4.6 HADOOP FRAMEWORK

"Hadoop is an Apache open source framework ~~written in java~~ that allows distributed processing of large datasets across clusters of computers using simple programming models."

→ A Hadoop frameworked application works in an environment that provides distributed storage and computation across clusters of computers.

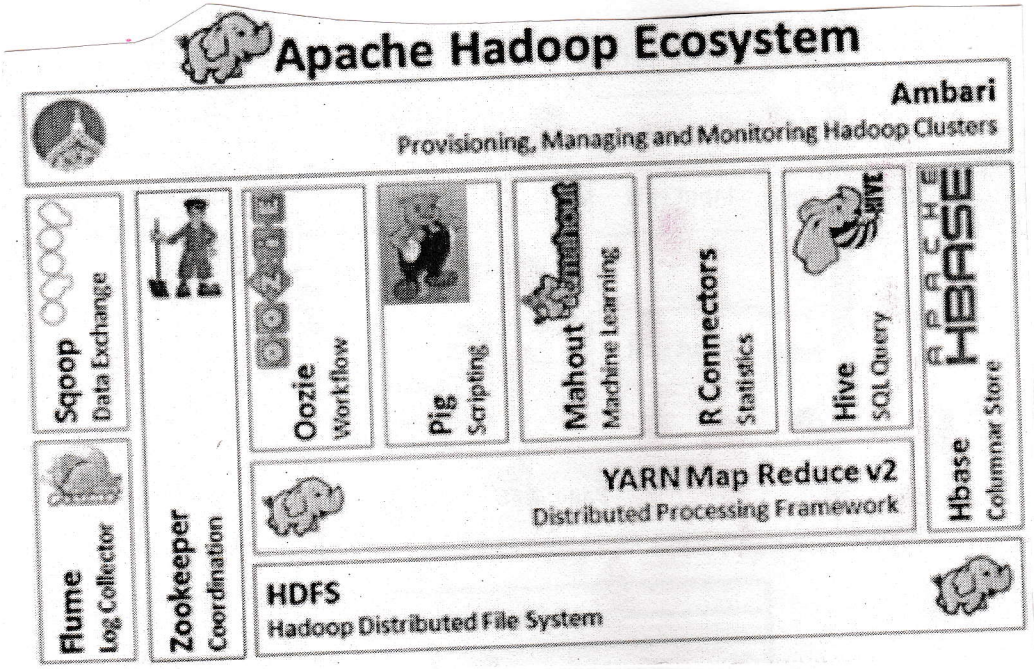


Fig: Hadoop framework.

Some important Frameworks:-

1) HDFS.
2) Hadoop map Reduce.

3) Flume.
4) Sqoop.
5) Zookeeper.
6) Oozie
7) Pig.
8) Mahout.
9) R Connectors.
10) Hive
11) Hbase.

We can discuss about some important types.

4.8.1 HDFS :-

- Hadoop distributed file system is a sub-project of the apache Hadoop project.
- It provides a fault-tolerant file system designed to run on Commodity hardware.
- When HDFS takes in data it breaks the information down into seperate pieces and distributes them to different nodes in a cluster.
- The file system also copies each piece of data multiple times and distributes the copies to individual nodes, placing at least one copy on a different server rack than the others.
- Support applications with large data sets.
- It uses a master/slave architecture.
- Name Node :- Manages file system operations and supporting datanode.
- DataNode :- Manages data storage on individual Compute nodes.

4.6.2 Hadoop MapReduce :-

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

Mapping :- Splits the input data-set into independent chunks which are processed by the map tasks in a Completely parallel manner.

Reducing :- Sorts the outputs of the maps, which are then input to the reduce tasks.

Master JobTracker :- Responsible for scheduling the task, monitoring, reexecuting ....

Slave Job Tracker :- Executes the task as directed by the master.

4.6.3 Flume :-

- Flume is a distributed, reliable and available Service for efficiently Collecting, aggregating, and moving large amount of log data.
- It has a simple and flexible architecture based on streaming data flows.
- It is robust and fault tolerent with tunable reliability mechanism & recovery mechanisms.

- It uses simple extensible model that allows for online analytic application.

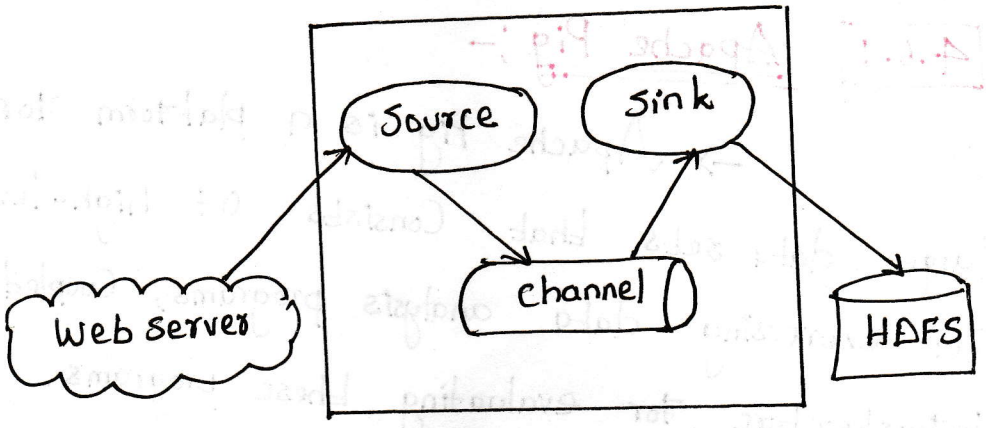


Fig: Apache Flume.

4.6.4 SQOOP :-

- SQOOP is designed to transfer data between hadoop and relational databases or mainframes
- It is used to import data from a relational database management System (RDBMS) such as MySQL or Oracle into the HDFS, transform data in Hadoop MapReduce, and then export back into an RDMS.

4.6.5 Zookeeper :-

- Zookeeper is a Centralized service for maintaining Configuration information, naming, providing distributed synchronization and providing group service.

4.6.6 Oozie :-

- Oozie is a workflow scheduler system to manage apache hadoop jobs.

- Oozie Workflow jobs are Directed Acyclical Graph (DAGs) of actions.

4.6.7 Apache Pig :-

→ Apache Pig is a platform for analyzing large data sets that Consists of high-level language for expressing data analysis programs, Coupled with infrastructure for evaluating these programs.

→ Supports parallelization.

→ Pig's language layer currently Consists of a textual language called Pig Latin.

4.6.8 Apache Mahout :-

→ Mahout is a project of the Apache Software foundation to produce free implementations of distributed or otherwise scalable machine learing algorithms focused primarily in the areas of clustering and classification.

4.6.9 R Connectors :-

→ Oracle R Connector for Hadoop invokes the sqoop utility to Connect to Oracle DB either to extract data or to store results

4.6.10 Hive :-

Hive is used for,

1) Data summarization.

2). query.

3) Analysis.

- It supports queries expressed in a language called HiveQL, which automatically translates SQL like queries into MapReduce jobs executed in Hadoop.

4.6.11 HBase :-

- HBase is an open source, non-relational, distributed database modeled after Google's BigTable and is written in Java.

- Apache HBase is an open source NoSQL database that provides real-time read/write access to those large datasets.

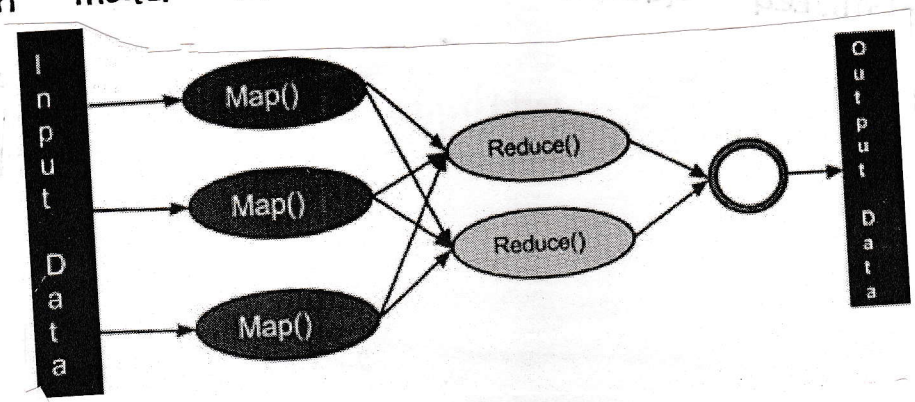
4.7 MAP REDUCE

"MapReduce is a framework using which we can write applications to process huge amount of data, in parallel on large clusters of commodity hardware in a reliable manner.

4.7.1 Map Reduce :-

- MapReduce is a processing technique and a program model for distributed Computing based on java.

- Two important task
  1. Map
  2. Reduce



1. Map :-

- Map takes a set of data and Converts it into another set of data, where individual elements are broken into tuples (key/value pairs).

2. Reduce :-

- It takes the output from a map as an input and Combines those data tuples into a smaller set of tuples.

- The reduce task is always performs after the map job.

- After processing, it produces a new set of output, which will be stored in the HDFS.

* During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.

* Most of the Computing tasks place on nodes which data on local disks that reduces the network traffic.

* After Completion of the given tasks, the cluster Collects and reduces the data to form an appropriate result, and sends it back to the hadoop server.

4.7.2 MapReduce Input & Output:

- MapReduce framework operates on <key, value> pairs, that is the framework views the input to the job as a set of <key, value> pairs and produces as set of <key, value> pairs as the output of the job.

	Input	Output
map	$<k_1, v_1>$	list $(<k_2, v_2>)$
Reduce	$<k_2, list(v_2)>$	list $(<k_3, v_3>)$

4.7.3 Input Splitting :-

- Hadoop divides the input into fixed-size pieces called input splits, or just splits.
- Each split is processed by a single map. Input split represents the data to be processed by an individual Mapper.
- Each split is divided into records, and the map processes each record, which is a key value pair.
- Split is basically a number of rows and € record is that number.
- The input split doesnot Contain input data but a reference to the data.

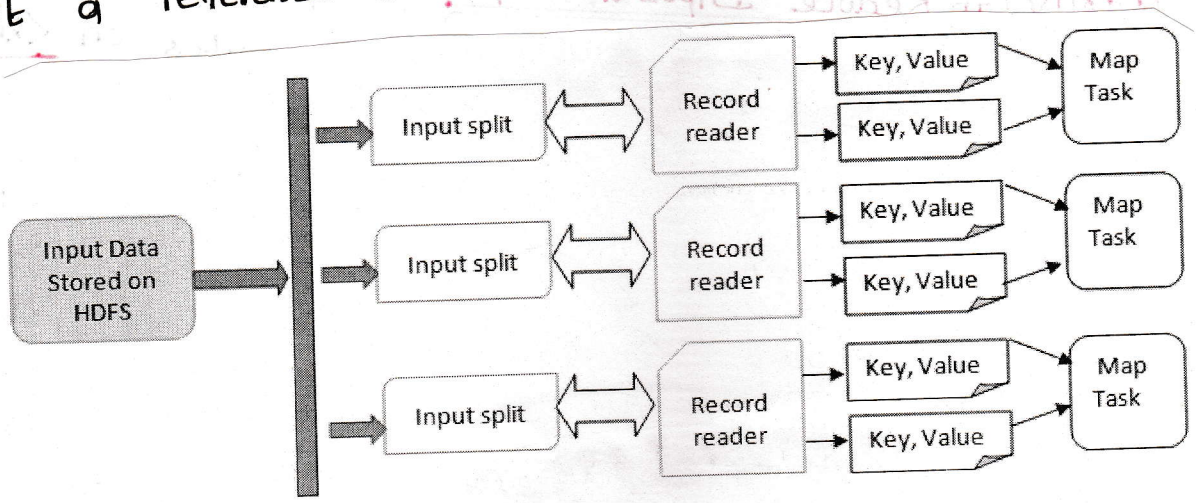


Fig: Input Splitting.

4.7.4 Other Terminology :-

* **NameNode :-** Node that manages the Hadoop Distributed file System (HDFS).

*. **DataNode :-** Node where data is presented in advance before any processing takes place.

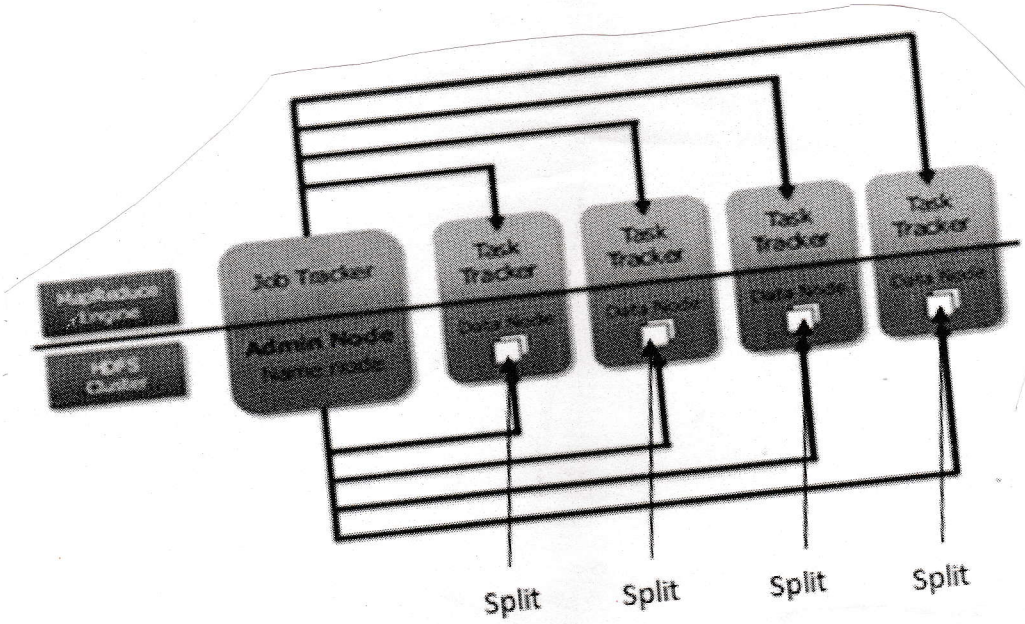


Fig: Job tracker Working Diagram.

* **Master Node :-** Node where JobTracker runs and which accepts job requests from clients.

* **Job Tracker :-** Schedules jobs and tracks the assign job to Task tracker.

*. **Task Tracker :-** Tracks the task and reports status to JobTracker.

Example Program :-

Mapper program:

```
packagecom.tom_e_white.drdobbs.mapreduce;

importorg.apache.hadoop.io.LongWritable;
importorg.apache.hadoop.io.Text;
importorg.apache.hadoop.mapreduce.Mapper;

importjava.io.IOException;

public class ProjectionMapper extends Mapper<LongWritable, Text, Text, LongWritable> {
 private Text word = new Text();
 privateLongWritable count = new LongWritable();

 @Override
 protected void map(LongWritable key, Text value, Context context)
    throwsIOException, InterruptedException {
  // value is tab separated values: word, year, occurrences, #books, #pages
  // we project out (word, occurrences) so we can sum over all years
  String[] split = value.toString().split("\t+");
  word.set(split[0]);
  if (split.length>= 2) {
   try {
    count.set(Long.parseLong(split[2]));
    context.write(word, count);
   } catch (NumberFormatException e) {
    // cannot parse - ignore
   }
  }
 }
}
```

Running through our tiny dataset, the map output looks like this:

```
("dobbs", 20)
("dobbs", 22)
("doctor", 545525)
("doctor", 668666)
```

In our abstract representation, the input to the reduce step looks like this:

```
("dobbs", [20, 22])
("doctor", [545525, 668666])
```

Reducer:

```
packageorg.apache.hadoop.mapreduce.lib.reduce;

importjava.io.IOException;
importorg.apache.hadoop.io.LongWritable;
importorg.apache.hadoop.mapreduce.Reducer;

public class LongSumReducer<KEY> extends Reducer<KEY, LongWritable,
                                    KEY,LongWritable> {

 privateLongWritable result = new LongWritable();

 public void reduce(KEY key, Iterable<LongWritable> values,
                    Context context) throws IOException, InterruptedException {
  long sum = 0;
  for (LongWritableval : values) {
   sum += val.get();
  }
  result.set(sum);
  context.write(key, result);
 }

}
```

The output of the reducer will be:

```
("dobbs", 42)
("doctor", 1214191)
```