Paradigms for Managing Large Scale Dataset

Content

- Comparison of Bigdata
- Introduction to Hadoop
- MapReduce
- Hadoop Ecosystem
- Apache Hive, Pig, Querying large datasets paradigms
- Distributed databases
- Distributed hash tables

Comparison

Small Data

- Mostly structured
- Store in MB, GB, TB
- Increase gradually
- Locally present, centralized data
- Oracle, SQL server
- Single node

Big Data

- Unstructured, 3Vs
- Store in PB, EB
- Increase exponentially
- Globally present, distributed
- Hadoop, Spark
- Multi node cluster

- More Data usually beats better algorithms

Introduction to Hadoop

- Internet started growing 2001.
- Before that data stored in rows and columns (Structured data)
- More solutions were available for structured data. As size is limited, thus it can store in single storage device required and single processing elements to process
- But as internet starts growing, different types of data change. As we use such as image, video, text, email are available , thus volume of data increased and thus to process , it needs new solutions.
- Doug cutting started working in 2002 and it made public in 2008 by Yahoo and Apache foundation in 2012.

HD File System- Data nodes

- Rather than storing data on single node, Store data on different different nodes distributedly and then we can access parallelly.

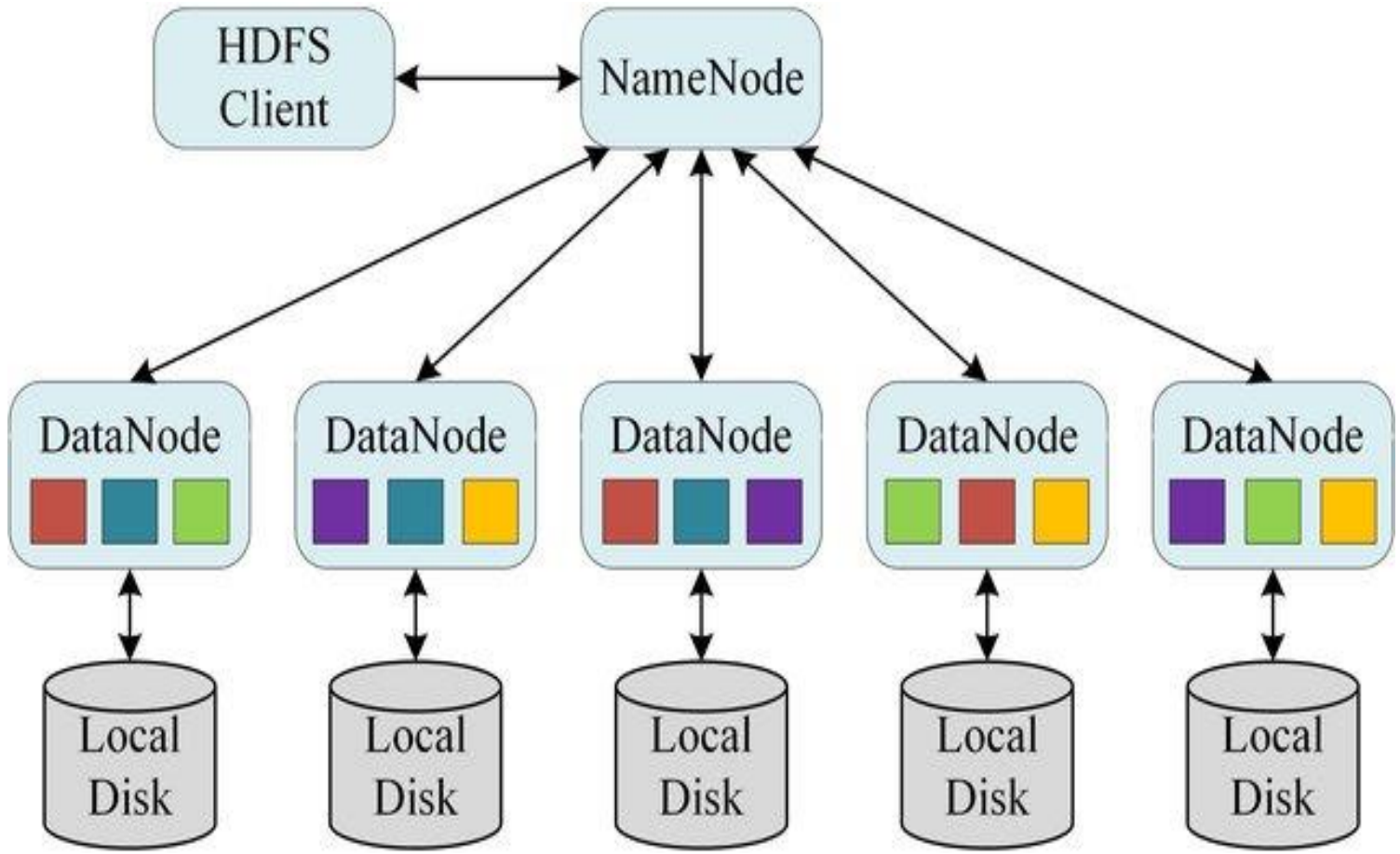
- Hadoop is a processing element.

- Use of Java Programming Languages

Introduction to Hadoop

- Hadoop is an open source framework that allows us to store and process large data sets in a parallel and distributed manner.
- Doug Cutting started the Hadoop project.
- Two main components, Hadoop and MapReduce
- Hadoop Distributed file system (HDFS) is the primary data storage system used by Hadoop applications.
- MapReduce is the processing unit of Hadoop.

HDFS


HDFS
Client
NameNode
DataNode
DataNode
DataNode
DataNode
DataNode
Local
Disk
Local
Disk
Local
Disk
Local
Disk
Local
Disk

HDFS

- Rack awareness
- It is a physical collection of 30-40 nodes under 1 rack.

- Data store locally, but copies store remotely.

Functions of Name Node

- It records the metadata of all files stored in the cluster, e.g. location of blocks stored, the size of the files, permissions.

- Two files associated woth metatdata:
  - FsImage: It contains complete state of the file system namespace since the start of the NameNode.
  - EditLogs: It contains all the recent modifications made to the file system.

- It regularly receives a communication messages and block report from all the data nodes on the cluster to ensure that DataNodes are live.

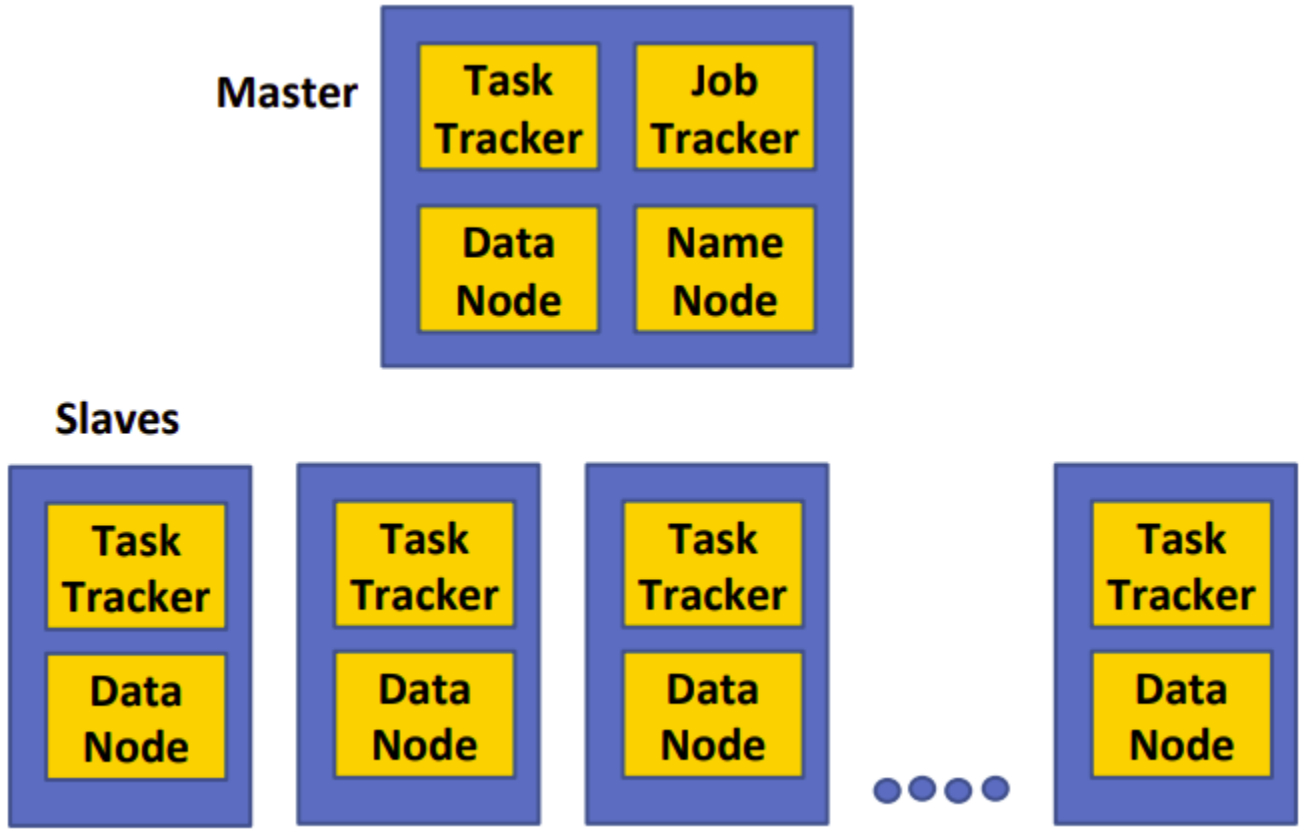
Functions of Data Node

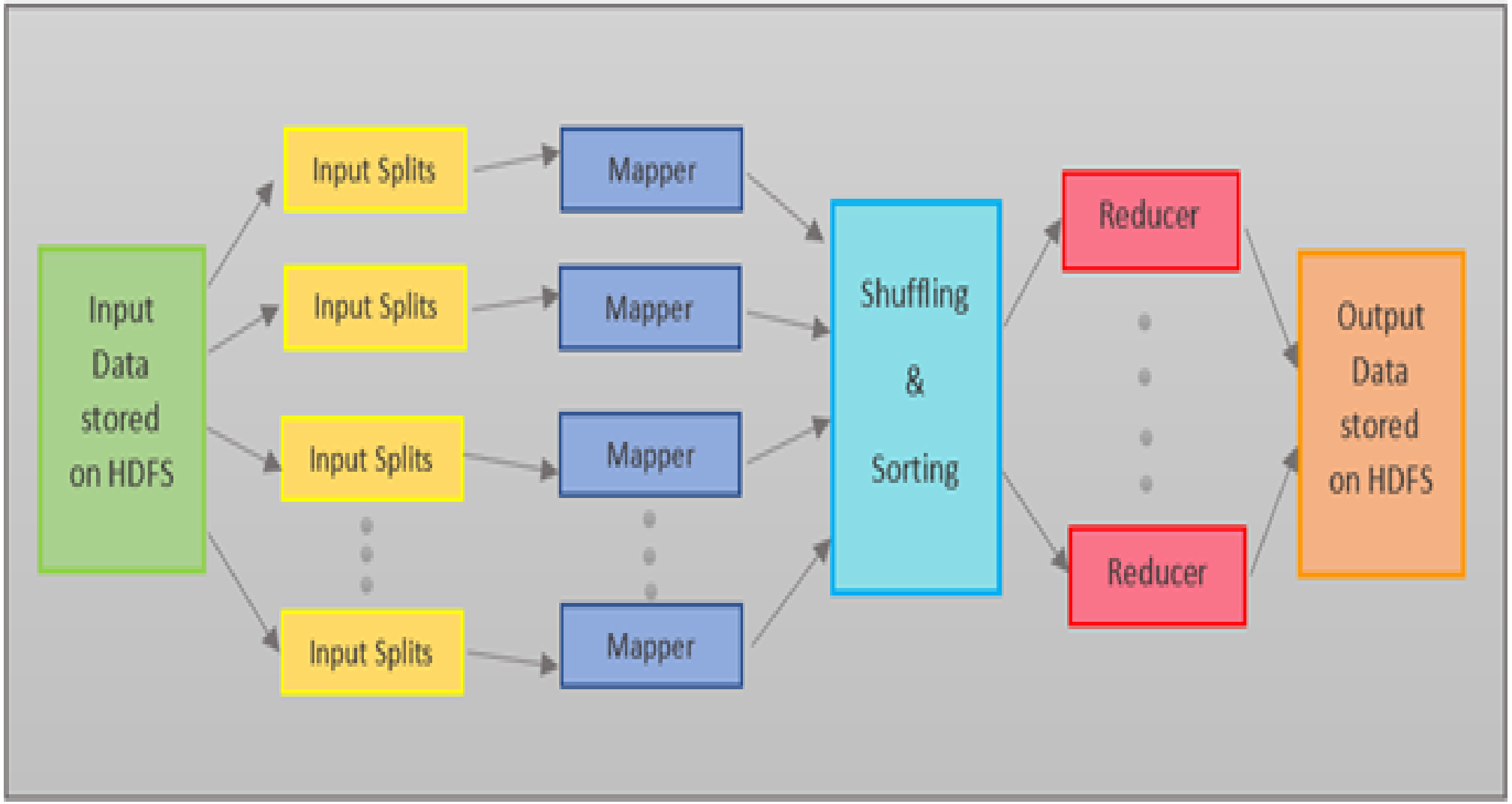
- The actual data is stored on DataNodes.
- The DataNodes perform the low-level read and write requests from the file system’s client.
- Write operation is expensive on HDFS.

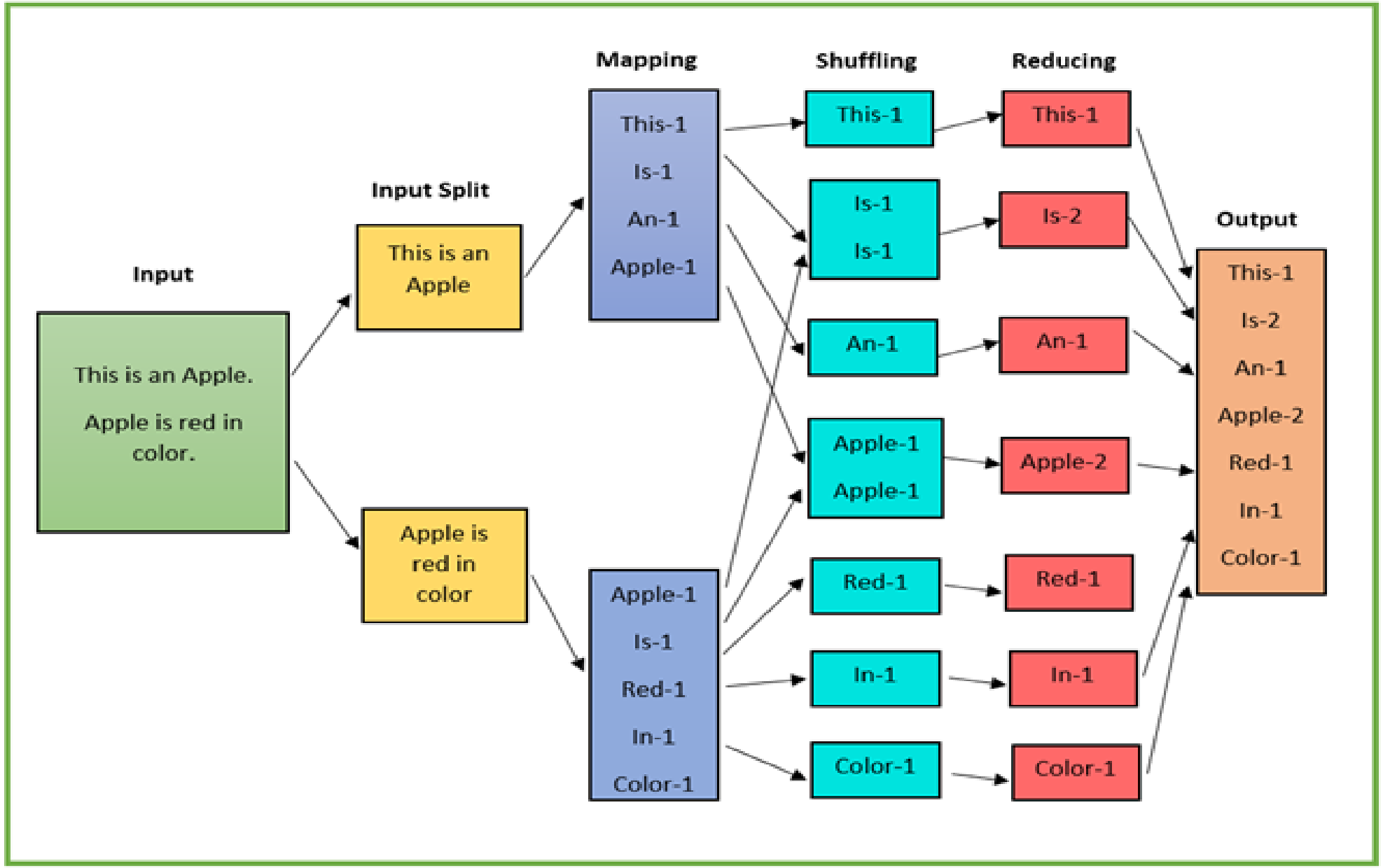
MapReduce

- MapReduce perform the processing of large data sets in a distributed and parallel manner.
- MapReduce consists of main two tasks- map and reduce
- Two main daemons of MapReduce- Job Tracker and Task Tracker

Map Reduce


Master
Task Tracker
Job Tracker
Data Node
Name Node
Slaves
Task Tracker
Data Node
Task Tracker
Data Node
Task Tracker
Data Node
Task Tracker
Data Node


Input Data stored on HDFS
Input Splits
Input Splits
Input Splits
Input Splits
Mapper
Mapper
Mapper
Mapper
Shuffling & Sorting
Reducer
Reducer
Output Data stored on HDFS


Input
This is an Apple.
Apple is red in color.
Input Split
This is an Apple
Apple is red in color
Mapping
This-1
Is-1
An-1
Apple-1
Apple-1
Is-1
Red-1
In-1
Color-1
Shuffling
This-1
Is-1
Is-1
An-1
Apple-1
Apple-1
Red-1
In-1
Color-1
Reducing
This-1
Is-2
An-1
Apple-2
Red-1
In-1
Color-1
Output
This-1
Is-2
An-1
Apple-2
Red-1
In-1
Color-1

Querying all your data

- For example, **Mailtrust, Rackspace's mail division**, used Hadoop for processing email logs. One ad hoc query they wrote was to **find the geographic distribution of their users**.
- In their words:
- This data was so useful that we've scheduled the MapReduce job to run monthly and we will be using this data to help us decide which Rackspace data centers to place new mail servers in as we grow.
- By bringing several hundred gigabytes of data together and having the tools to analyze it, the Rackspace engineers were able to gain an understanding of the data that they otherwise would never have had, and furthermore, they were able to use what they had learned to improve the service for their customers.

MapReduce

- **MapReduce** is a programming model for data processing.
- Hadoop can run MapReduce programs written in various languages; such as Java, Ruby, and Python.
- Most importantly, MapReduce programs are inherently parallel.
- **A weather dataset :** a program that mines weather data. Weather **sensors collect data every hour at many locations across the globe and gather a large volume of log data,** which is a good candidate for analysis with MapReduce because we want to process all the data, and the data is semi-structured and record-oriented.

A Weather Dataset: Format of a National Climatic Data Center record

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

Analysing data with traditional unix tools

- Awk script took 42 minutes in one run on a single machine.
- To speed up the processing, we need to run parts of the program in parallel.
- We could process different years in different processes, using all the available hardware threads on a machine. There are a few problems with this, however.

- **First**, dividing the work into equal-size pieces isn't always easy or obvious. In this case, the file size for different years varies widely, so some processes will finish much earlier than others. Even if they pick up further work, the whole run is dominated by the longest file.
- A better approach, although one that requires more work, **is to split the input into fixed-size chunks and assign each chunk to a process.**

Analysing data with traditional unix tools

- **Second**, combining the results from independent processes may require further processing.
- In this case, the result for each year is independent of other years, and they may be combined by concatenating all the results and sorting by year. **If using the fixed-size chunk approach, the combination is more delicate. For this example, data for a particular year will typically be split into several chunks, each processed independently**.
- We'll end up with the maximum temperature for each chunk, so the final step is to look for the highest of these maximums for each year.

Analysing data with traditional unix tools

- **Third**, you are still limited by the processing capacity of a single machine. If the best time you can achieve is 20 minutes with the number of processors you have, then that's it. You can't make it go faster. Also, some datasets grow beyond the capacity of a single machine.
- When we start using multiple machines, a whole host of other factors come into play, mainly falling into the categories of coordination and reliability.
- Who runs the overall job? How do we deal with failed processes?
- **So, although it's feasible to parallelize the processing, in practice it's messy.** Using a framework like Hadoop to take care of these issues is a great help.

Analyzing the Data with Hadoop

- To take advantage of the parallel processing that Hadoop provides, we need to express our query as a MapReduce job.
- MapReduce works by breaking the processing into two phases: the map phase and the reduce phase.
- Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.

Analyzing the Data with Hadoop

- The input to our map phase is the raw NCDC data. We choose a text input format that gives us each line in the dataset as a text value. The key is the offset of the beginning of the line from the beginning of the file, but as we have no need for this, we ignore it. Our map function is simple. We pull out the year and the air temperature, because these are the only fields we are interested in.
- In this case, the map function is just a data preparation phase, setting up the data in such a way that the reduce function can do its work on it: finding the maximum temperature for each year. The map function is also a good place to drop bad records: here we filter out temperatures that are missing, suspect, or erroneous.

Analyzing the Data with Hadoop

To visualize the way the map works, consider the following sample lines of input data

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

These lines are presented to the map function as the key-value pairs:

```
(0, 0067011990999991950051507004…9999999N9+00001+99999999999…)
(106, 0043011990999991950051512004…9999999N9+00221+99999999999…)
(212, 0043011990999991950051518004…9999999N9-00111+99999999999…)
(318, 0043012650999991949032412004…0500001N9+01111+99999999999…)
(424, 0043012650999991949032418004…0500001N9+00781+99999999999…)
```

Example

- The keys are the line offsets within the file, which we ignore in our map function. **The map function merely extracts the year and the air temperature** and emits them as its output (the temperature values have been interpreted as integers):

```
(1950, 0)
(1950, 22)
(1950, −11)
(1949, 111)
(1949, 78)
```

- The output from the map function is processed by the MapReduce framework before being sent to the reduce function.

- This processing sorts and groups the key-value pairs by key. So, continuing the example, our reduce function sees the following input:

```
(1949, [111, 78])
(1950, [0, 22, −11])
```

- Each year appears with a list of all its air temperature readings. All the reduce function has to do now is iterate through the list and pick up the maximum reading:

```
(1949, 111)
(1950, 22)
```

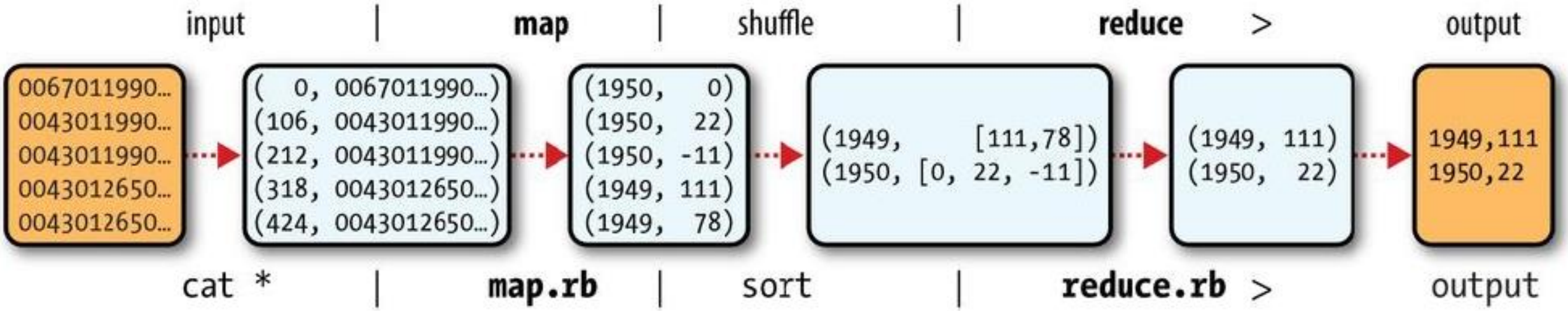


Figure 2-1. MapReduce logical data flow

Mapper for the maximum temperature example

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper

extends Mapper<LongWritable, Text, Text, IntWritable> {

        private static final int MISSING = 9999;

        @Override

        public void map(LongWritable key, Text value, Context context)

        throws IOException, InterruptedException {

                String line = value.toString();

                String year = line.substring(15, 19);

                int airTemperature;

if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs

                        airTemperature =
Integer.parseInt(line.substring(88, 92));

                }

else {

                        airTemperature =
Integer.parseInt(line.substring(87, 92));

                }

                String quality = line.substring(92, 93);

                if (airTemperature != MISSING &&
quality.matches("[01459]")) {

                        context.write(new Text(year), new
IntWritable(airTemperature));

                }

        }

}
```

Mapper Function

Mapper Class:

- Generic type with four formal type parameters.
- Specifies input key, input value, output key, and output value types for the map function.
- For the example:
  - Input key: long integer offset,
  - Input value: line of text,
  - Output key: year,
  - Output value: air temperature (integer).

Hadoop Basic Types:

- Uses Hadoop's own basic types .
- Found in the `org.apache.hadoop.io` package.
- Example types:
  - `LongWritable` (corresponds to Java Long),
  - `Text` (similar to Java String),
  - `IntWritable` (similar to Java Integer).

Mapper Function

map() Method:

- Takes a key and a value as parameters.
- Converts the `Text` value (containing the input line) into a Java `String`.
- Uses `substring()` method to extract the columns of interest from the line.

Context in map() Method:

- Provides an instance of `Context` to write the output.
- Wraps the temperature in an `IntWritable`.
- Outputs a record only if the temperature is present and the quality code indicates a valid temperature reading.

Reducer for the maximum temperature example

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class MaxTemperatureReducer
        extends Reducer<Text, IntWritable, Text,
IntWritable> {
        @Override
        public void reduce(Text key, Iterable<IntWritable>
values, Context context)
                throws IOException, InterruptedException {
                int maxValue = Integer.MIN_VALUE;
                for (IntWritable value : values) {
                        maxValue = Math.max(maxValue,
value.get());
                }
                context.write(key, new IntWritable(maxValue));
        }
}
```

Reducer Function

- Reducer Class:
  - Four formal type parameters specify input and output types for the reduce function.
- Input Types for Reduce Function:
  - **Must match the output types of the map function**.
  - Example input types: `Text` and `IntWritable`.
- Output Types for Reduce Function:
  - Example output types: `Text` and `IntWritable`.
  - Represents a year and its maximum temperature.
- Finding Maximum Temperature:
  - Iterates through temperatures.
  - Compares each temperature with the current record of the highest found so far.
  - Determines the maximum temperature for each year.

Application to find the maximum temperature in the weather dataset

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Application Class

- Job Object:
  - Specifies and controls how the job is run.
  - Packages code into a JAR file for distribution across the Hadoop cluster.
  - JAR file location is inferred using the class specified in the `setJarByClass()` method.
- Input and Output Paths:
  - Input path specified by `addInputPath()` method on `FileInputFormat`.
  - Can be a single file, directory (containing all files), or file pattern.
  - Output path specified by `setOutputPath()` method on `FileOutputFormat`.
  - Specifies the directory for the output files from the reduce function.
- Precaution for Output Path:
  - Output directory should not exist before running the job to prevent accidental data loss.

Application Class

- Map and Reduce Classes:
  - Map and reduce types specified by `setMapperClass()` and `setReducerClass()` methods.
  - Output types for the reduce function controlled by `setOutputKeyClass()` and `setOutputValueClass()` methods.
- Map Output Types:
  - Map output types default to the same types as the reducer.
  - If different, set using `setMapOutputKeyClass()` and `setMapOutputValueClass()` methods.
- Input Types:
  - Input types controlled by the input format (using the default `TextInputFormat` in this case).

Application Class

- Running the Job:
  - `waitForCompletion()` method on Job submits the job and waits for completion.
  - Verbose output flag controls whether progress information is displayed.
- Return Value:
  - `waitForCompletion()` returns a Boolean indicating success (true) or failure (false).
  - Translated into the program's exit code (0 for success, 1 for failure).

- Output:
  - The output was written to the output directory, which contains one output file per reducer. The job had a single reducer, so we find a single file, named part-r-00000:

Basic File handling in HDFS

Command	Description	Usage Example
`hdfs dfs -ls`	List files and directories in a directory.	`hdfs dfs -ls /user/<username>/`
`hdfs dfs -mkdir`	Create a directory in HDFS.	`hdfs dfs -mkdir /user/<username>/new_directory`
`hdfs dfs -put`	Copy files from the local filesystem to HDFS.	`hdfs dfs -put local_file.txt /user/<username>/`
`hdfs dfs -get`	Copy files from HDFS to the local filesystem.	`hdfs dfs -get /user/<username>/file.txt local_dir`
`hdfs dfs -cat`	Display the contents of a file in HDFS.	`hdfs dfs -cat /user/<username>/file.txt`
`hdfs dfs -cp`	Copy files within HDFS.	`hdfs dfs -cp /user/<username>/file.txt /user/<username>/new_location/`
`hdfs dfs -mv`	Move files within HDFS.	`hdfs dfs -mv /user/<username>/old_location/file.txt /user/<username>/new_location/`
`hdfs dfs -rm`	Remove files or directories from HDFS.	`hdfs dfs -rm /user/<username>/file.txt`
`hdfs dfs -du`	Display disk usage of files and directories.	`hdfs dfs -du -h /user/<username>/`
`hdfs dfs -chmod`	Change permissions of files or directories.	`hdfs dfs -chmod 755 /user/<username>/file.txt`
`hdfs dfs -chown`	Change owner of files or directories.	`hdfs dfs -chown <new_owner>:<new_group> /user/<username>/file.txt`
`hdfs dfs -setrep`	Set the replication factor for a file.	`hdfs dfs -setrep 3 /user/<username>/file.txt`
`hdfs dfsadmin -report`	Display a summary of the HDFS cluster.	`hdfs dfsadmin -report`

Hadoop Framework

- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
- It is designed to scale up from single servers to thousands of machines, each offering local computation and storage

HDFS Master Slave Architecture

- Read and Write operation diagram
- Hadoop Ecosystem: Hive, Pig, Sqoop, Oozie, HBase, Flume

Hive and Pig

- Pig is a data flow language for exploring very large datasets.
- Hive is a data warehouse for managing data stored in HDFS and provides a query language based on SQL.

Apache Hive

- Architecture of Hive
- Embedded and Remote Metastore mechanism and their Purpose
- Working of Hive with Hadoop diagram including steps
- Datatypes
- Hive QL
  - Create, Drop databases
  - Create table
  - Load data
  - Alter, drop table

Apache Hive

ID	Name	Salary	Designation	Dept
1201	Gopal	25000	Technical manager	TP
1202	Manisha	34000	Proofreader	PR
1203	Praveen	40000	Technical writer	TP
1204	Krian	15000	Hr Admin	HR
1205	Kranthi	30000	Op Admin	Admin

Apache Hive- View

- A view is a virtual table derived from one or more tables or other views.
- It doesn't store the data itself but represents the result of a SELECT query.
- Views are used for simplifying complex queries, providing a layer of abstraction, and controlling access to specific columns or rows of a table.
- They can be used to join tables, aggregate data, or select specific columns to present a simplified and meaningful representation of the underlying data.

View

- Generate a query to retrieve the employee details who earn a salary of more than Rs 30000. We store the result in a view named **emp_30000.**

- hive> CREATE VIEW emp_30000 AS SELECT * FROM employee WHERE salary>30000;

- hive> DROP VIEW emp_30000;

Creating an Index

- An index is a database object that improves the speed of data retrieval operations on a database table.
- It works similar to an index in a book, allowing the database engine to quickly locate and access the rows that satisfy a WHERE clause in a query.
- Indexes are created on one or more columns of a table to accelerate the search for specific data.
- While indexes improve query performance, they may have some overhead during data modification operations (INSERT, UPDATE, DELETE).

Creating an Index

- An Index is nothing but a pointer on a particular column of a table. Creating an index means creating a pointer on a particular column of a table.
- Let us take an example for index. Use the same employee table that we have used earlier with the fields Id, Name, Salary, Designation, and Dept. Create an index named index_salary on the salary column of the employee table
- hive> CREATE INDEX index_salary ON TABLE employee(salary)

  AS ‘handler’;

Creating an Index

- It is a pointer to the salary column.

- hive> DROP INDEX index_salary ON employee;

Hive QL

- **Select-Where**
  - hive> SELECT * FROM employee WHERE salary>30000;

Hive QL

- **Select-Order By**

```
+------+-------------+------------+------------------+--------+
| ID   | Name        | Salary     | Designation      | Dept   |
+------+-------------+------------+------------------+--------+
|1201  | Gopal       | 25000      | Technical manager | TP     |
|1202  | Manisha     | 34000      | Proofreader      | PR     |
|1203  | Praveen     | 40000      | Technical writer | TP     |
|1204  | Krian       | 15000      | Hr Admin         | HR     |
|1205  | Kranthi     | 30000      | Op Admin         | Admin  |
+------+-------------+------------+------------------+--------+
```

- a query to retrieve the employee details in order by clause and descending; using salary
- hive> SELECT * FROM employee ORDER BY salary DESC;

HiveQL - Select-Group By

```
+------+-------------+------------+-----------------+--------+
| ID   | Name        | Salary     | Designation     | Dept   |
+------+-------------+------------+-----------------+--------+
|1201  | Gopal       | 45000      | Technical manager | TP     |
|1202  | Manisha     | 45000      | Proofreader     | PR     |
|1203  | Masthanvali | 40000      | Technical writer | TP     |
|1204  | Krian       | 45000      | Proofreader     | PR     |
|1205  | Kranthi     | 30000      | Op Admin        | Admin  |
+------+-------------+------------+-----------------+--------+
```

- a query to retrieve the number of employees in each department.
- hive> SELECT Dept, count(*) FROM employee GROUP BY DEPT;

Select Joins

- JOIN is a clause that is used for combining specific fields from two tables by using values common to each one. It is used to combine records from two or more tables in the database.

- Types of Join
- JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

Example - Select Joins

Customer

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1  | Ramesh   | 32  | Ahmedabad | 2000.00  |
| 2  | Khilan   | 25  | Delhi     | 1500.00  |
| 3  | kaushik  | 23  | Kota      | 2000.00  |
| 4  | Chaitali | 25  | Mumbai    | 6500.00  |
| 5  | Hardik   | 27  | Bhopal    | 8500.00  |
| 6  | Komal    | 22  | MP        | 4500.00  |
| 7  | Muffy    | 24  | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Order

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 | 3000   |
| 100 | 2009-10-08 00:00:00 |           3 | 1500   |
| 101 | 2009-11-20 00:00:00 |           2 | 1560   |
| 103 | 2008-05-20 00:00:00 |           4 | 2060   |
+-----+---------------------+-------------+--------+
```

Join

- JOIN clause is used to combine and retrieve the records from multiple tables. JOIN is same as OUTER JOIN in SQL.
- hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT FROM CUSTOMERS c JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
| 3  | kaushik  | 23  | 3000   |
| 3  | kaushik  | 23  | 1500   |
| 2  | Khilan   | 25  | 1560   |
| 4  | Chaitali | 25  | 2060   |
+----+----------+-----+--------+
```

LEFT OUTER JOIN

- The HiveQL LEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table.
- This means, if the ON clause matches 0 (zero) records in the right table, the JOIN still returns a row in the result, but with NULL in each column from the right table.
- A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.
- hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c LEFT OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);

LEFT OUTER JOIN

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

RIGHT OUTER JOIN

- A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate.
- hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c RIGHT OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

FULL OUTER JOIN

- The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition

- hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c
  FULL OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);

FULL OUTER JOIN

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

Hive

- Partitioning
- Working of partitioning
- Static and dynamic partition
- Create partition, insert data, and display data queries

- Bucketing
- Need of buckets
- working process
- Create, load, insert, display data queries

Apache Hive Limitations

- Subqueries are not supported.
- The latency in the apache hive query is very high.
- Hive is not used for real-time data querying since it takes a while to produce a result.

Apache Pig

- Apache Pig is an abstraction over MapReduce.
- It is a tool/platform which is used to analyze larger sets of data representing them as data flows.
- To write data analysis programs, Pig provides a high-level language known as Pig Latin.
- This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

Apache Pig

- To analyze data using Apache Pig, programmers need to write scripts using Pig Latin language.
- All these scripts are internally converted to Map and Reduce tasks.
- Apache Pig has a component known as Pig Engine that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.
- Pig Latin is SQL-like language and it is easy to learn Apache Pig when you are familiar with SQL.

Features of Pig

- **Rich set of operators** − It provides many operators to perform operations like join, sort, filter, etc.
- **Ease of programming** − Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
- **Extensibility** − Using the existing operators, users can develop their own functions to read, process, and write data.
- **UDF's** − Pig provides the facility to create User-defined Functions in other programming languages such as Java and invoke or embed them in Pig Scripts.
- **Handles all kinds of data** − Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

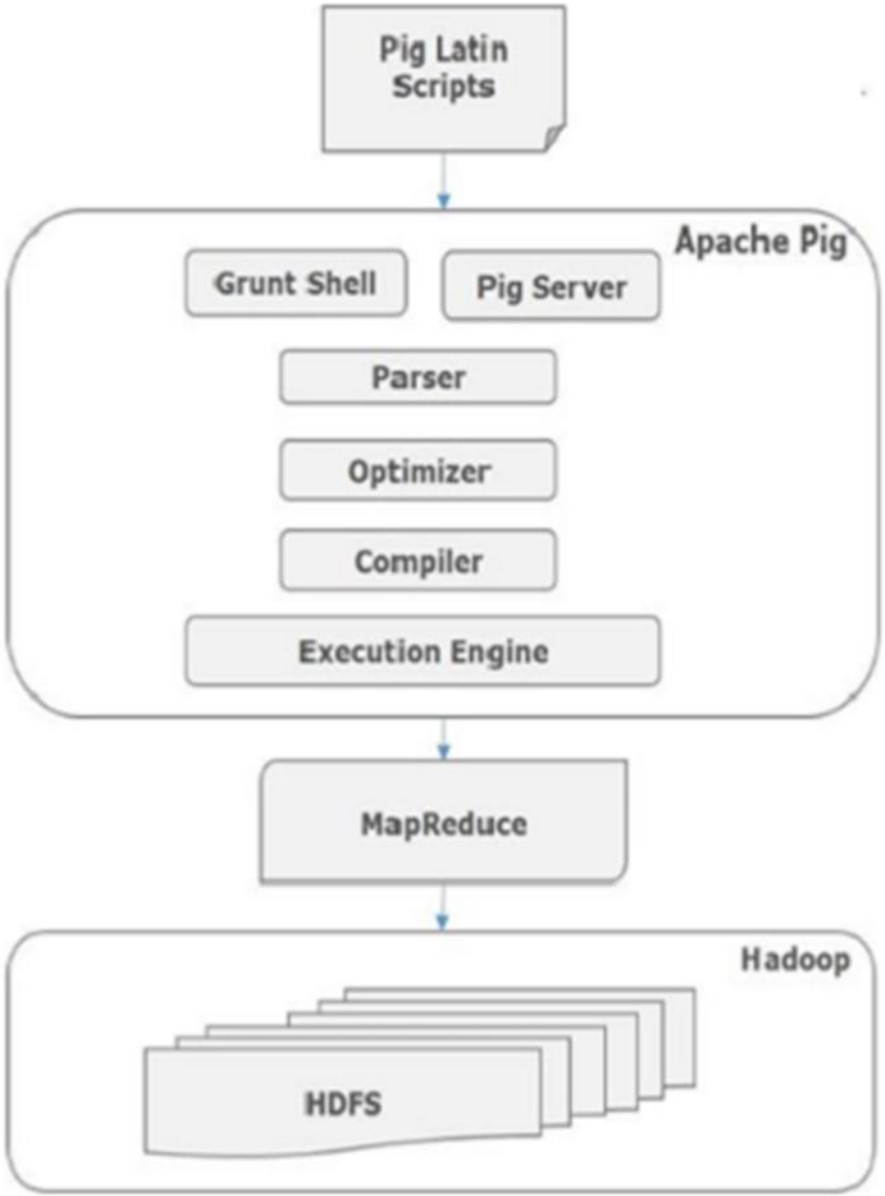
Apache Pig

- To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded).
- After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

- Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer’s job easy.

Apache Pig - Architecture

- The parser is a crucial component in Apache Pig's architecture as it acts as the gateway for processing Pig Latin scripts.
- It ensures syntactic correctness, tokenizes the script, contributes to the generation of the logical plan. This early stage of processing is essential for subsequent phases like optimization and execution on a Hadoop cluster.

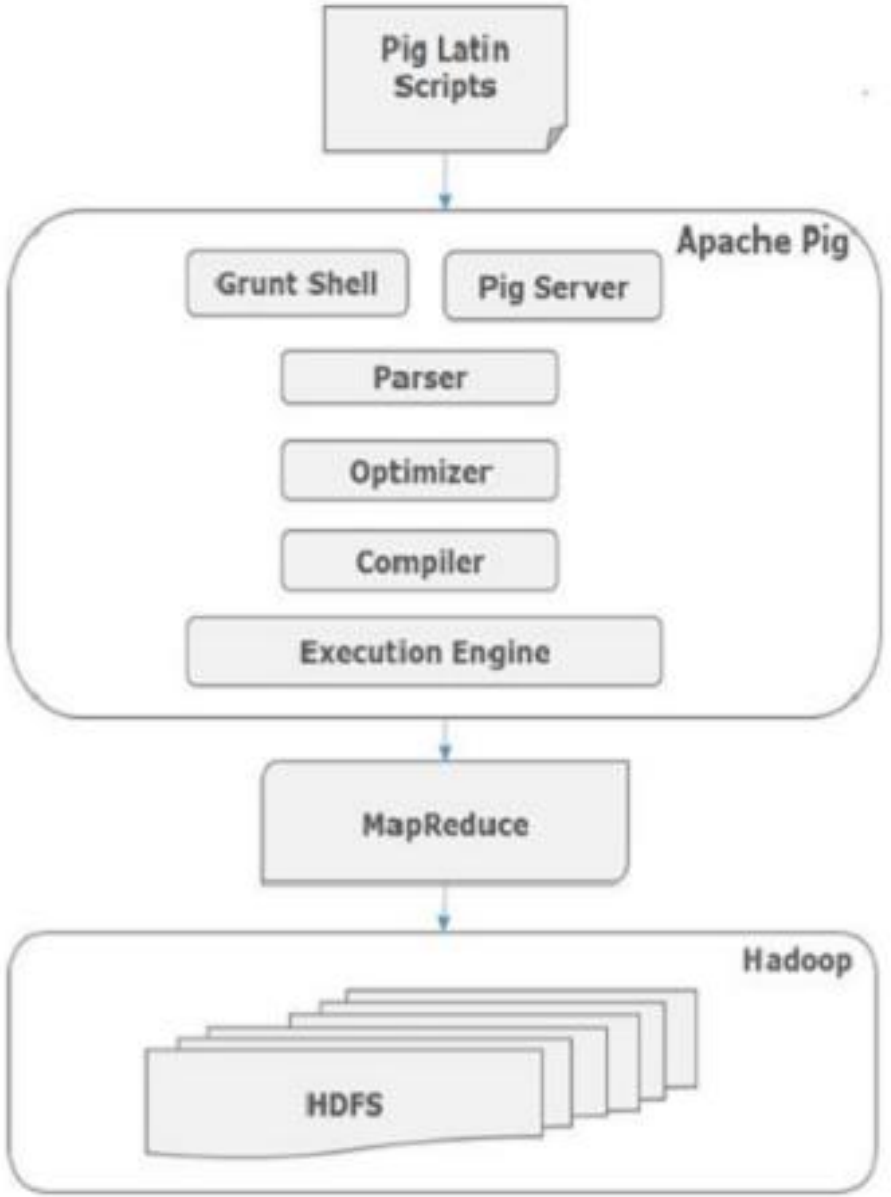


Apache Pig - Architecture

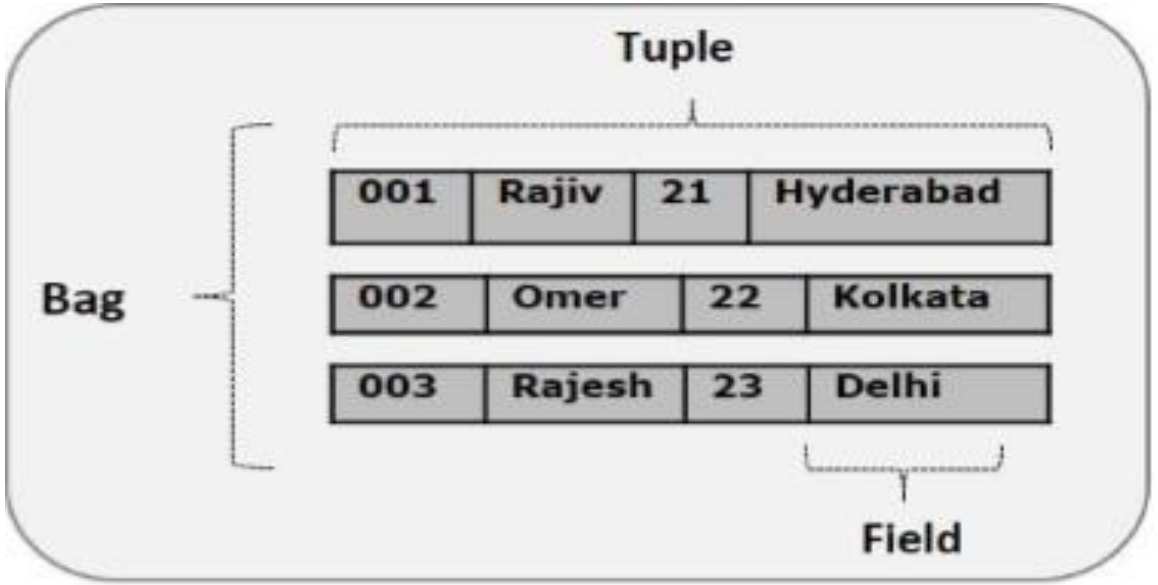
- The logical plan is passed

to the logical optimizer, which carries out the logical optimizations.

- The compiler compiles the optimized logical plan into a series of MapReduce jobs.



Apache Pig



- Relation

  A relation is a bag of tuples.

1. Relations:

Data is represented as relations (tables) in Pig.

Relations consist of tuples, which are ordered sets of fields.

2. Data Types:

Pig supports various data types, including simple types (int, long, float, chararray) and complex types (tuple, bag, map).

Apache Pig

- **Transformations and Operations**:
- Pig provides a set of built-in operators for performing transformations on data, such as LOAD, FILTER, GROUP, JOIN, FOREACH, ORDER BY, and STORE.
- Users can also define their custom functions in Pig Latin
- **Grunt Shell:**
- Pig provides an interactive shell called Grunt, which allows developers to test and run Pig Latin commands interactively.
- Apache Pig is suitable for a wide range of data processing tasks, including ETL (Extract, Transform, Load), data cleaning, and data analysis

Apache Pig

- **User Defined Functions (UDFs):**
- Developers can write custom functions in Java, known as User Defined Functions (UDFs).
- UDFs can be integrated into Pig scripts for custom processing tasks.

Pig scripts

- Load data from a CSV file into a Pig relation:
  - grunt> Student_data = LOAD 'student_data.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);
- Filter data to include only records where age is greater than 25:
  - grunt> filtered_data = FILTER input_data BY age > 25;
- Group data by department and calculate the average salary:
  - grouped_data = GROUP input_data BY department;
  - average_salary = FOREACH grouped_data GENERATE group AS department, AVG(input_data.salary) AS avg_salary;

Pig scripts

- -- Join two datasets
- employee_data = LOAD 'employee_data.csv' USING PigStorage(',') AS (id: int, name: chararray, department: chararray);
- department_data = LOAD 'department_data.csv' USING PigStorage(',') AS (dept_id: int, dept_name: chararray);
- joined_data = JOIN employee_data BY department, department_data BY dept_id;

Pig scripts

- Sort data
  - sorted_data = ORDER input_data BY salary DESC;
- Extract specific fields
  - extracted_data = FOREACH input_data GENERATE id, name;
- Store result in a CSV file
  - STORE result_data INTO 'output.csv' USING PigStorage(',');

Limitations of Apache Pig

- The choice of data processing tools often depends on the specific requirements and characteristics of the data and the processing tasks at hand.

- **Performance Overhead:**
- Pig scripts are translated into a series of MapReduce jobs, which can introduce performance overhead due to the multiple stages of processing. For certain operations, writing custom MapReduce code might be more efficient.

Limitations of Apache Pig

- **Not Suitable for Real-Time Processing:**
- Pig is designed for batch processing and is not well-suited for real-time data processing. If low-latency processing is a requirement, other tools like Apache Storm or Apache Flink might be more appropriate.
- **Limited Support for Complex Data Structures:**
- While Pig supports complex data types like tuples, bags, and maps, it may have limitations when dealing with highly nested or complex data structures compared to other data processing tools.

Distributed Databases

- Motivation for distributed databases
- Advantages
- Working

Distributed hash tables

- Use of DHT and its applications
- Working of DHT
- Assign key value pairs
- Resolve Querying from one peer to retrieve the value from another peer
- Peer churn
- Advantages