Unit III : Hadoop

History of Hadoop- The Hadoop Distributed File System – Components of Hadoop-Analyzing the Data with Hadoop- Scaling Out- Hadoop Streaming-Design of HDFS-Java interfaces to HDFS- Basics Developing a Map Reduce Application-How Map Reduce Works-Anatomy of a Map Reduce Job runFailures-Job Scheduling-Shuffle and Sort – Task execution - Map Reduce Types and Formats- MapReduce Features.

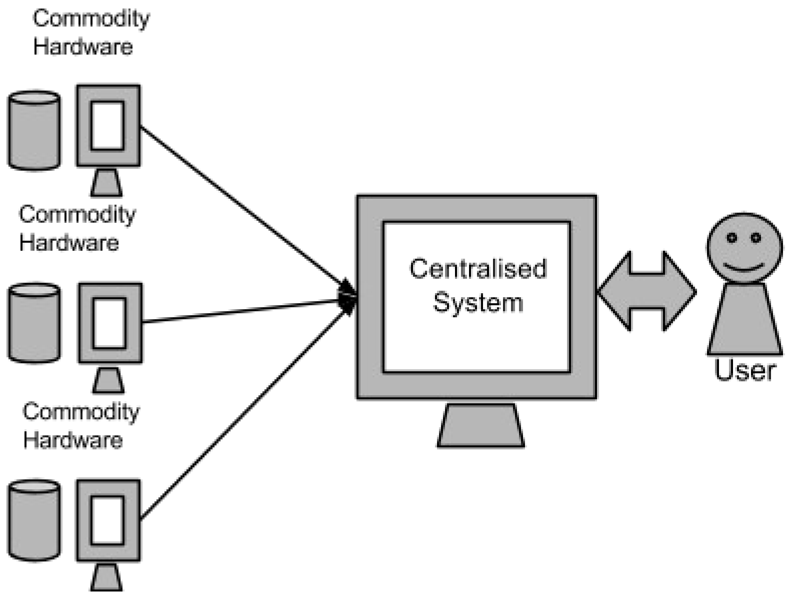
Imagine this scenario: There is 1GB of data that to be processed. The data is stored in a relational database in a desktop computer which has no problem handling the load. Suppose that gradually the data grows to 10GB, then 100GB, reaching the limits of the current desktop computer.

Moreover, applications may deal with unstructured data coming from sources like Facebook, Twitter, RFID readers, sensors, and so on. When it comes to dealing with huge amounts of scalable data, it is a hectic task to process such data through a single database bottleneck.

What should you do? Hadoop may be the answer

Introduction

- Hadoop is an open source project of the Apache Foundation.
- It is optimized to handle massive quantities of data which could be structured, unstructured or semi-structured.
- It is a framework written in Java originally who named it after his son's toy elephant.
- Hadoop uses Google's ***MapReduce*** technology as its foundation.
- The data is processed in parallel with others.
- Uses commodity hardware, that is, relatively inexpensive computers.

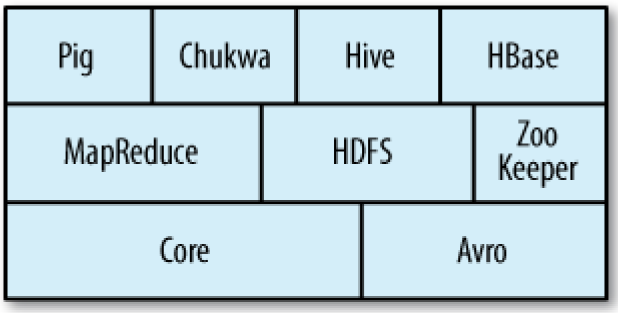


History of Hadoop

- History of Hadoop had started in the year 2002 with the project Apache Nutch.
- Apache Nutch was started in the year 2002 by Doug Cutting which is an effort to build an open source web search engine based on Lucene and Java

Hadoop Framework

- Hadoop is a collection of related subprojects hosted by the Apache Software Foundation.
    - Core
        - A set of components and interfaces for distributed filesystems and general I/O (serialization, Java RPC, persistent data structures).
    - Avro
        - A data serialization system for efficient, cross-language RPC, and persistent data storage.
    - MapReduce
        - A distributed data processing model and execution environment that runs on large clusters of commodity machines.
    - HDFS
        - A distributed filesystem that runs on large clusters of commodity machines.
    - Pig
        - A data flow language and execution environment for exploring very large datasets. Pig runs on HDFS and MapReduce clusters.
    - HBase
        - A distributed, column-oriented database. HBase uses HDFS for its underlying storage, and supports both batch-style computations using MapReduce and point queries (random reads).
    - ZooKeeper
        - A distributed, highly available coordination service. ZooKeeper provides primitives such as distributed locks that can be used for building distributed applications.
    - Hive
        - A distributed data warehouse. Hive manages data stored in HDFS and provides a query language based on SQL for querying the data.
    - Chukwa
        - A distributed data collection and analysis system. Chukwa runs collectors that store data in HDFS, and it uses MapReduce to produce reports.

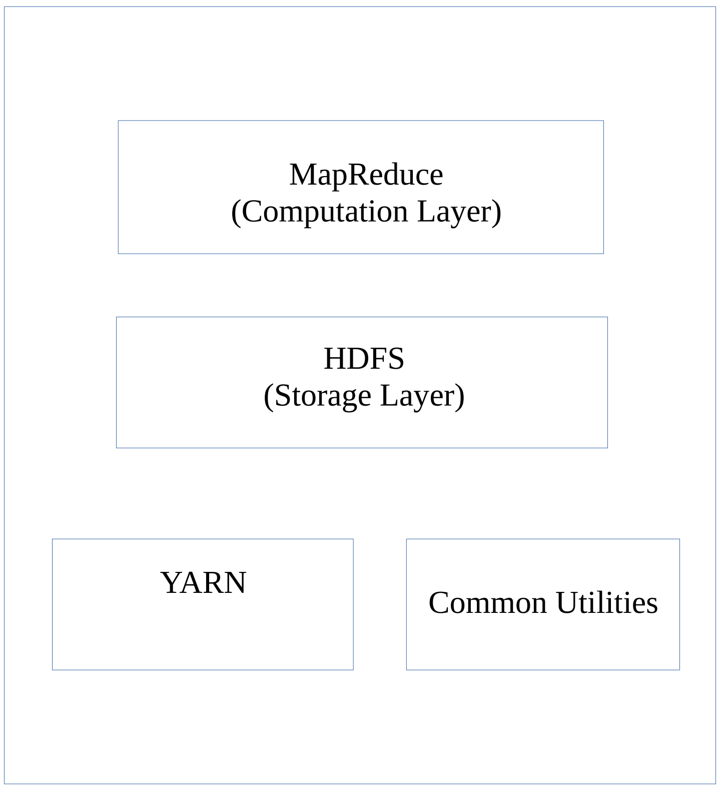


Hadoop Architecture

Hadoop has two major layers namely:

(a) Processing/Computation layer (MapReduce),

(b) Storage layer (Hadoop Distributed File System)



MapReduce

- A parallel programming model.
  A big task is divided into smaller units and assigned to different computer.
- After processing each task, the results are collected at a single site and integrated.

HDFS

- Based on Google File system (GFS).
- Provides a distributed file system that run on commodity hardware.
- **Fault tolerant, since files are replicated.**
- Suitable for applications involving large dataset.
- Deployed on low-cost commodity hardware.

Hadoop YARN

- Framework for job scheduling and cluster resource management.

Hadoop Common Utilities

- Java libraries and utilities required by other Hadoop modules.

Hadoop Working

- Hadoop runs across low-cost clustered machines.
- Initially data is divided into directories and files with uniform size of 128MB or 64MB
- These files are then assigned to different computers for processing.
- HDFS supervises the processing.
- Checks whether the code was executed successfully.
- Performs the sort between Map and Reduce Phase.
- Sending the sorted data to a certain computer.
- Writing the debugging logs for each job.

Advantages of Hadoop

- Hadoop is open source
- Compatible on all the platforms since it is Java based.
- Scalable. Servers can be added to the cluster, without any interruption.
- Utilizes the benefits of parallel processing, by distributing the data and work across machines.
- Hadoop library is designed to detect and handle failures.

MapReduce

- Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets)
  - in-parallel on large clusters of commodity hardware
  - in a reliable, fault-tolerant manner.

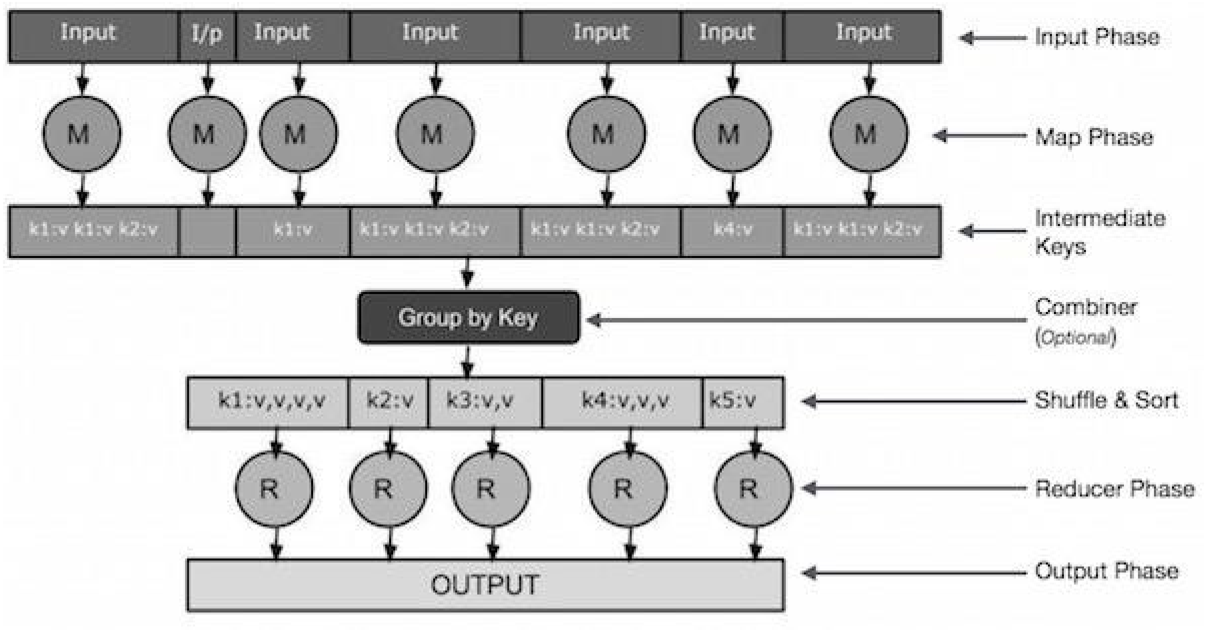
Why MapReduce?

Centralized systems are not suitable for processing huge volumes of data. Traditional database servers are not sufficient for accomodating large datasets. In such cases MapReduce model is well suited, where it processes data parallely by distributing files across machines in a cluster.

Working of MapReduce Algorithm

MapReduce works by dividing the task into two phases:

- Map phase
  - Takes the dataset and the individual elements are broken down into tuples of key-value pairs.
  - A map function is used.
  - The map function can be a simple data preparation phase, setting up the data for the reducer phase.
  - The map function filters the data by dropping missing, suspect or errorneous data.
  - MapReduce framework sorts and groups the key-value pairs of the output of the map function, before sending to the reduce phase.
- Reduce Phase
  - Reduce task is performed after map task.
  - Takes the output from the Map as an input and combines data tuples into a smaller set of tuples.



- Input Phase
  - 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
  - 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.

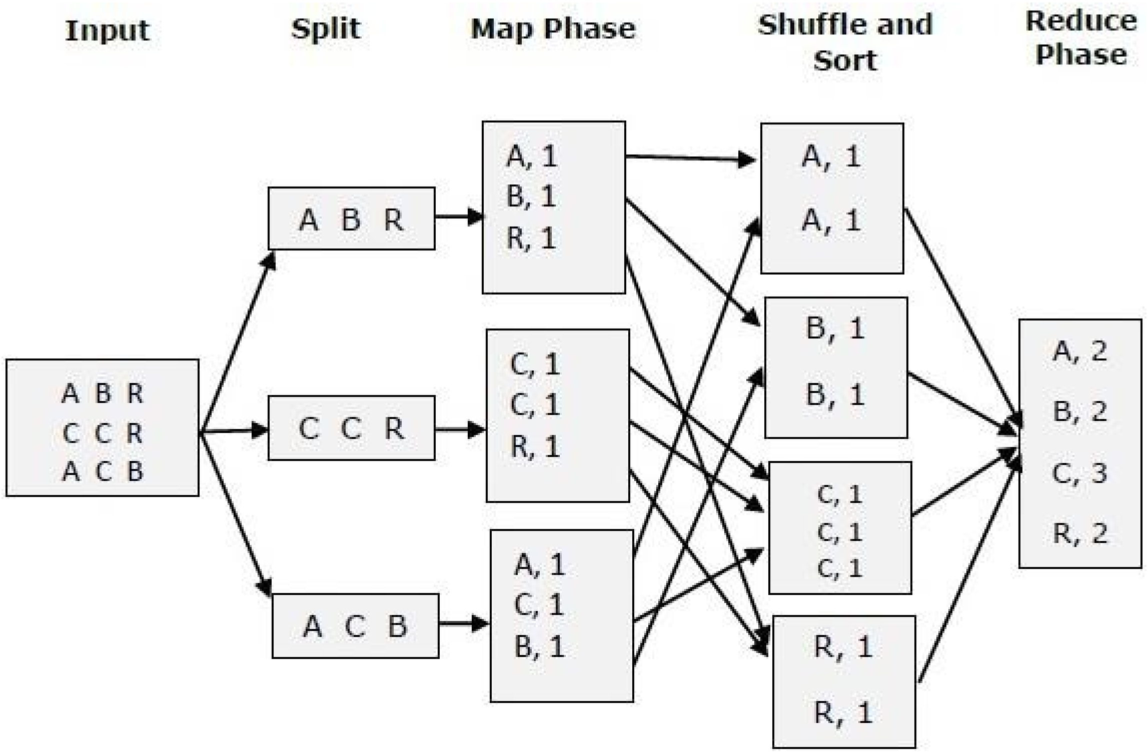
- Combiner Function
  - A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets.
  - It is a user defined function
  - It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one 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**.
  - 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.
    - The data is aggregated, filtered, and combined in a number of ways
    - It requires a wide range of processing.
    - Once the execution is over, it gives zero or more key-value pairs to the final step.

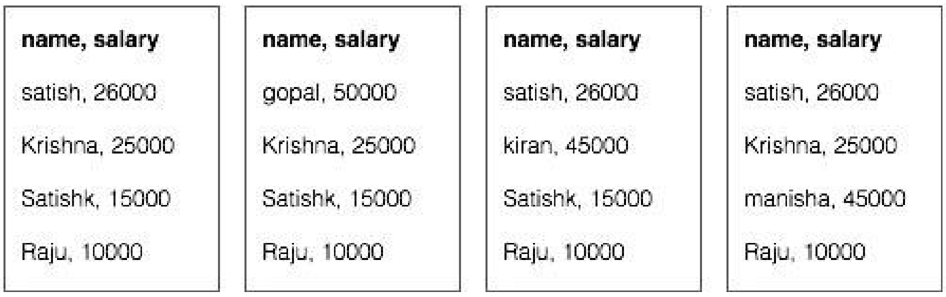
- Output Phase
    - In the output phase, an output formatter translates the final key-value pairs from the Reducer function.
    - Writes them onto a file using a record writer.



MapReduce Example

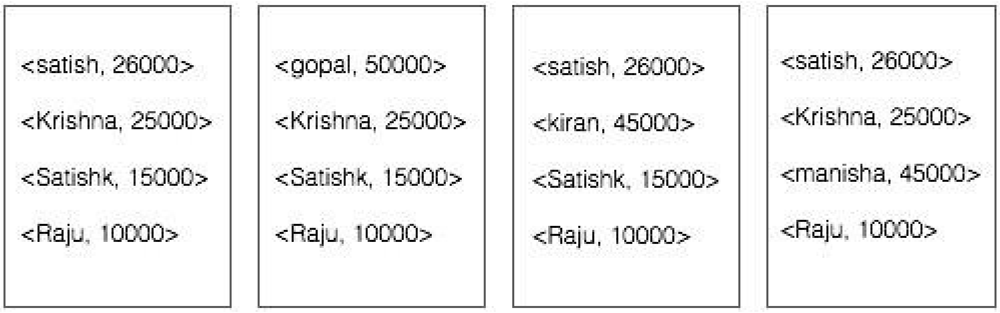
The following example shows how MapReduce employs Searching algorithm to find out the details of the employee who draws the highest salary in a given employee dataset.

- Let us assume we have employee data in four different files − A, B, C, and D.
- Let us also assume there are duplicate employee records in all four files because of importing the employee data from all database tables repeatedly.



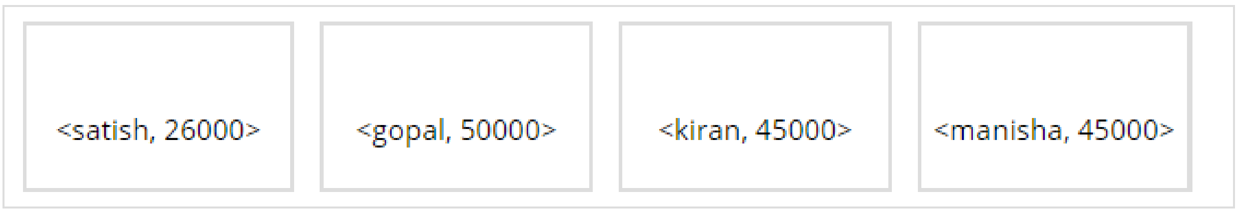
name, salary	**name, salary**	**name, salary**	**name, salary**
satish, 26000	gopal, 50000	satish, 26000	satish, 26000
Krishna, 25000	Krishna, 25000	kiran, 45000	Krishna, 25000
Satishk, 15000	Satishk, 15000	Satishk, 15000	manisha, 45000
Raju, 10000	Raju, 10000	Raju, 10000	Raju, 10000

- The Map phase processes each input file and provides the employee data in key-value pairs (<k, v> : <emp name, salary>).



<satish, 26000>	<gopal, 50000>	<satish, 26000>	<satish, 26000>
<Krishna, 25000>	<Krishna, 25000>	<kiran, 45000>	<Krishna, 25000>
<Satishk, 15000>	<Satishk, 15000>	<Satishk, 15000>	<manisha, 45000>
<Raju, 10000>	<Raju, 10000>	<Raju, 10000>	<Raju, 10000>

- The combiner phase (searching technique) will accept the input from the Map phase as a key-value pair with employee name and salary.
- Using searching technique, the combiner will check all the employee salary to find the highest salaried employee in each file.



<satish, 26000>	<gopal, 50000>	<kiran, 45000>	<manisha, 45000>

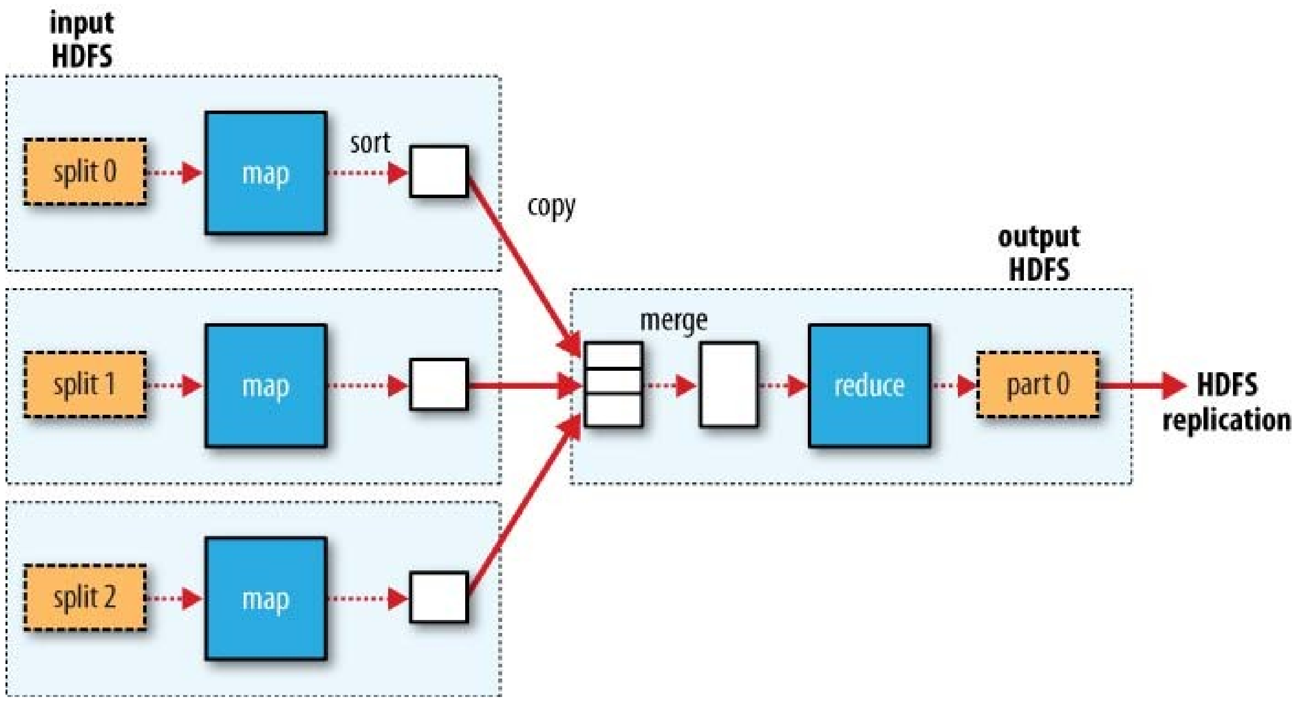
- Reducer phase
  - Form each file, you will find the highest salaried employee.
  - To avoid redundancy, check all the <k, v> pairs and eliminate duplicate entries, if any.
  - The same algorithm is used in between the four <k, v> pairs, which are coming from four input files.
  - The final output will be <gopal, 50000>.

Scaling Out

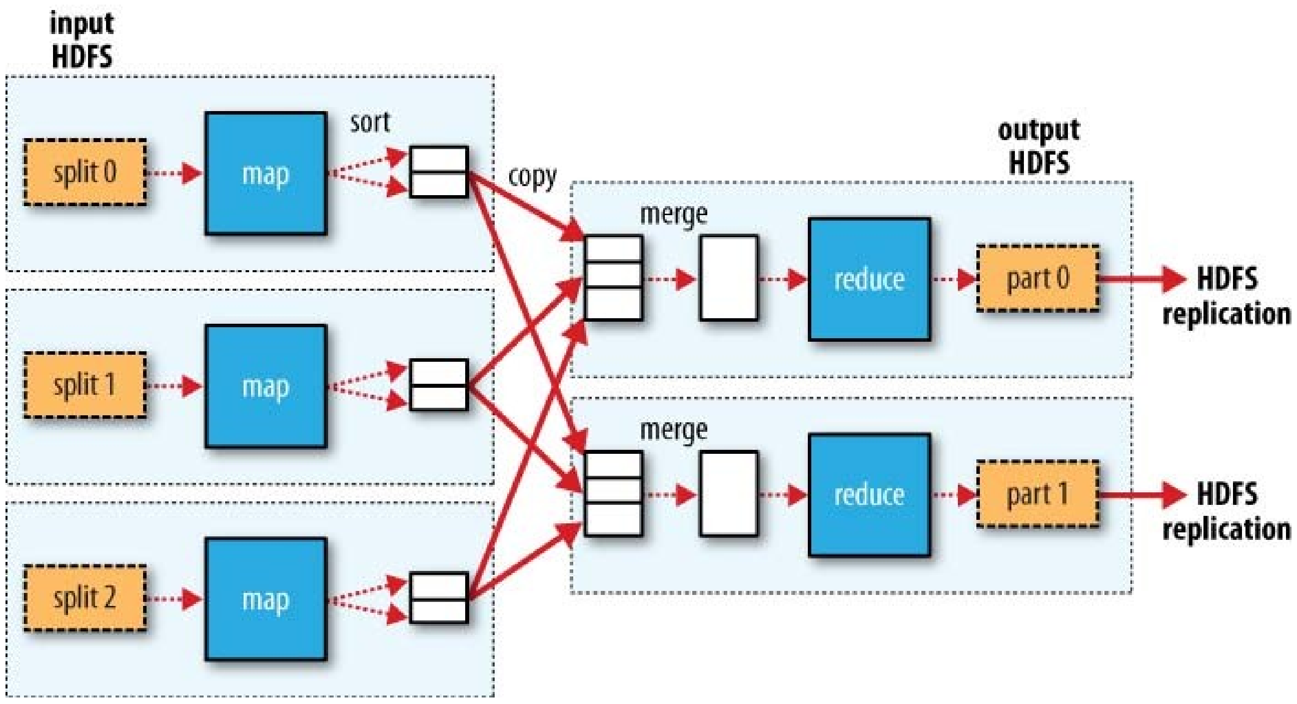
In the above example, the files are assumed to be stored in local file system. However, for applications involving large data sets the data is stored in distributed file system.

Data Flow

- A MapReduce job is a unit of work that the client wants to be performed.
- Hadoop runs the job by dividing it into tasks, of which there are two types: map tasks and reduce tasks.
- There are two types of nodes that control the job execution process:
  - ***Jobtracker***
    - The jobtracker coordinates all the jobs run on the system by scheduling tasks to run on tasktrackers.
  - Number of ***Tasktrackers***
    - Tasktrackers run tasks and send progress re-ports to the jobtracker, which keeps a record of the overall progress of each job.
    - If a tasks fails, the jobtracker can reschedule it on a different tasktracker.
- Hadoop divides the input to a MapReduce job into fixed-size pieces called input ***splits***.



- **Partition Phase**
  - When there are multiple reducers, the map tasks partition their output, each creating one partition for each reduce task.
  - There can be many keys (and their associated values) in each partition, but the records for every key are all in a single partition.
  - The partitioning can be controlled by a user-defined partitioning function.



Hadoop Streaming

- API to MapReduce that allows to write map and reduce functions in languages other than Java.
- Unix standard streams as the interface between Hadoop and application program.
- Any language can be used to read standard input and write to standard output to write to the MapReduce program.
- Suited for text processing.

Hadoop Pipes

- C++ interface to Hadoop MapReduce.
- Pipes uses sockets as the channel over which the tasktracker communicates with the process running the C++ map or reduce function.

HADOOP DISTRIBUTED FILE SYSTEM

When the size of the data set grows out of the storage capacity of single machine, it becomes necessary to partition the data across several machines. Such filesystems that manages storage across a network of machines are called distributed file systems. In most case the files are replicated. If one node fails the file can be loaded from another node.

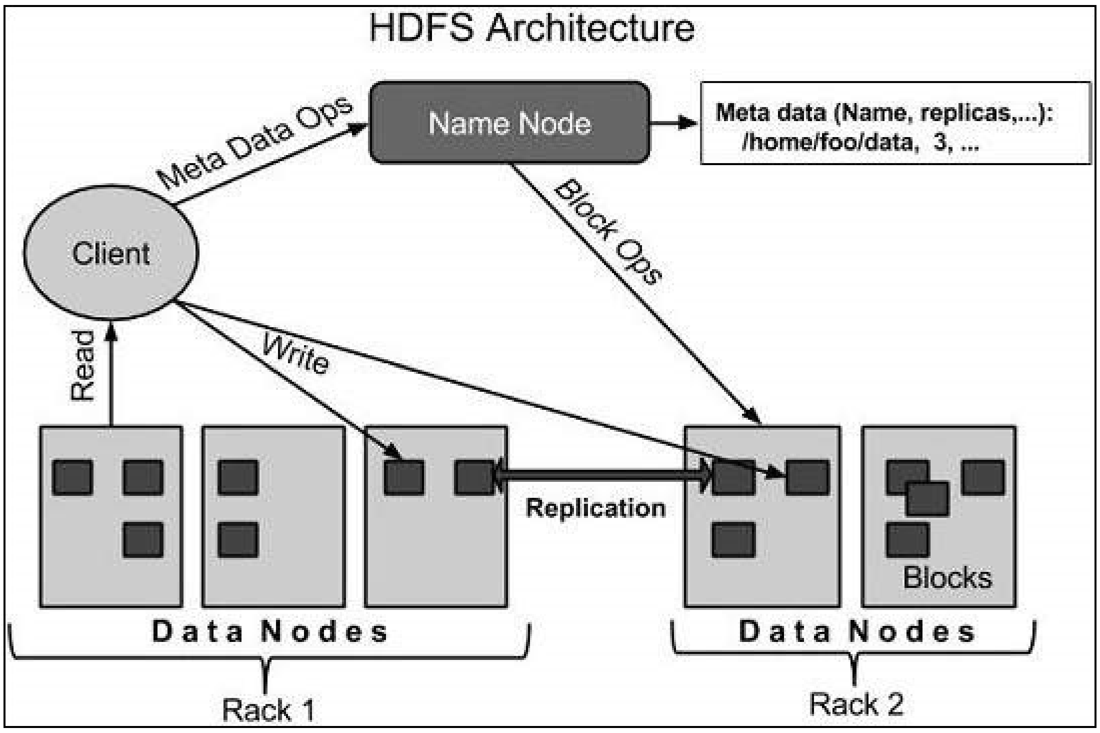
HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters on commodity hardware.

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.
- **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.(Fault tolerance).**

HDFS Concepts

The following figure shows HDFS architecture.



- Namenode
  - The commodity hardware that contains the GNU/Linux operating system and the namenode software.
  - Software that can be run on commodity hardware.
  - The system having the namenode acts as the master server and it does the following tasks −
    - Manages the file system namespace.

        - Regulates client's access to files.
        - It also executes file system operations such as renaming, closing, and opening files and directories.

- Datanode
    - Commodity hardware having the GNU/Linux operating system and datanode software.
    - For every node (Commodity hardware/System) in a cluster, there will be a datanode.
    - These nodes manage the data storage of their system.
        - Datanodes perform read-write operations on the file systems, as per client request.
        - They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

- Block
    - The file in a file system will be divided into one or more segments and/or stored in individual data nodes.
    - These file segments are called as ***blocks***.
    - ***The minimum amount of data that HDFS can read or write is called a Block.***
    - The default block size is 64MB.
    - Files in HDFS are broken into block-sized chunks, which are stored as independent units.
    - HDFS blocks are large compared to disk blocks, to minimize the cost
    - of seeks.

- Secondary Namenode
    - Specially dedicated node in HDFS cluster whose main function is to take checkpoints of the file system metadata present on namenode.
    - It is not a backup namenode.
    - It just checkpoints namenode's file system namespace.

- Active and Passive Namenodes
    - In Hadoop 1.0, there is only one namenode which is a Single point of failure.
    - So if the namenode fails we lose all the information about the datanodes.
    - in Hadoop 2.0, two namenodes were maintained.One active and one Passive or Stand by namenode.
    - Passive node maintains sufficient metadata information to recover from a namenode failover.
    - This is done by maintaining a Shared storage between Active & Passive namenode.

- The File System Namespace
    - HDFS supports a traditional hierarchical file organization.
    - A user or an application can create directories and store files inside these directories.
    - The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file.
    - The NameNode maintains the file system namespace

- Data Replication
    - HDFS stores each file as a sequence of blocks.
    - ***The blocks of a file are replicated for fault tolerance.***
    - An application can specify the number of replicas of a file.
    - The NameNode makes all decisions regarding replication of blocks.
    - A Blockreport contains a list of all blocks on a DataNode.

Goals of HDFS

- **Fault detection and recovery**
    - Since HDFS includes a large number of commodity hardware, failure of components is frequent.
    - Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.
- **Huge datasets**
    - HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
- **Hardware at data**
    - A requested task can be done efficiently, when the computation takes place near the data.
    - Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

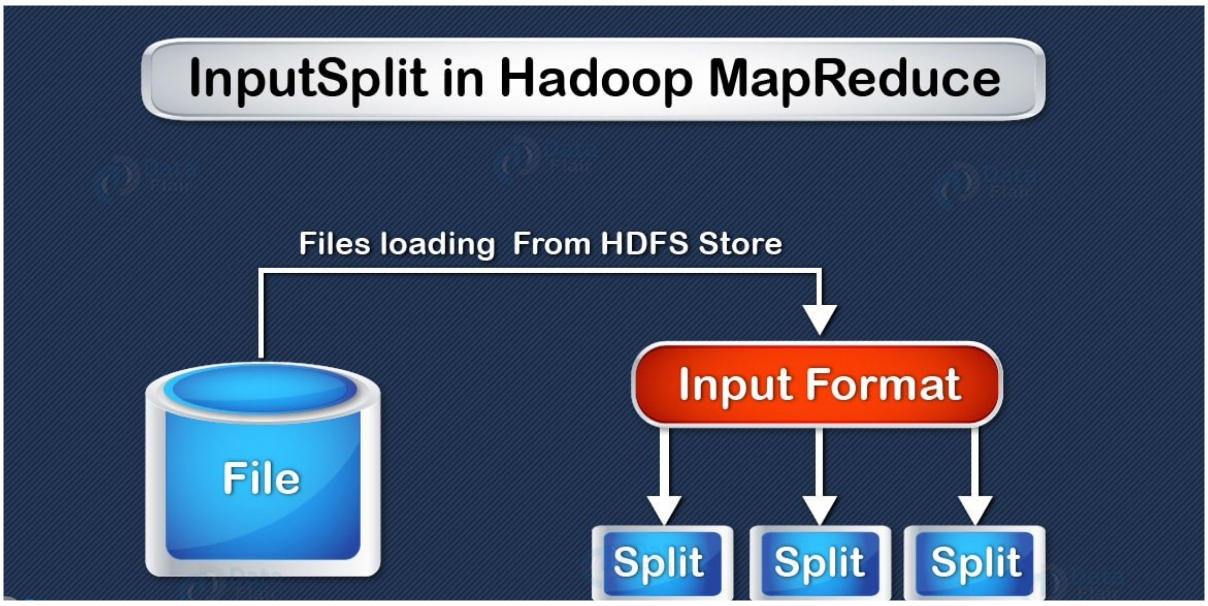
Fault Tolerance in HDFS

- Refers to the working strength of a system in unfavorable conditions and how that system can handle such a situation.
- Before Hadoop 3 it creates a replica of users' data on different machines in the HDFS cluster.
- So whenever if any machine in the cluster goes down, then data is accessible from other machines in which the same copy of data was created.
- Hadoop 3 introduced **Erasure Coding** to provide Fault tolerance.
- Erasure coding is a method used for fault tolerance that durably stores data with significant space savings compared to replication.
- RAID uses Erasure Coding.

- Erasure coding works by striping the file into small units and storing them on various disks.
- For each strip of the original dataset, a certain number of parity cells are calculated and stored.
- If any of the machines fails, the block can be recovered from the parity cell.
- Erasure coding **reduces the storage overhead to 50%.**

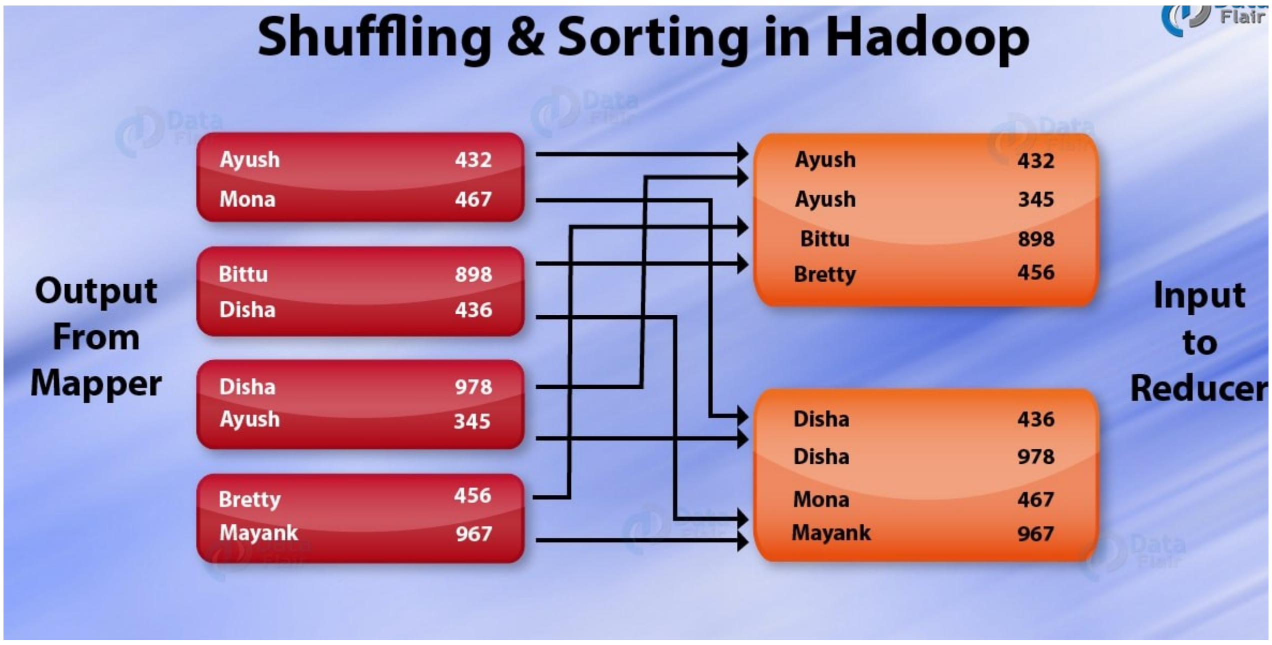
InputSplit in Hadoop MapReduce

- The logical representation of data.
- Describes a unit of work that contains a single map task in a MapReduce program.
- Hadoop InputSplit represents the data which is processed by an individual Mapper.
- The split is divided into records.
- InputSplit length is measured in bytes.
- Every InputSplit has storage locations.
- InputSplit in Hadoop is user defined. User can control split size according to the size of data in MapReduce program. Thus the number of map tasks is equal to the number of InputSplits.
- The client (running the job) can calculate the splits for a job by calling **'getSplit()'**,
- Then sent to the application master, which uses their storage locations to schedule map tasks that will process them on the cluster.



Shuffling and Sorting in Hadoop MapReduce

- In Hadoop, the process by which the intermediate output from mappers is transferred to the reducer is called Shuffling.
- Reducer gets 1 or more keys and associated values on the basis of reducers.
- Intermediate key-value generated by mapper is sorted automatically by key.



Shuffling in MapReduce

- The process of transferring data from the mappers to reducers is known as shuffling
  - i.e. the process by which the system performs the sort and transfers the map output to the reducer as input.
  - MapReduce shuffle phase is necessary for the reducers, otherwise, they would not have any input (or input from every mapper).
  - Shuffling can start even before the map phase has finished so this saves some time and completes the tasks in lesser time.

Sorting in MapReduce

- Before starting of reducer, all intermediate key-value pairs in MapReduce that are generated by mapper get sorted by key.
- Values passed to each reducer are not sorted; they can be in any order.

- Sorting in Hadoop helps reducer to easily distinguish when a new reduce task should start.
- This saves time for the reducer.
- Reducer starts a new reduce task when the next key in the sorted input data is different than the previous.
- Each reduce task takes key-value pairs as input and generates key-value pair as output.

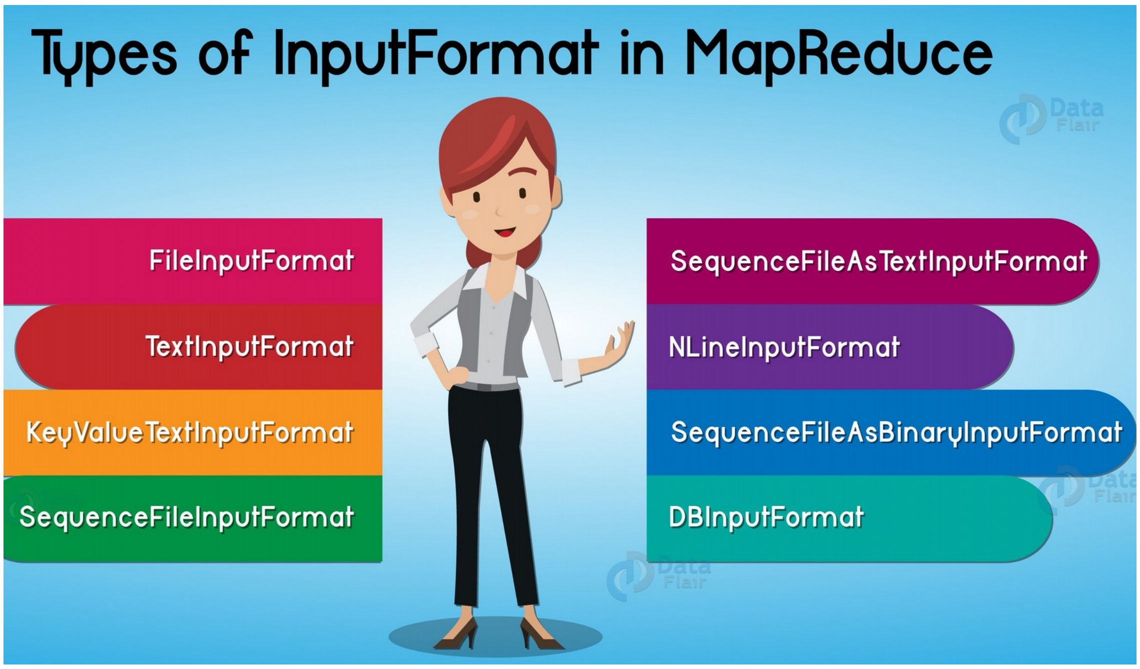
Note that shuffling and sorting in Hadoop MapReduce is not performed at all if there are zero reducers. Then, the MapReduce job stops at the map phase, and the map phase does not include any kind of sorting.

MapReduce Key-Value Pairs

- **Key** – It will be the field/ text/ object on which the data has to be grouped and aggregated on the **reducer** side.
- **Value** – It will be the field/ text/ object which is to be handled by each individual reduce method.

MapReduce-InputFormats

- InputFormat is responsible for creating the input splits and dividing them into records.
- The data for a MapReduce task is stored in input files, which are residing in **HDFS**.
- These files format is arbitrary, line-based log files and binary format can be used.
- Using InputFormat class we define how these input files are split and read.
  - The files or other objects that should be used for input is selected by the InputFormat.
  - InputFormat defines the Data splits, which defines both the size of individual **Map tasks** and its potential execution server.
  - InputFormat defines the **RecordReader**, which is responsible for reading actual records from the input files.

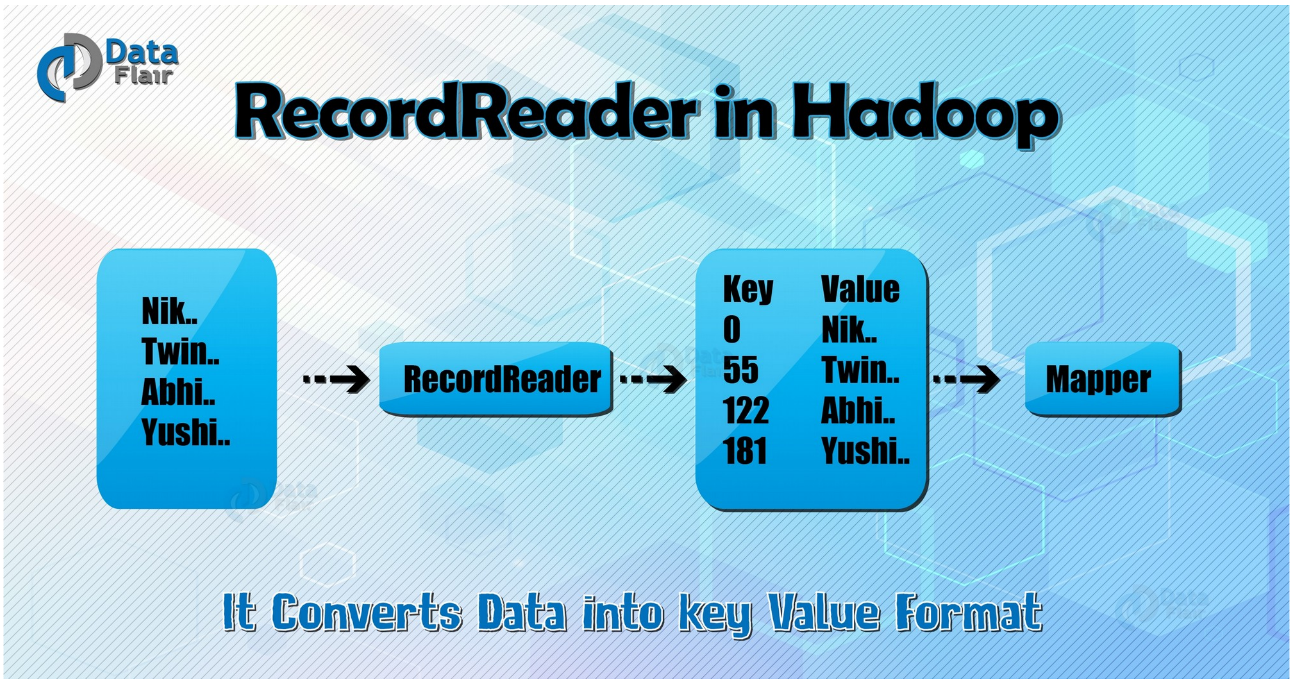


- FileInputFormat
    - Base class for all file-based InputFormats.
    - FileInputFormat specifies input directory where data files are located.
    - When we start a Hadoop job, FileInputFormat is provided with a path containing files to read.
    - FileInputFormat will read all files and divides these files into one or more InputSplits.
- TextInputFormat
    - Default InputFormat of MapReduce.
    - TextInputFormat treats each line of each input file as a separate record and performs no parsing.
- KeyValueTextInputFormat
    - Similar to TextInputFormat.
    - Treats each line of input as a separate record.
    - The KeyValueTextInputFormat breaks the line itself into key and value by a tab character ('/t').
- SequentialFileFormat
    - reads sequence files.
    - Sequence files are binary files that stores sequences of binary **key-value pairs**.
- SequentialFileAsTextInputFormat
    - Another form of SequenceFileInputFormat which converts the sequence file key values to Text objects.
    - The conversion is performed by calling toString() function.
- NlineInputFormat
    - Another form of TextInputFormat.
    - The keys are byte offset of the line and values are contents of the line.
- SequentialFileAsBinaryInputFormat
    - SequenceFileInputFormat using which we can extract the sequence file's keys and values as an opaque binary object.
- DBInputFormat
    - InputFormat that reads data from a relational database, using JDBC.

Hadoop RecordReader

- Converts data into key-value format.

  Hadoop RecordReader uses the data within the boundaries that are being created by the inputsplit and creates Key-value pairs for the mapper.
- The "start" is the byte position in the file where the RecordReader should start generating key/value pairs
- The "end" is where it should stop reading records.
- In Hadoop RecordReader, the data is loaded from its source and then the data is converted into key-value pairs suitable for reading by the Mapper.

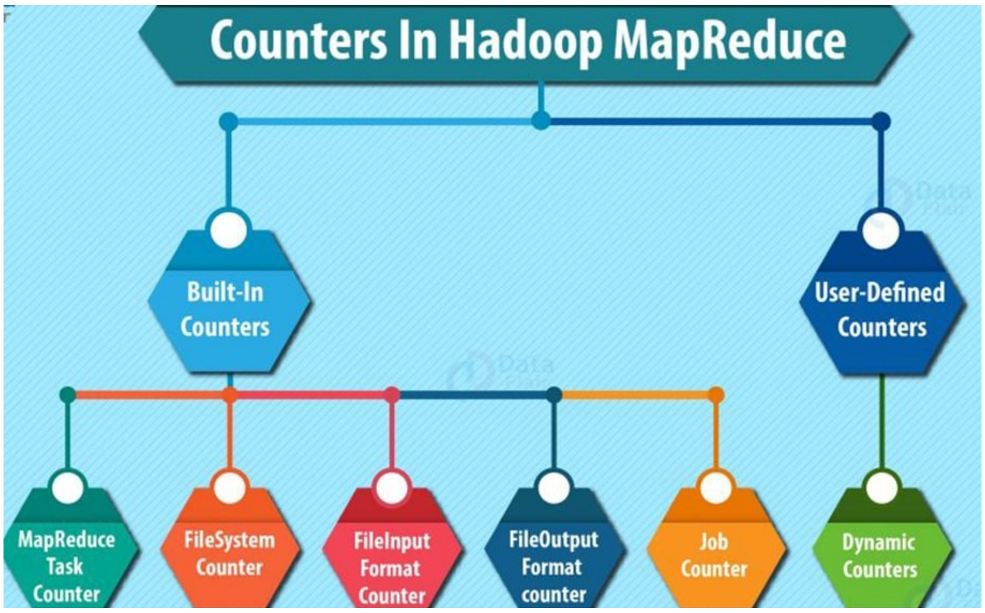


Speculative Execution in Hadoop

- Process that takes place during the slower execution of a task at a node.
- The master node starts executing another instance of that same task on the other node.
- The task which is finished first is accepted and the execution of other is stopped by killing that.
- Backup tasks are called Speculative tasks in Hadoop.
- In a Hadoop cluster with 100s of nodes, problems like hardware failure or network congestion are common.
- Running parallel or duplicate task would be better since there won’t be waiting for the task in the problem to complete.

MapReduce-Counters

- Counters are generally used to keep track of the occurences of events.
- Whenever a MapReduce job is initiated in Hadoop, the Hadoop initiates a counter along with the task.
- This counter is used to keep track of the job statistics like no.of rows read, no.of rows written as output, no. of tasks was launched and successfully ran,the amount of CPU and memory consumed is appropriate for our job and cluster nodes. Etc



- There are two types of counters in Hadoop.
  - Built-in counters
  - User defined counters

- Buit-in Counters
  - these report various metrics, like, there are counters for the number of bytes and records.
  - These metrics are used to confirm that the expected amount of input is consumed and the expected amount of output is produced.

Buit-in Counter	Description
MapReduce Task Counter	collects specific information (like number of records read and written) about tasks during its execution time.
FileSystem Counter	gather information like a number of bytes read and written by the file system.
FileInputFormat Counter	gather information of a number of bytes read by map tasks via FileInputFormat.
FileOutputFormat Counter	gathers information of a number of bytes written by map tasks (for map-only jobs) or reduce tasks via FileOutputFormat.
Job Counter	Measures the job-level statistics. For example, count of the number of map tasks that were launched over the course of a job

- User Defined Counters
    - MapReduce allows user code to define a set of counters, which are then incremented as desired in the mapper or reducer**.**
    - For example, in Java, ‘enum’ is used to define counters.
    - A job may define an arbitrary number of ‘enums’, each with an arbitrary number of fields.
    - The name of the enum is the group name, and the enum’s fields are the counter names.

Hadoop Output Format

- **Reducer** takes as input a set of an intermediate **key-value pair** produced by the mapper.
- Then it runs a reducer function on them to generate output that is again zero or more key-value pairs.
- RecordWriter writes these output key-value pairs from the Reducer phase to output files.
- OutputFormat instances provided by Hadoop are used to write to files on the HDFS or local disk.
- OutputFormat describes the output-specification for a **MapReduce job**.

- TextOutputFormat
    - Default hadoop reducer output format.
    - Writes (key, value) pairs on individual lines of text files.
    - Each key-value pair is separated by a tab character, which can be changed.
    - Keyvaluetextoutputformat is used for reading these output text files.

- Sequencefileoutputformat
    - Writes sequences files for its output.
    - Intermediate format use between mapreduce jobs.
    - Rapidly serialize arbitrary data types to the file.

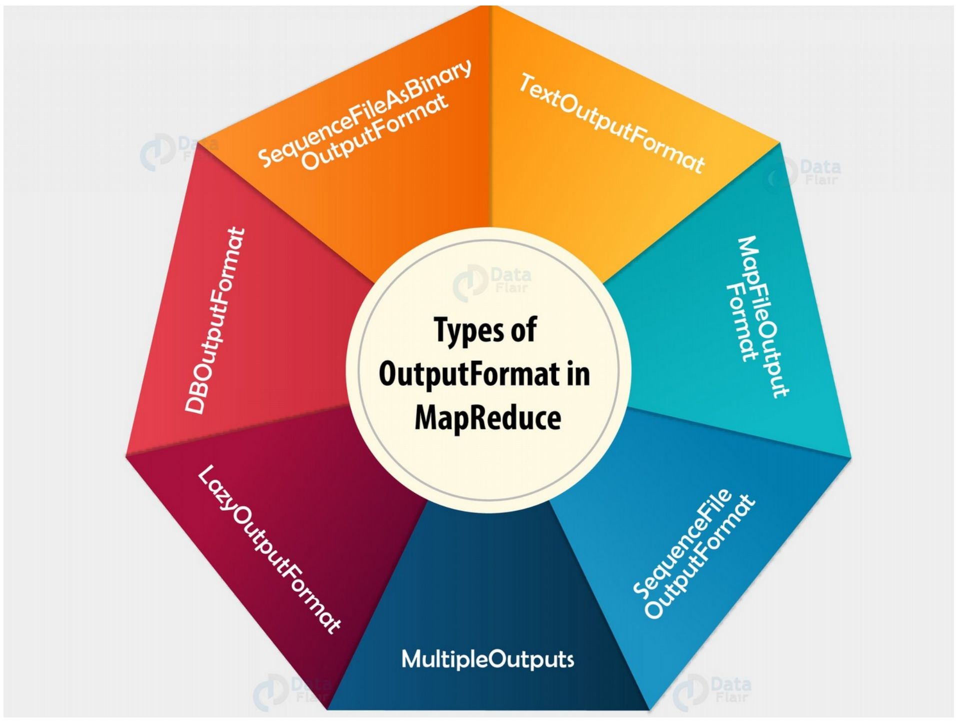
- Sequencefileasbinaryoutputformat
    - Another form of sequencefileinputformat.
    - Writes keys and values to sequence file in binary format.

- MapFileOutputFormat
    - Used to write output as map files.
    - The key in a mapfile must be added in order, ensure that reducer emits keys in sorted order.

- Multipleoutputs
    - Allows writing data to files whose names are derived from the output keys and values.

- Lazyoutputformat
  - Fileoutputformat will create output files, even if they are empty.
  - Lazyoutputformat is a wrapper outputformat which ensures that the output file will be created only when the record is emitted for a given partition.

- DbOutputFormat
  - Output Format for writing to relational databases and **HBase**.
  - It sends the reduce output to a SQL table.
  - It accepts key-value pairs, where the key has a type extending DBwritable.
  - Returned RecordWriter writes only the key to the database with a batch SQL query.


Types of OutputFormat in MapReduce
SequenceFileAsBinary OutputFormat
TextOutputFormat
MapFileOutput Format
SequenceFile OutputFormat
MultipleOutputs
LazyOutputFormat
DBOutputFormat