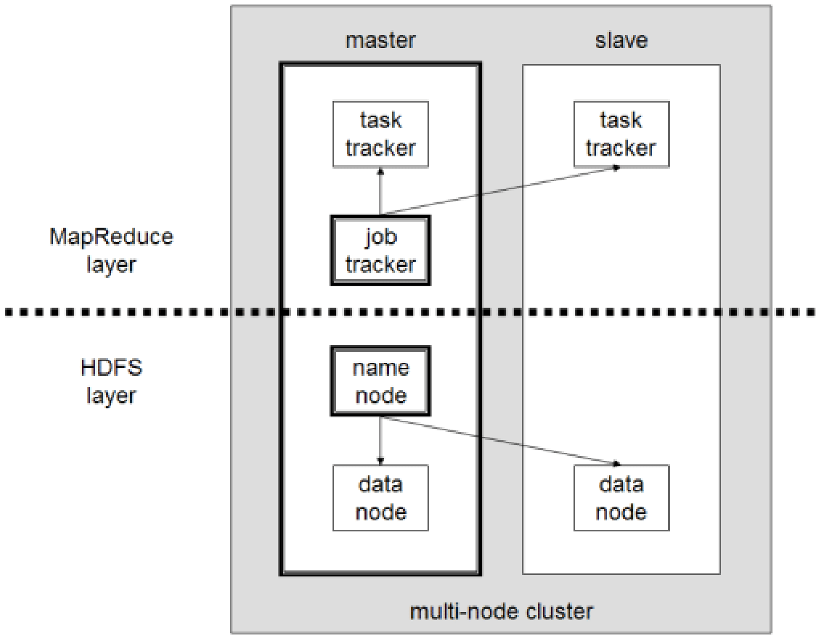


Fig. 2.11 HDFS Layer

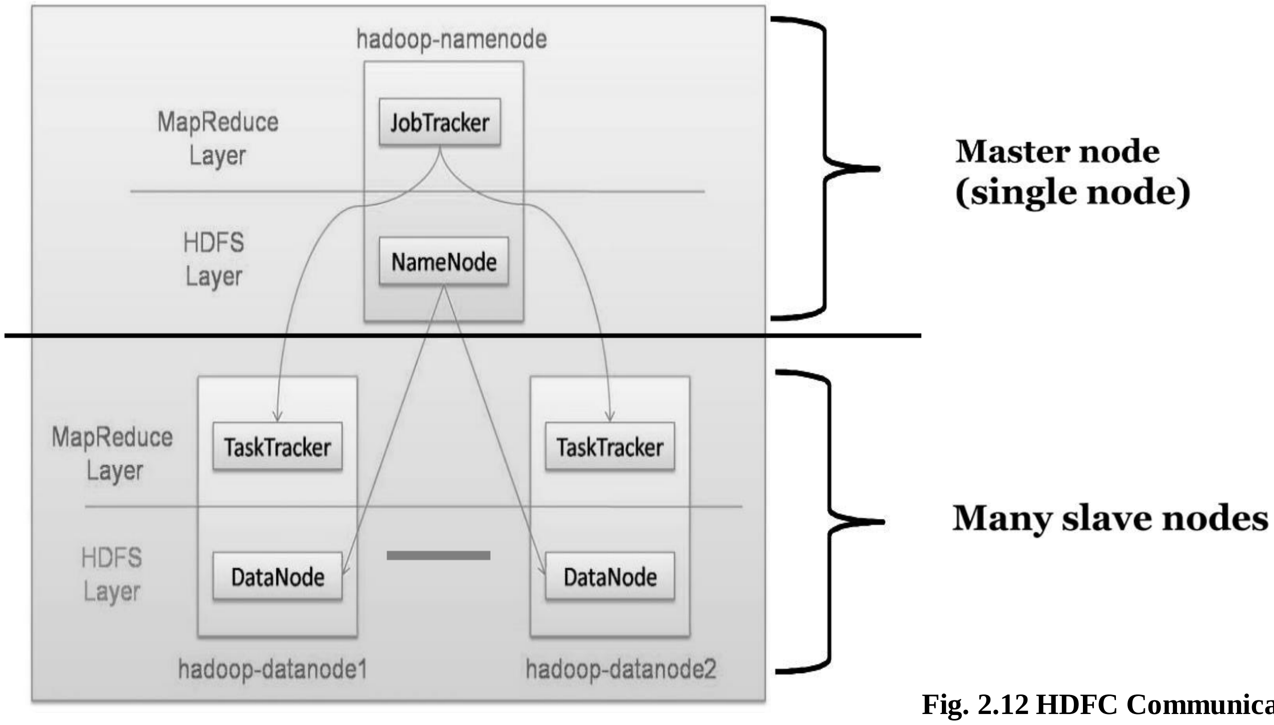


Fig. 2.12 HDFC Communication

Components of HDFS

NameNode

- Single node in cluster
- Brain of HDFS
- Hadoop employs master/slave architecture for both distributed storage and distributed computation
- Distributed Storage system is HDFS
- NameNode is the master of HDFS that directs slave DataNode to perform low level tasks. (Figure 2.13)

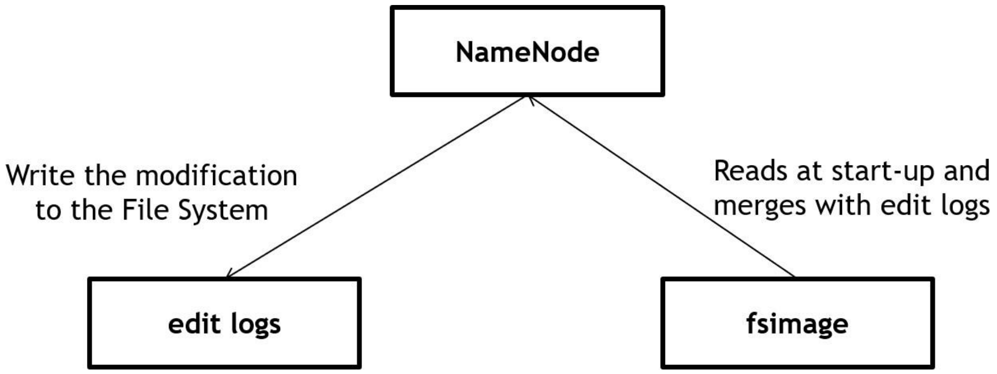

NameNode
Write the modification to the File System
Reads at start-up and merges with edit logs
edit logs
fsimage

Fig. 2.13 Components of HDFS

NameNode as book Keeper of HDFS

- NameNode 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 NameNode is memory and IO intensive
- The server hosting NameNode 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 NameNode 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 (Figure 2.14, 2.15)

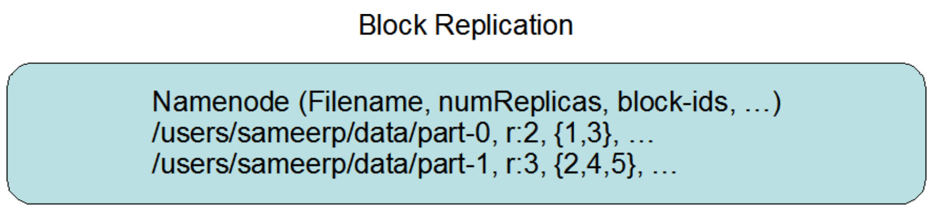

Block Replication
Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

Fig. 2.14 Block Replication

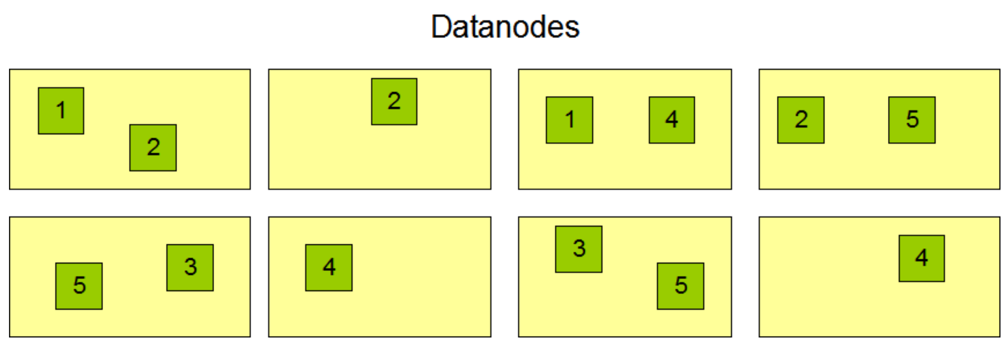


Fig. 2.15 DataNodes

- 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/hdfs-default.xml and src/mapred/mapred-default.xml.
    - Site-specific configuration - *conf/core-site.xml*, *conf/hdfs-site.xml* and *conf/mapred-site.xml*.

Configuring the Environment of the Hadoop Daemons

Table 2.4 Hadoop Configuration Options

Daemon	Configuration Options
NameNode	HADOOP_NAMENODE_OPTS
DataNode	HADOOP_DATANODE_OPTS
SecondaryNamenode	HADOOP_SECONDARYNAMENODE_OPTS
JobTracker	HADOOP_JOBTRACKER_OPTS
TaskTracker	HADOOP_TASKTRACKER_OPTS

Configuring the Hadoop Daemons

Table 2.5 Specified in conf/core-site.xml

Parameter	Value
fs.default.name	URI of NameNode.

Table 2.6 conf/hdfs-site.xml

Parameter	Description
dfs.name.dir	Path on the local filesystem where the NameNode stores the namespace and transactions logs persistently.
dfs.data.dir	Comma separated list of paths on the local filesystem of a DataNode where it should store its blocks.

Table 2.7 conf/mapred-site.xml

Parameter	Value
mapred.job.tracker	Host or IP and port of JobTracker.
mapred.system.dir	Path on the HDFS where the MapReduce framework stores system files e.g. /hadoop/mapred/system/.
mapred.local.dir	Comma-separated list of paths on the local filesystem where temporary MapReduce data is written.
mapred.tasktracker.{map|reduce}. tasks.maximum	The maximum number of MapReduce tasks, which are run simultaneously on a given TaskTracker, individually.
dfs.hosts/dfs.hosts.exclude	List of permitted/excluded DataNodes.
mapred.hosts/mapred.hosts.exclude	List of permitted/excluded TaskTrackers.
mapred.queue.names	Comma separated list of queues to which jobs can be submitted.
mapred.acls.enabled	Boolean, specifying whether checks for queue ACLs and job ACLs are to be done for authorizing users for doing queue operations and job operations

Table 2.8 conf/mapred-queue-acls.xml

Parameter	Value
mapred.queue.*queue-name*.acl-submit-job	List of users and groups that can submit jobs to the specified *queue-name*.
mapred.queue.*queue-name*.acl-administer-jobs	List of users and groups that can view job details, change the priority or kill jobs that have been submitted to the specified *queue-name*.

Introduction to Apache Hadoop

- With the continuous business growth and start-ups flourishing up, the need to store a large amount of data has also increased rapidly.
- To meet the growth of business and gain profits the companies would fetch tools to analyse the Big Data.
- To meet the growing demand, Apache Software Foundation launched Hadoop, a tool to store, analyse, and process Big Data.

What is Hadoop?

- Hadoop is an open source Java-based framework for big data processing.

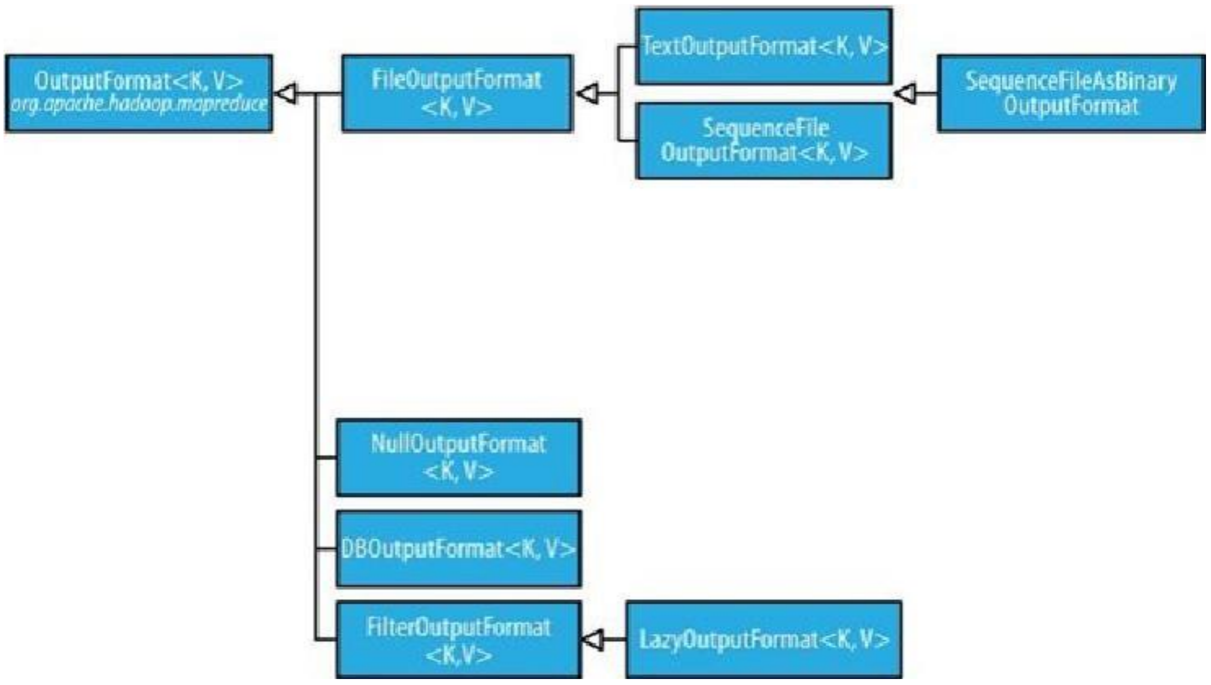


Fig. 3.31 Output File Format Hierarchy

Input / Output Formats

Table 3.2 Input Format vs. Output Format

org.apache.hadoop.mapreduce.lib.input	org.apache.hadoop.mapreduce.OutputFormat.

InputSplit class
in org.apache.hadoop.mapreduce
 	
getLength()
getLocations()	

InputFormat provides the Record Reader class that will generate the series of key/value pairs from a split.	OutputFormat provides theRecordWriter implementation to be used towrite out the output files of the job.
Mapper class's run () method retrieves these key/value pairs from context by callinggetCurrentKey() and getCurrentValue() methods and passes onto map() method for further processing of the record.	checkOutputSpecs() - Based on Output specification, Mapreduce job checks that the output directory RecordWriter<K,V> - used to write out the output files of the job.
TextInputFormat The default InputFormat class	TextOutputFormat It writes records as lines of text.
KeyValueTextInputFormat An InputFormat for plain text files. Files are broken into lines. Each line is divided into key and value parts by a separator byte	MultipleOutputFormat class. Using this format the output data can be written to different output files in Text output format.
FixedLengthInputFormat An input format to read input files with	

fixed length records.	
NLineInputFormat It splits N lines of input as one split which will be fed to a single map task.	NullOutputFormat NullOutputFormat writes nothing to output directory.
CombineFileInputFormat This input file format is suitable for processing huge number of small files. CombineFileInputFormat packs many small files into each split so that each mapper has more to process.	MapFileOutputFormat It is used to write output as Map files.
SequenceFileInputFormat Hadoop specific Binary file format for efficient file processing	SequenceFileOutputFormat This output format class is useful to write out sequence files
SequenceFileAsTextInputFormat which converts the input keys and values to their String forms by calling toString() method	MultipleTextOutputFormat This is also a sub class of MultipleOutputFormat class. Using this format the output data can be written to different output files in Text output format.
SequenceFileAsBinaryInputFormat: It is an input format for reading keys, values from Sequence Files in binary (raw) format	SequenceFileAsBinaryOutputFormat It writes keys and values to Sequence Files in binary format.
MultipleInputs This class supports MapReduce jobs that have multiple input paths with a different InputFormat	MultipleOutputs The MultipleOutputs class is used to write output data to multiple outputs
DBInputFormat A InputFormat that reads input data from an SQL table	DBOutputFormat This output format is used to write an output into SQL tables using mapreduce jobs. DBRecordWriter writes only the key to the database with a batch SQL query

HADOOP COUNTERS

Map Reduce Types and Formats:

What functionalities are provided by MapReduce InputFormat. We will also cover the types of InputFormat in MapReduce, and how to get the data from mapper using InputFormat.

Hadoop InputFormat: describes the input-specification for execution of the Map-Reduce job. InputFormat describes how to split up and read input files. In MapReduce job execution, InputFormat is the first step. It is also responsible for creating the input splits and dividing them into records.

Input files store the data for MapReduce job. Input files reside in **HDFS**. Although these files format is arbitrary, we can also use line-based log files and binary format. Hence, In MapReduce, InputFormat class is one of the fundamental classes which provides below functionality:

- InputFormat selects the files or other objects for input.
- It also defines the Data splits. It defines both the size of individual Map tasks and its potential execution server.
- Hadoop InputFormat defines the RecordReader. It is also responsible for reading actual records from the input files.

Types of InputFormat in MapReduce

There are different types of MapReduce InputFormat in Hadoop which are used for different purpose. Let's discuss the Hadoop InputFormat types below:

1. File Input Format

It is the base class for all file-based Input Formats. File InputFormat also specifies input directory which has data files location. When we start a MapReduce job execution, File InputFormat provides a path containing files to read.

This Inpu Format will read all files. Then it divides these files into one or more Input Splits.

2. Text InputFormat

It is the default InputFormat. This InputFormat treats each line of each input file as a separate record. It performs no parsing. Text InputFormat is useful for unformatted data or line-based records like log files.

- **Key –** It is the byte offset of the beginning of the line within the file (not whole file one split). So it will be unique if combined with the file name.
- **Value –** It is the contents of the line. It excludes line terminators.

3. Key Value Text InputFormat

It is similar to Text InputFormat. This InputFormat also treats each line of input as a separate record. While the difference is that Text InputFormat treats entire line as the value, but the KeyValue Text InputFormat breaks the line itself into key and value by a tab character ('/t').

- **Key –** Everything up to the tab character.
- **Value –** It is the remaining part of the line after tab character.

<u>4. Sequence FileInputFormat</u>

It is an InputFormat which reads sequence files. Sequence files are binary files. These files also store sequences of binary key-value pairs. These are block-compressed and provide direct serialization and deserialization of several arbitrary data. Hence,

Key & Value both are user-defined.

<u>5. Sequence FileAsTextInputFormat</u>

It is the variant of SequenceFileInputFormat. This format converts the sequence file key values to Text objects. So, it performs conversion by calling '**tostring()**' on the keys and values. Hence, Sequence FileAsTextInputFormat makes sequence files suitable input for streaming.

<u>6. Sequence FileAsBinaryInputFormat</u>

By using SequenceFileInputFormat we can extract the sequence file's keys and values as an opaque binary object.

<u>7. Nline InputFormat</u>

It is another form of Text InputFormat where the keys are byte offset of the line. And values are contents of the line. So, each mapper receives a variable number of lines of input with Text InputFormat and KeyValue Text InputFormat.

different than the previous. Each reduce task takes key-value pairs as input and generates key-value pair as output.

Note that shuffling and sorting in Hadoop MapReduce is not performed at all if you specify zero reducers (setNumReduceTasks(0)). Then, the MapReduce job stops at the map phase, and the map phase does not include any kind of sorting (so even the map phase is faster).