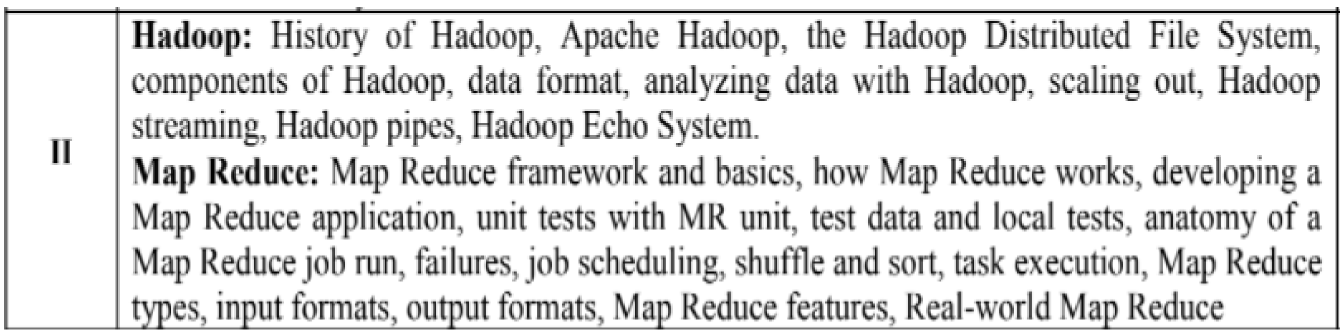


II	**Hadoop:** History of Hadoop, Apache Hadoop, the Hadoop Distributed File System, components of Hadoop, data format, analyzing data with Hadoop, scaling out, Hadoop streaming, Hadoop pipes, Hadoop Echo System. **Map Reduce:** Map Reduce framework and basics, how Map Reduce works, developing a Map Reduce application, unit tests with MR unit, test data and local tests, anatomy of a Map Reduce job run, failures, job scheduling, shuffle and sort, task execution, Map Reduce types, input formats, output formats, Map Reduce features, Real-world Map Reduce

HADOOP

History of Hadoop :

Hadoop is an **open-source software** framework for storing and processing large datasets ranging in size from **gigabytes** to **petabytes**.

Hadoop was developed at the Apache Software Foundation in 2005.

It is written in Java.

The traditional approach like RDBMS is not sufficient due to the heterogeneity of the data.

So Hadoop comes as the solution to the problem of big data i.e. storing and processing the big data with some extra capabilities.

Its co-founder Doug Cutting named it on his son's toy elephant.

There are mainly two components of Hadoop which are :

Hadoop Distributed File System (HDFS)

Yet Another Resource Negotiator(YARN).

In April 2006 Hadoop 0.1.0 was released.

Apache Hadoop :

Hadoop is an **open-source software** framework for storing and processing large datasets ranging in size from **gigabytes** to **petabytes**.

Hadoop was developed at the Apache Software Foundation in 2005.

It is written in Java.

Hadoop is designed to scale up from a single server to thousands of machines, each offering local computation and storage.

Applications built using HADOOP are run on large data sets distributed across clusters of commodity computers.

Commodity computers are cheap and widely available, these are useful for achieving greater computational power at a low cost.

In Hadoop data resides in a distributed file system which is called a Hadoop Distributed File system.

Hadoop Distributed File System :

In Hadoop data resides in a distributed file system which is called a **Hadoop Distributed File system**.

HDFS splits files into blocks and sends them across various nodes in form of large clusters.

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware.

Commodity hardware is cheap and widely available, these are useful for achieving greater computational power at a low cost.

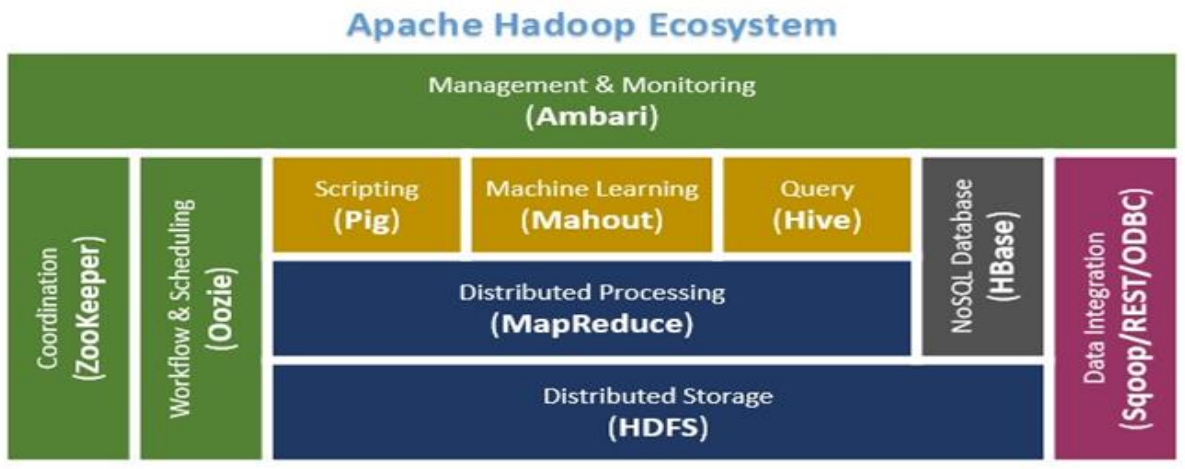
It is highly fault-tolerant and is designed to be deployed on low-cost hardware.

It provides high throughput access to application data and is suitable for applications having large datasets.

Hadoop framework includes the following two modules :

> Hadoop Common: These are Java libraries and utilities required by other Hadoop modules.
>
> Hadoop YARN: This is a framework for job scheduling and cluster resource management.

Hadoop Ecosystem And Components:



There are three components of Hadoop :

Hadoop HDFS -

Hadoop Distributed File System (HDFS) is the storage unit of Hadoop.

HDFS splits files into blocks and sends them across various nodes in form of large clusters.

It is highly fault-tolerant and is designed to be deployed on low-cost hardware.

It provides high throughput access to application data and is suitable for applications having large datasets

Hadoop MapReduce -

Hadoop MapReduce is the processing unit of Hadoop.

MapReduce is a computational model and software framework for writing applications that are run on Hadoop.

These MapReduce programs are capable of processing enormous data in parallel on large clusters of computation nodes.

Hadoop YARN –

Hadoop YARN is a resource management unit of Hadoop.

This is a framework for job scheduling and cluster resource management.

YARN helps to open up Hadoop by allowing to process and run data for batch processing, stream processing, interactive processing and graph processing which are stored in HDFS.

It helps to run different types of distributed applications other than MapReduce.

DATA FORMAT :

A data/file format defines how information is stored in HDFS.

Hadoop does not have a default file format and the choice of a format depends on its use.

The big problem in the performance of applications that use HDFS is the information search time and the writing time.

Managing the processing and storage of large volumes of information is very complex that's why a certain data format is required.

The choice of an appropriate file format can produce the following benefits:

- Optimum writing time
- Optimum reading time
- File divisibility
- Adaptive scheme and compression support

Some of the most commonly used formats of the Hadoop ecosystem are :

● **Text/CSV:** A plain text file or CSV is the most common format both outside and within the Hadoop ecosystem.

● **SequenceFile:** The SequenceFile format stores the data in binary format, this format accepts compression but does not store metadata.

● **Avro:** Avro is a row-based storage format. This format includes the definition of the scheme of your data in JSON format. Avro allows block compression along with its divisibility, making it a good choice for most cases when using Hadoop.

● **Parquet:** Parquet is a column-based binary storage format that can store nested data structures. This format is very efficient in terms of disk input/output operations when the necessary columns to be used are specified.

● **RCFile (Record Columnar File):** RCFile is a columnar format that divides data into groups of rows, and inside it, data is stored in columns.

● **ORC (Optimized Row Columnar):** ORC is considered an evolution of the RCFile format and has all its benefits alongside some improvements such as better compression, allowing faster queries.

Analysing Data with Hadoop :

While the MapReduce programming model is at the heart of Hadoop, it is low-level and as such becomes an unproductive way for developers to write complex analysis jobs.

To increase developer productivity, several higher-level languages and APIs have been created that abstract away the low-level details of the MapReduce programming model.

There are several choices available for writing data analysis jobs.

The Hive and Pig projects are popular choices that provide SQL-like and procedural data flow-like languages, respectively.

HBase is also a popular way to store and analyze data in HDFS. It is a column-oriented database, and unlike MapReduce, provides random read and write access to data with low latency.

MapReduce jobs can read and write data in HBase's table format, but data processing is often done via HBase's own client API.

Scaling In Vs Scaling Out :

Once a decision has been made for data scaling, the specific scaling approach must be chosen.

There are two commonly used types of data scaling :

1. Up
2. Out

Scaling up, or vertical scaling :

It involves obtaining a faster server with more powerful processors and more memory.

This solution uses less network hardware, and consumes less power; but ultimately.

For many platforms, it may only provide a short-term fix, especially if continued growth is expected.

Scaling out, or horizontal scaling :

It involves adding servers for parallel computing.

The scale-out technique is a long-term solution, as more and more servers may be added when needed.

But going from one monolithic system to this type of cluster may be difficult, although extremely effective solution.

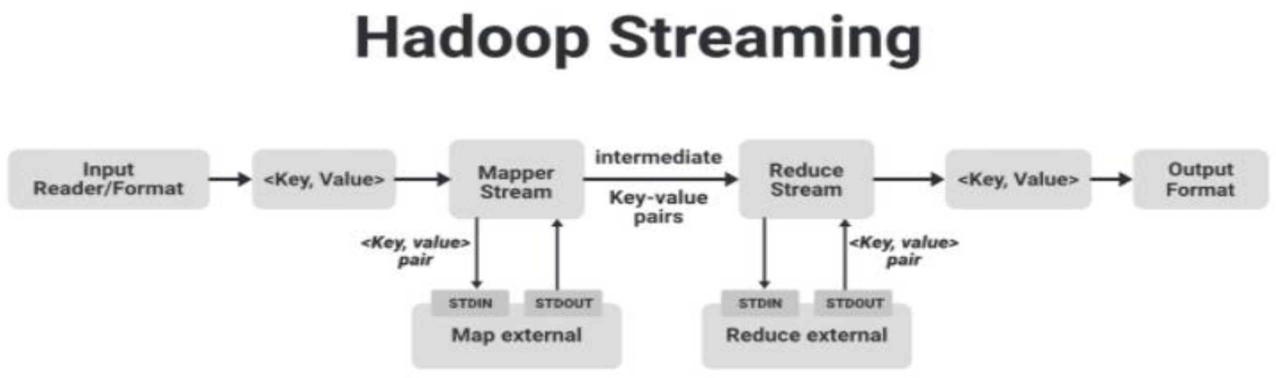
Hadoop Streaming :

It is a feature that comes with a Hadoop distribution that allows developers or programmers to write the Map-Reduce program using different programming languages like Ruby, Perl, Python, C++, etc.

We can use any language that can read from the standard input(STDIN) like keyboard input and all and write using standard output(STDOUT).

Although Hadoop Framework is completely written in java programs for Hadoop do not necessarily need to code in Java programming language.

In the diagram,



We have an **Input Reader** which is responsible for reading the input data and produces the list of key-value pairs. We can read data in .csv format, in delimiter format, from a database table, image data(.jpg, .png), audio data etc.

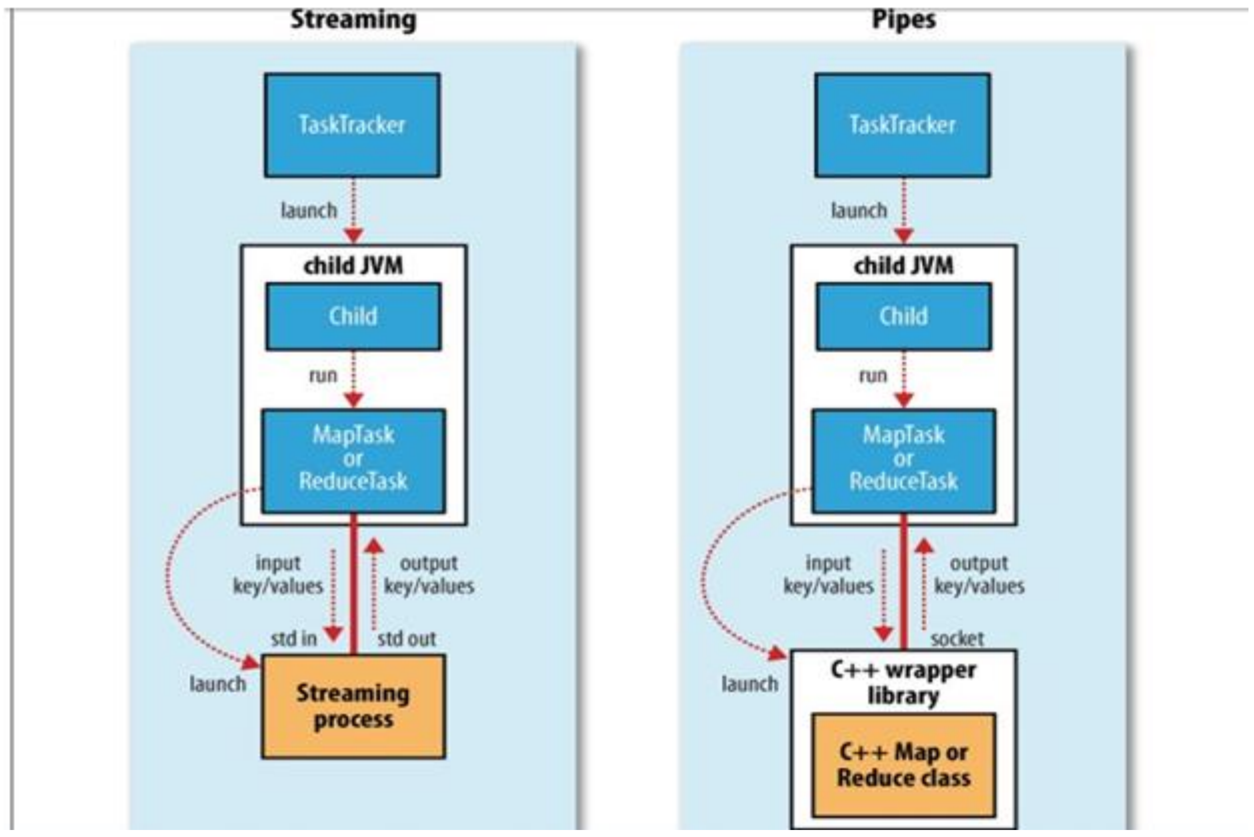
This list of key-value pairs is fed to the **Map phase** and Mapper will work on each of these key-value pair of each pixel and generate some intermediate key-value pairs.

Hadoop Pipes :

Hadoop Pipes is the name of the C++ interface to Hadoop MapReduce.

Unlike Streaming, this uses standard input and output to communicate with the map and reduce code.

Pipes uses sockets as the channel over which the task tracker communicates with the process running the C++ map or reduce function.



After shuffling and sorting, the intermediate key-value pairs are fed to the **Reducer**: then the final output produced by the reducer will be written to the HDFS. These are how a simple Map-Reduce job works.

Map Reduce Framework and Basics :

MapReduce is a software framework for processing data sets in a distributed fashion over several machines.

Prior to Hadoop 2.0, MapReduce was the only way to process data in Hadoop.

A MapReduce job usually splits the input data set into independent chunks, which are processed by the map tasks in a completely parallel manner.

The core idea behind MapReduce is mapping your data set into a collection of < key, value> pairs, and then *reducing* overall pairs with the same key.

The framework sorts the outputs of the maps, which are then inputted to the reduced tasks.

Both the input and the output of the job are stored in a file system.

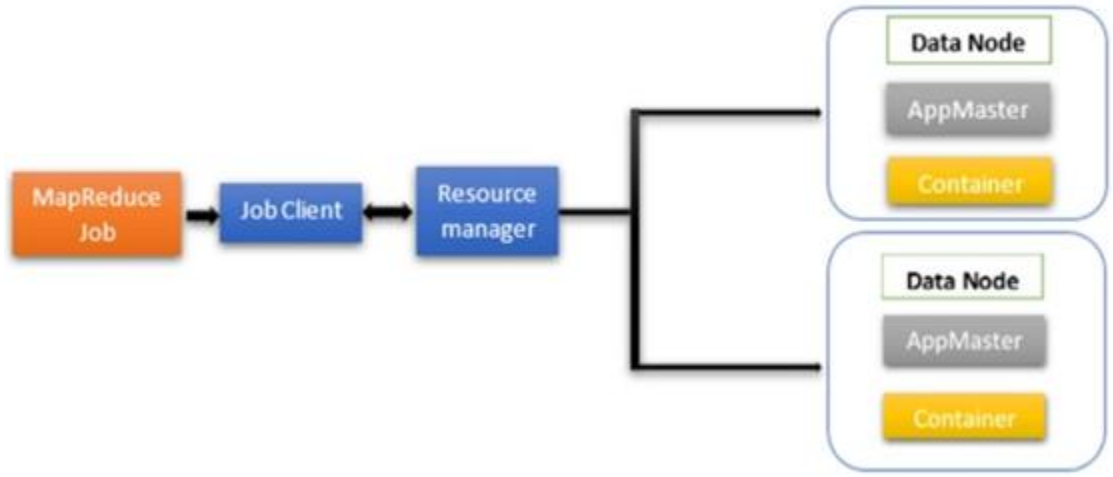
The framework takes care of scheduling tasks, monitors them, and re-executes the failed tasks.

The overall concept is simple :

1. Almost all data can be mapped into pairs somehow, and

2.Your keys and values may be of any type: strings, integers, dummy types and, of course, pairs themselves.

<u>How Map Reduce Works</u> :



Map Reduce contains two core components :

1. Mapper component
2. Reducer component

Uses master-slave architecture.

Storing data in HDFS is low cost, fault-tolerant, and easily scalable.

MapReduce integrates with HDFS to provide the exact same benefits for parallel data processing.

Sends computations where the data is stored on local disks.

Programming model or framework for distributed computing.

It hides complex “housekeeping” tasks from you as a developer.

Developing A Map-Reduce Application :

Writing a program in MapReduce follows a certain pattern.

You start by writing your map and reduce functions, ideally with unit tests to make sure they do what you expect.

Then you write a driver program to run a job, which can run from your IDE using a small subset of the data to check that it is working.

If it fails, you can use your IDE’s debugger to find the source of the problem.

When the program runs as expected against the small dataset, you are ready to unleash it on a cluster.

Running against the full dataset is likely to expose some more issues, which you can fix by expanding your tests and altering your mapper or reducer to handle the new cases.

After the program is working, you may wish to do some tuning :

- First by running through some standard checks for making MapReduce programs faster
- Second by doing task profiling.

Profiling distributed programs are not easy, but Hadoop has hooks to aid in the process.

Before we start writing a MapReduce program, we need to set up and configure the development environment.

Components in Hadoop are configured using Hadoop’s own configuration API.

An instance of the Configuration class represents a collection of configuration properties and their values.

Each property is named by a String, and the type of a value may be one of several, including Java primitives such as boolean, int, long, and float and other useful types such as String, Class, and java.io.File; and collections of Strings.

Developing A Map-Reduce Application :

Writing a program in MapReduce follows a certain pattern.

You start by writing your map and reduce functions, ideally with unit tests to make sure they do what you expect.

Then you write a driver program to run a job, which can run from your IDE using a small subset of the data to check that it is working.

If it fails, you can use your IDE's debugger to find the source of the problem.

When the program runs as expected against the small dataset, you are ready to unleash it on a cluster.

Running against the full dataset is likely to expose some more issues, which you can fix by expanding your tests and altering your mapper or reducer to handle the new cases.

After the program is working, you may wish to do some tuning :

- First by running through some standard checks for making MapReduce programs faster
- Second by doing task profiling.

Profiling distributed programs are not easy, but Hadoop has hooks to aid in the process.

Before we start writing a MapReduce program, we need to set up and configure the development environment.

Components in Hadoop are configured using Hadoop's own configuration API.

An instance of the Configuration class represents a collection of configuration properties and their values.

Each property is named by a String, and the type of a value may be one of several, including Java primitives such as boolean, int, long, and float and other useful types such as String, Class, and java.io.File; and collections of Strings.

<u>Unit Tests With MR Unit</u> :

Hadoop MapReduce jobs have a unique code architecture that follows a specific template with specific constructs.

This architecture raises interesting issues when doing test-driven development (TDD) and writing unit tests.

With MRUnit, you can craft test input, push it through your mapper and/or reducer, and verify its output all in a JUnit test.

As do other JUnit tests, this allows you to debug your code using the JUnit test as a driver.

A map/reduce pair can be tested using MRUnit's MapReduceDriver. , a combiner can be tested using MapReduceDriver as well.

A PipelineMapReduceDriver allows you to test a workflow of map/reduce jobs.

Currently, partitioners do not have a test driver under MRUnit.

MRUnit allows you to do TDD(Test Driven Development) and write lightweight unit tests which accommodate Hadoop's specific architecture and constructs.

Example: We're processing road surface data used to create maps. The input contains both linear surfaces and intersections. The mapper takes a collection of these mixed surfaces as input, discards anything that isn't a linear road surface, i.e., intersections, and then processes each road surface and writes it out to HDFS. We can keep count and eventually print out how many non-road surfaces are input. For debugging purposes, we can additionally print out how many road surfaces were processed.

Anatomy of a Map-Reduce Job Run :

Hadoop Framework comprises of two main components :

- Hadoop Distributed File System (HDFS) for Data Storage
- MapReduce for Data Processing.

A typical Hadoop MapReduce job is divided into a set of Map and Reduce tasks that execute on a Hadoop cluster.

The execution flow occurs as follows:

- Input data is split into small subsets of data.
- Map tasks work on these data splits.
- The intermediate input data from Map tasks are then submitted to Reduce task after an intermediate process called 'shuffle'.
- The Reduce task(s) works on this intermediate data to generate the result of a MapReduce Job.

Job Scheduling :

Early versions of Hadoop had a very simple approach to scheduling users' jobs: they ran in order of submission, using a FIFO scheduler.

Typically, each job would use the whole cluster, so jobs had to wait their turn.

Although a shared cluster offers great potential for offering large resources to many users, the problem of sharing resources fairly between users requires a better scheduler.

Production jobs need to complete in a timely manner while allowing users who are making smaller ad hoc queries to get results back in a reasonable time.

The ability to set a job's priority was added, via the mapred. job.priority property or the setJobPriority() method on JobClient.
When the job scheduler is choosing the next job to run, it selects the one with the highest priority.
However, with the FIFO scheduler, priorities do not support preemption, so a high-priority job can still be blocked by a long-running low priority job that started before the high-priority job was scheduled.
MapReduce in Hadoop comes with a choice of schedulers.
The default is the original FIFO queue-based scheduler, and there are also multiuser schedulers called :

- The Fair Scheduler
- The Capacity Scheduler.

<u>The Fair Scheduler</u> :

The Fair Scheduler aims to give every user a fair share of the cluster capacity over time.
If a single job is running, it gets all of the clusters.
As more jobs are submitted, free task slots are given to the jobs in such a way as to give each user a fair share of the cluster.
A short job belonging to one user will complete in a reasonable time even while another user's long job is running, and the long job will still make progress.
Jobs are placed in pools, and by default, each user gets their own pool.
It is also possible to define custom pools with guaranteed minimum capacities defined in terms of the number of maps and reduce slots, and to set weightings for each pool.
The Fair Scheduler supports preemption, so if a pool has not received its fair share for a certain period of time, then the scheduler will kill tasks in pools running over capacity in order to give the slots to the pool running under capacity.

<u>The Capacity Scheduler</u> :

The Capacity Scheduler takes a slightly different approach to multiuser scheduling.
A cluster is made up of a number of queues (like the Fair Scheduler's pools), which may be hierarchical (so a queue may be the child of another queue), and each queue has an allocated capacity.

This is like the Fair Scheduler, except that within each queue, jobs are scheduled using FIFO scheduling (with priorities).

The Capacity Scheduler allows users or organizations to simulate a separate MapReduce cluster with FIFO scheduling for each user or organization.

The Fair Scheduler, by contrast, enforces fair sharing within each pool, so running jobs share the pool's resources.

Task Execution :

After the task tracker assigns a task, the next step is for it to run the task.

First, it localizes the job JAR by copying it from the shared filesystem to the tasktracker's filesystem.

It also copies any files needed from the distributed cache by the application to the local disk.

Second, it creates a local working directory for the task and un-jars the contents of the JAR into this directory.

Third, it creates an instance of TaskRunner to run the task.

TaskRunner launches a new Java Virtual Machine to run each task so that any bugs in the user-defined map and reduce functions don't affect the task tracker (by causing it to crash or hang, for example).

It is, however, possible to reuse the JVM between tasks.

The child process communicates with its parent through the umbilical interface.

This way it informs the parent of the task's progress every few seconds until the task is complete.

Map Reduce Types :

Hadoop uses the MapReduce programming model for the data processing of input and output for the map and to reduce functions represented as key-value pairs.

They are subject to the parallel execution of datasets situated in a wide array of machines in a distributed architecture.

The programming paradigm is essentially functional in nature in combining while using the technique of map and reduce.

Map Reduce Types :

Mapping is the core technique of processing a list of data elements that come in pairs of keys and values.

The map function applies to individual elements defined as key-value pairs of a list and produces a new list.

The general idea of the map and reduce the function of Hadoop can be illustrated as follows:

map: (K1, V1)-> list (K2, V2)

reduce: (K2, list(V2)) -> list (K3, V3)

The input parameters of the key and value pair, represented by K1 and V1 respectively, are different from the output pair type: K2 and V2.

The reduce function accepts the same format output by the map, but the type of output again of the reduce operation is different: K3 and V3.

The Java API for this is as follows:

```
public interface Mapper<K1, V1, K2, V2> extends JobConfigurable, Closeable {
void map(K1 key, V1 value, OutputCollector<K2, V2> output,Reporter
reporter) throws IOException;
}
public interface Reducer<K2, V2, K3, V3> extends JobConfigurable,Closeable {
void reduce(K2 key, Iterator<V2> values,OutputCollector<K3, V3> output, Reporter
reporter)throws IOException;
}
```

The OutputCollector is the generalized interface of the Map-Reduce framework to facilitate the collection of data output either by the Mapper or the Reducer.

These outputs are nothing but the intermediate output of the job.

Therefore, they must be parameterized with their types.

The reporter facilitates the Map-Reduce application to report progress and update counters and status information.

If the combine function is used, it has the same form as the reduce function and the output is fed to the reduce function.

This may be illustrated as follows:

map: (K1, V1) → list (K2, V2)

combine: (K2, list(V2)) → list (K2, V2)

reduce: (K2, list(V2)) → list (K3, V3)

Note that they combine and reduce functions use the same type, except in the variable names where K3 is K2 and V3 is V2.

The partition function operates on the intermediate key-value types.

It controls the partitioning of the keys of the intermediate map outputs.

The key derives the partition using a typical hash function.

The total number of partitions is the same as the number of reduced tasks for the job.

The partition is determined only by the key ignoring the value.

```
public interface Partitioner<K2, V2> extends JobConfigurable {
int getPartition(K2 key, V2 value, int numberOfPartition);
}
```

Input Format :

Hadoop has to accept and process a variety of formats, from text files to databases.

A chunk of input, called input split, is processed by a single map.

Each split is further divided into logical records given to the map to process in key-value pair.

In the context of a database, the split means reading a range of tuples from an SQL table, as done by the DBInputFormat and producing LongWritables containing record numbers as keys and DBWritables as values.

The Java API for input splits is as follows:

```
public interface InputSplit extends Writable {
long getLength() throws IOException;
String[] getLocations() throws IOException;
}
```

The InputSplit represents the data to be processed by a Mapper.

It returns the length in bytes and has a reference to the input data.

It is the responsibility of the InputFormat to create the input splits and divide them into records.

```
public interface InputFormat<K, V> {
InputSplit[] getSplits(JobConf job, int numSplits) throws IOException;
RecordReader<K, V> getRecordReader(InputSplit split,JobConf
job, throws IOException;
}
```

The JobClient invokes the getSplits() method with an appropriate number of split arguments.

Once the split is calculated it is sent to the jobtracker.

The jobtracker schedules map tasks for the tasktracker using storage location.

The task tracker then passes the split by invoking the getRecordReader() method on the InputFormat to get RecordReader for the split.

The FileInputFormat is the base class for the file data source.

It has the responsibility to identify the files that are to be included as the job input and the definition for generating the split.

Hadoop also includes the processing of unstructured data that often comes in textual format, the TextInputFormat is the default InputFormat for such data.

The SequenceInputFormat takes up binary inputs and stores sequences of binary key-value pairs.

DBInputFormat provides the capability to read data from a relational database using JDBC.

Output Format :

The output format classes are similar to their corresponding input format classes and work in the reverse direction.

For example :

The TextOutputFormat is the default output format that writes records as plain text files, whereas key-values any be of any type, and transforms them into a string by invoking the toString() method. The key-value character is separated by the tab character, although this can be customized by manipulating the separator property of the text output format.

For binary output, there is SequenceFileOutputFormat to write a sequence of binary output to a file. Binary outputs are particularly useful if the output becomes an input to a further MapReduce job.

The output formats for relational databases and to HBase are handled by DBOutputFormat. It sends the reduced output to a SQL table like the HBase's TableOutputFormat enables the MapReduce program to work on the data stored in the HBase table and uses it for writing outputs to the HBase table.

Map Reduce Features :

Features of MapReduce are as follows :

Scalability: Apache Hadoop is a highly scalable framework. This is because of its ability to store and distribute huge data across plenty of servers.

Flexibility : MapReduce programming enables companies to access new sources of data. It enables companies to operate on different types of data.

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.

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.

Fast: 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.

A simple model of programming: One of the most important features is that it is based on a simple programming model.

Parallel Programming: It divides the tasks in a manner that allows their execution in parallel. Parallel processing allows multiple processors to execute these divided tasks.

Availability: 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.

Resilient nature: 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.