

UNIT-3

Introduction to MAPREDUCE Programming

Introduction , Mapper, Reducer, Combiner, Partitioner , Searching, Sorting , Compression, Real time applications using MapReduce, Data serialization and Working with common serialization formats, Big data serialization formats.

Q) What is MapReduce. Explain in detail different phases in MapReduce. (or) Explain MapReduce anatomy.

MapReduce is a programming model for data processing. Hadoop can run MapReduce programs written in Java, Ruby and Python.

MapReduce programs are inherently parallel, thus very large scale data analysis can be done fastly.

In MapReduce programming, Jobs(applications) are split into a set of **map tasks and reduce tasks**.

Map task takes care of **loading, parsing, transforming and filtering**.

The responsibility of reduce task is **grouping and aggregating** data that is produced by map tasks to generate final output.

Each **map** task is broken down into the following phases:

1. Record Reader
2. Mapper
3. Combiner
4. Partitioner.

The output produced by the map task is known as intermediate <keys, value> pairs. These intermediate <keys, value> pairs are sent to reducer.

The **reduce** tasks are broken down into the following phases:

1. Shuffle
2. Sort
3. Reducer
4. Output format.

Hadoop assigns map tasks to the DataNode where the actual data to be processed resides. This way, Hadoop ensures data locality. **Data locality** means that data is not moved over network; only computational code moved to process data which saves network bandwidth.

Mapper Phases:

Mapper maps the input <keys, value> pairs into a set of intermediate <keys, value> pairs.

Each **map** task is broken into following phases:

1. **RecordReader**: converts byte oriented view of input in to Record oriented view and presents it to the Mapper tasks. It presents the tasks with keys and values.
 - i) InputFormat: It reads the given input file and splits using the method **getsplits()**.
 - ii) Then it defines RecordReader using **createRecordReader()** which is responsible for generating <keys, value> pairs.
2. **Mapper**: Map function works on the <keys, value> pairs produced by RecordReader and generates intermediate (key, value) pairs.

Methods:

- protected void cleanup(Context context): called once at tend of task.
- **protected void map(KEYIN key, VALUEIN value, Context context): called once for each key-value pair in input split.**
- void run(Context context): user can override this method for complete control over execution of Mapper.
- protected void setup(Context context): called once at beginning of task to perform required activities to initiate map() method.

3. **Combiner**: It takes intermediate <keys, value> pairs provided by mapper and applies user specific aggregate function to only one mapper. It is also known as local Reducer.

We can optionally specify a combiner using **Job.setCombinerClass(ReducerClass)** to perform local aggregation on intermediate outputs.

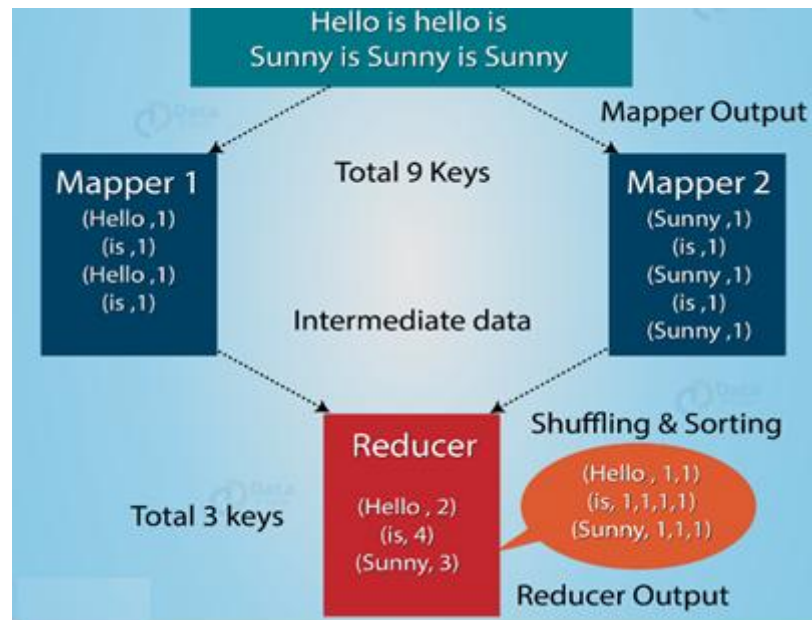


Fig. MapReduce without Combiner class

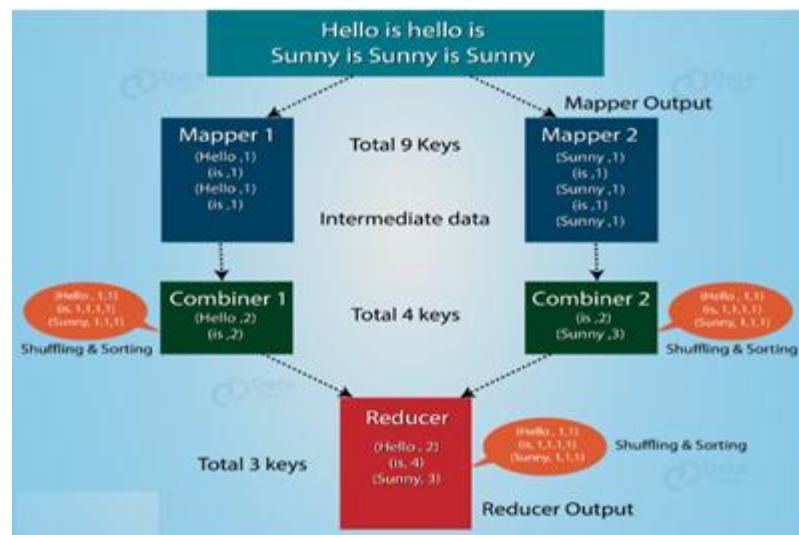


Fig. MapReduce with Combiner class

4. **Partitioner:** Take intermediate <keys, value> pairs produced by the mapper, splits them into partitions the data using a user-defined condition.

The default behavior is to hash the key to determine the reducer. User can control by using the method:

int getPartition(KEY key, VALUE value, int numPartitions)

Reducer Phases:

1. **Shuffle & Sort:**

- Downloads the grouped key-value pairs onto the local machine, where the Reducer is running.
- The individual <keys, 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.

2. **Reducer:**

- The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them.
- Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing.
- Once the execution is over, it gives zero or more key-value pairs to the final step.

Methods:

- protected void cleanup(Context context): called once at the end of task.

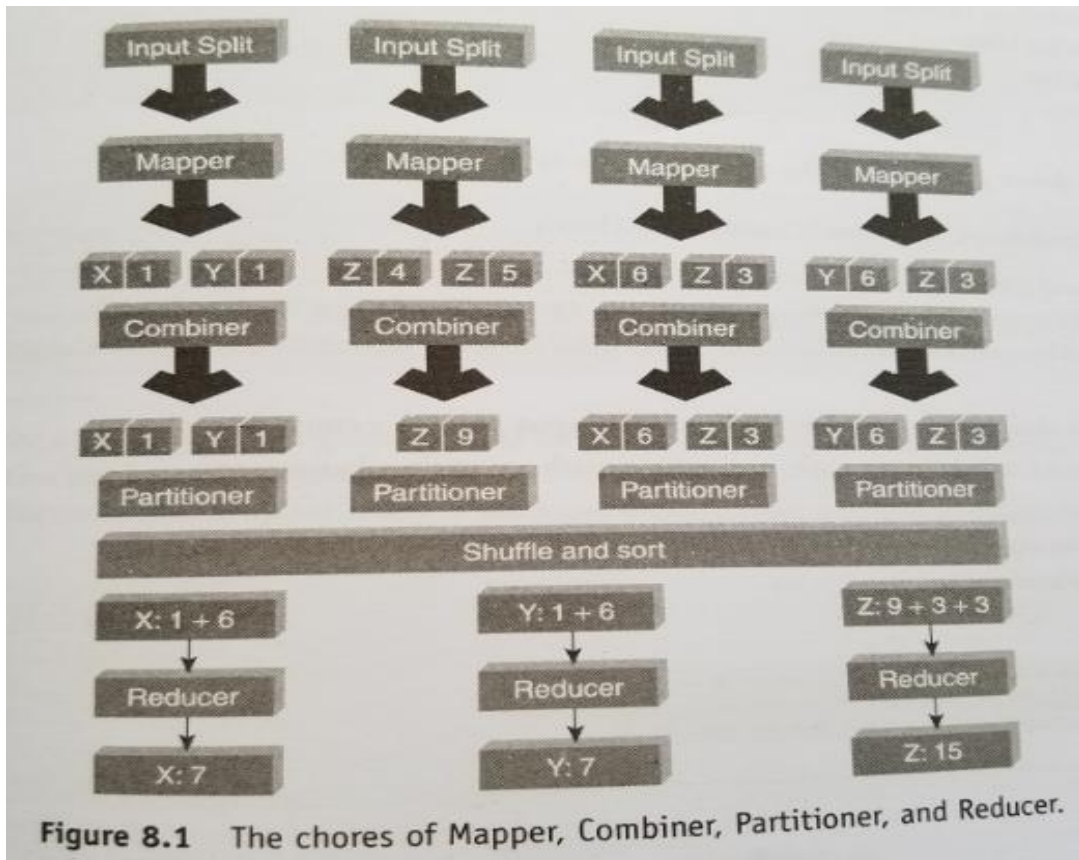
- **protected void reduce(KEYIN key, VALUEIN value, Context context):**
called once for each key-value pair.

- void run(Context context): user can override this method for complete control over execution of Reducer.

- protected void setup(Context context): called once at the beginning of task to perform required activities to initiate reduce() method.

3. **Output format:**

- In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.



Compression: In MapReduce programming we can compress the output file. Compression provides two benefits as follows:

- Reduces the space to store files.
- Speeds up data transfer across the network.

We can specify compression format in the Driver program as below:

```
conf.setBoolean("mapred.output.compress",true);
```

```
conf.setClass("mapred.output.compression.codec",GzipCodec.class,CompressionCodec.class);
```

Here, codec is the implementation of a compression and decompression algorithm, GzipCodec is the compression algorithm for gzip.

Q) Write a MapReduce program for WordCount problem.

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount
{
    public static class WCMapper extends Mapper <Object, Text, Text, IntWritable>

    {
        final static IntWritable one = new IntWritable(1);
        Text word = new Text();
        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {

            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
}
```

```
}
```

```
    public static class WCReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context )
throws IOException, InterruptedException {

            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }
}
```

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(WCMapper.class);
    job.setReducerClass(WCReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

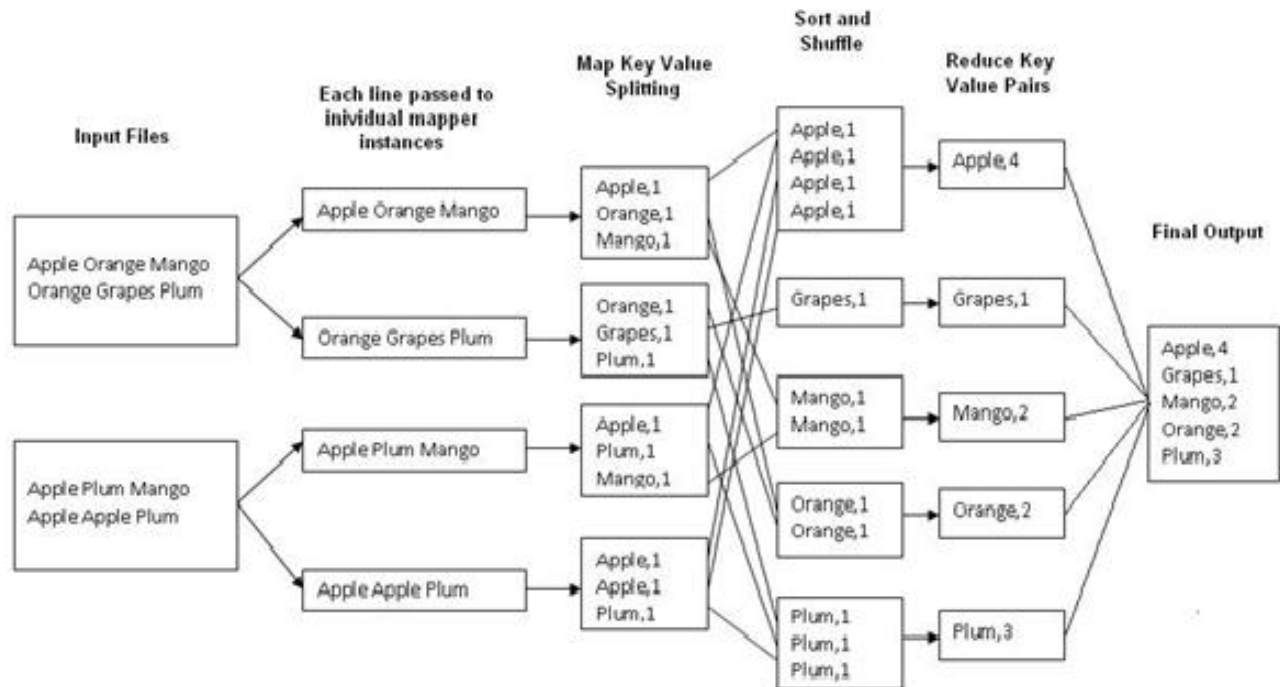


Fig. MapReduce paradigm for WordCount

Q) Write a MapReduce program to calculate employee salary of each department in the university.

I/P:

001,it,10000

002,cse,20000

003,it,30000

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
```



```

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Salary
{

    public static class SalaryMapper extends Mapper<LongWritable, Text, Text,
IntWritable>
    {

        public void map(LongWritable key, Text value, Context context) throws IOException,
            InterruptedException
        {

            String[] token = value.toString().split(",");
            int s = Integer.parseInt(token[2]);
            IntWritable sal = new IntWritable();
            sal.set(s);
            context.write(new Text(token[1]),sal);

        }
    }

    public static class SalaryReducer extends Reducer<Text, IntWritable, Text, IntWritable>
    {

        private IntWritable result = new IntWritable();

```

```

        public void reduce(Text key, Iterable<IntWritable> values, Context context ) throws
IOException, InterruptedException
        {

            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key,result);
        }
    }

```

```

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Salary");

    job.setJarByClass(Salary.class);
    job.setMapperClass(SalaryMapper.class);
    job.setReducerClass(SalaryReducer.class);

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

```

Q) Write a user define partitioner class for WordCount problem.

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;

public class WordCountPartitioner extends Partitioner<Text,IntWritable>{

    public int getPartition(Text key, IntWritable value, int numPartitions){
        String word = key.toString();
        char alphabet = word.toUpperCase().charAt(0);
        int partitionNumber = 0;
        switch(alphabet){
            case 'A': partitionNumber = 1;break;
            case 'B': partitionNumber = 1;break;
            case 'C': partitionNumber = 1;break;
            case 'D': partitionNumber = 1;break;
            case 'E': partitionNumber = 1;break;
            case 'F': partitionNumber = 1;break;
            case 'G': partitionNumber = 1;break;
            case 'H': partitionNumber = 1;break;
            case 'I': partitionNumber = 1;break;
            case 'J': partitionNumber = 1;break;
            case 'K': partitionNumber = 1;break;
            case 'L': partitionNumber = 1;break;
            case 'M': partitionNumber = 1;break;
            case 'N': partitionNumber = 1;break;
            case 'O': partitionNumber = 1;break;
            case 'P': partitionNumber = 1;break;
            case 'Q': partitionNumber = 1;break;
            case 'R': partitionNumber = 1;break;
```

```

        case 'S': partitionNumber = 1;break;
        case 'T': partitionNumber = 1;break;
        case 'U': partitionNumber = 1;break;
        case 'V': partitionNumber = 1;break;
        case 'W': partitionNumber = 1;break;
        case 'X': partitionNumber = 1;break;
        case 'Y': partitionNumber = 1;break;
        case 'Z': partitionNumber = 1;break;
        default: partitionNumber = 0;break;
    }
    return partitionNumber;
}
}

```

In the driver program set the partitioner class as shown below:

```

job.setNumReduceTasks(27);
job.setPartitionerClass(WordCountPartitioner.class);

```

Q) Write a MapReduce program for sorting following data according to name.

Input:

001,chn

002,yg

003,dmr

004,bns

```

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;

```

```

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class Sort{
    public static class SortMapper extends Mapper<LongWritable,Text,Text,Text>{
        protected void map(LongWritable key, Text value, Context context) throws
IOException,InterruptedException{

            String[] token = value.toString().split(",");
context.write(new Text(token[1]),new Text(token[0]+"-"+token[1]));
        }
    }

    public static class SortReducer extends Reducer<Text,Text,NullWritable,Text>{
        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException,InterruptedException{

            for(Text details:values){
                context.write(NullWritable.get(),details);
            }
        }
    }
}

```

```

public          static          void          main(String          args[])          throws
IOException,InterruptedException,ClassNotFoundException{

    Configuration conf = new Configuration();
    Job job = new Job(conf);
    job.setJarByClass(Sort.class);
    job.setMapperClass(SortMapper.class);
    job.setReducerClass(SortReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true)?0:1);
}
}

```

Q) Write a MapReduce program to arrange the data on user-id, then within the user id sort them in increasing order of the page count.

Input:

```

001,3,www.tutorialspoint.com
001,4,www.javapoint.com
002,5,www.javapoint.com
003,2,www.analyticsvidhya.com

```

```

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;

```

```

import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class Sort{
    public static class SortMapper extends Mapper<LongWritable,Text,Text,Text>{
        Text comp_key = new Text();
        protected void map(LongWritable key, Text value, Context context) throws
IOException,InterruptedException{

            String[] token = value.toString().split(",");
            comp_key.set(token[0]);
            comp_key.set(token[1]);
            context.write(comp_key,new Text(token[0]+"-"+token[1]+"-"+token[2]));
        }
    }

    public static class SortReducer extends Reducer<Text,Text,NullWritable,Text>{
        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException,InterruptedException{

            for(Text details:values){
                context.write(NullWritable.get(),details);
            }
        }
    }

    public static void main(String[] args) throws
IOException,InterruptedException,ClassNotFoundException{

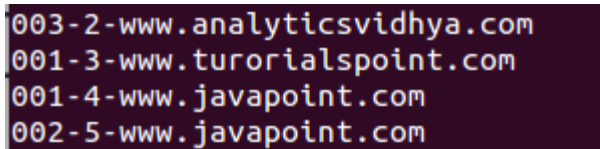
```

```

    Configuration conf = new Configuration();
    Job job = new Job(conf);
    job.setJarByClass(Sort.class);
    job.setMapperClass(SortMapper.class);
    job.setReducerClass(SortReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true)?0:1);
}
}

```

o/p:



```

003-2-www.analyticsvidhya.com
001-3-www.tutorialspoint.com
001-4-www.javapoint.com
002-5-www.javapoint.com

```

Q) Write a MapReduce program to search an employee name in the following data:

Input:

001,chn

002,yg

003,dmr

004,bns

```

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;

```



```

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.InputSplit;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

public class Search{
    public class SearchMapper extends Mapper<LongWritable, Text, Text, Text>{

        static String key;
        static int pos=0;
        protected void setup(Context context) throws
IOException,InterruptedException{
            Configuration config = context.getConfiguration();
            keyword = config.get("keyword");
        }

        protected void map(LongWritable key, Text value, Context
context)IOException,InterruptedException{
            InputSplit in = context.getInputSplit();
            FileSplit f = f.getPath.getName();
            Integer wordPos;
            pos++;
            if(value.toString().contains(keyword){
                wordPos = value.find(keyword);
                context.write(value,      new      Text(fileName      +      ","+new
IntWritable(pos),toString()+"," +wordPos.toString()));

```

```

    }
}

public static class SearchReducer extends Reducer<Text,Text,Text,Text>{
    public void reduce(Text key, Text value, Context context) throws
IOException,InterruptedException{
        context.write(key,value);
    }
}

public static void main(String args[]) throws
IOException,InterruptedException,ClassNotFoundException{
    Configuration conf = new Configuration();
    Job job = new Job(conf);
    job.setJarByClass(Search.class);
    job.setMapperClass(SearchMapper.class);
    job.setReducerClass(SearchReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    job.setNumReduceTasks(1);
    job.getConfiguration().set("keyword","chp");
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true)?0:1);
}
}

```

Q) What are the Real time applications using MapReduce Programming?

- Social networks
- Media and Entertainment
- Health Care
- Business
- Banking
- Stock Market
- Weather Forecasting

Q) Define Data Serialization. Explain BigData serialization formats.

Data Serialization is the process of converting object data into byte stream data for transmission over a network across different nodes in a cluster or for persistent data storage.

MapReduce offers straightforward, well-documented support for working with simple data formats such as log files.

But the use of MapReduce has evolved beyond log files to more sophisticated data serialization formats—such as text, XML, and JSON—to the point that its documentation and built-in support runs dry.

characteristics for big data serialization:

- splittability - easier to achieve splits on byte streams rather than JSON or XML files
- portability - schema can be consumed by different languages
- versioning - flexibility to define fields with default values or continue to use old schema version
- data integrity - serialization schemas enforces data correctness. Thanks to them, errors can be detected earlier, when data is written.
- Native support in MapReduce—The input/output formats that support reading and writing files in their native format

- Code generation—The ability to generate Java classes and utilities that can be used for serialization and deserialization.

Most common BigData serialization formats are XML and JSON.

Working with XML and JSON in MapReduce, however, poses two equally important challenges:

1. MapReduce requires classes that can support reading and writing a particular data serialization format.
2. MapReduce's power lies in its ability to parallelize reading your input data. If the **input files are large** it's crucial that the classes reading serialization format be able to split large files, so that multiple map tasks can read them in parallel.

MapReduce and XML

- XML has existed since 1998 as a mechanism to represent data that's readable by machine and human alike.
- While MapReduce comes bundled with an InputFormat that works with text, it doesn't come with one that supports XML.
- Problem to work with large XML files in MapReduce and be able to split and process them in parallel.
- Solution Mahout's XMLInputFormat can be used to work with XML files in HDFS with MapReduce. It reads records that are delimited by a specific XML begin and end tag. This technique also covers how XML can be emitted as output in MapReduce output.

```
Eg. job.setInputFormatClass(XmlInputFormat.class);
```

MapReduce and JSON

A JSON object contains data in the form of key/value pair. The keys are strings and the values are the JSON types. Keys and values are separated by colon. Each entry (key/value pair) is separated by comma.

Eg.

```
{  
  "employee": {  
    "name":    "sonoo",  
    "salary": 56000,  
    "married": true  
  }  
}
```

To write a MapReduce job that can take as input these large JSON files, there are 2 problems:

1. MapReduce doesn't come with an InputFormat that works with JSON
2. how does one even go about splitting JSON?

Solution: Elephant Bird, an open source project that contains some useful utilities for working with LZOP compression, has an `LzoJsonInputFormat` that can read JSON, though it requires that the input file be LZOP-compressed.

UNIT-4(upto 2nd Mid)

HIVE

Q) What is Hive? Explain features of Hive.

Hive is data warehousing tool and is used to query structured data that was built on top of Hadoop for providing data summarization, query, and analysis.

Hive Provides HQL (Hive Query Language) which is similar to SQL. Hive compiles SQL queries into MapReduce jobs and then runs the job in the Hadoop cluster.

Features of Hive:

- It is similar to SQL
- HQL is easy to code
- Hive supports rich datatypes such as structs, lists and maps
- Hive supports SQL filters, group-by and order-by clauses
- Custom types and custom functions can be defined.

Q) Explain various Hive Data Units

Databases: The name space for tables

Tables: set of records that have similar schema

Partitions: Logical separations of data based on classification of given information as per specific attributes.

Buckets or clusters: Similar to partitions but uses hash function to segregate data and determines the cluster or bucket into which the record should be placed.

Q) Explain Hive Architecture with a neat diagram.

External Interfaces- CLI, WebUI, JDBC, ODBC programming interfaces

Hive CLI: The most commonly used interface to interact with Hadoop.

Hive Web Interface: It is simple graphic interface to interact with Hive and to execute query.

Thrift Server – Cross Language service framework . This is an optional Sever. This can be used to submit Hive jobs from a remote client.

JDBC/ODBC: Jobs can be submitted from a JDBC client. One can write java code to connect to Hive and submit jobs on it.

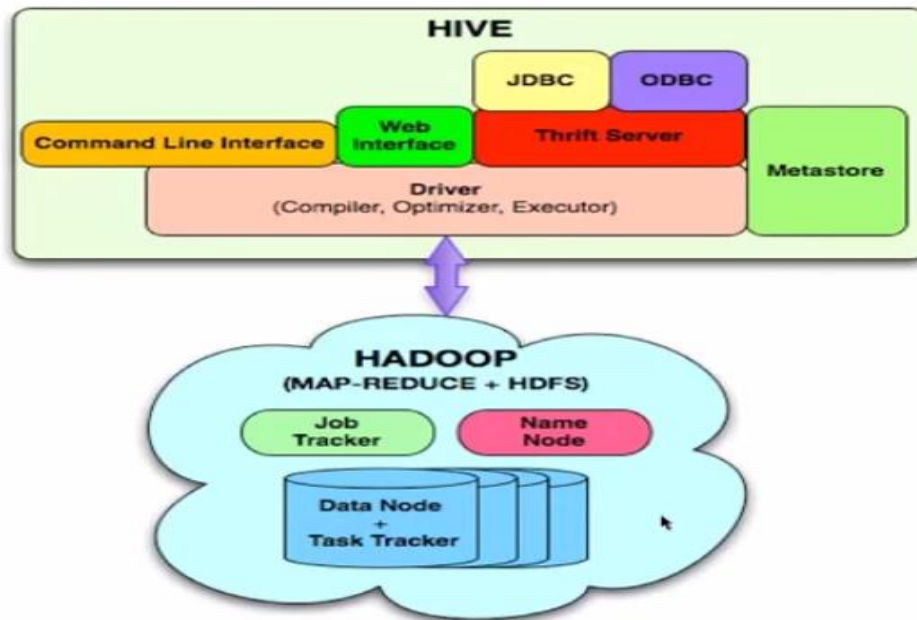


Fig. Hive Architecture

Metastore- Meta data about the Hive tables, partitions. A metastore consists of Meta store service and Database.

There are three kinds of Metastore:

1. Embedded Meta store
2. Local Metastore
3. Remote Metastore

Driver- Brain of Hive! Hive queries are sent to the driver for Compilation, Optimization and Execution.

Q) Explain different data types in Hive.

Hive Data types are used for specifying the column/field type in Hive tables.

Mainly Hive Data Types are classified into 5 major categories, let's discuss them one by one:

a. Primitive Data Types in Hive

Primitive Data Types also divide into 3 types which are as follows:

- Numeric Data Type
- Date/Time Data Type
- String Data Type

TINYINT	'1 byte signed integer',	-128 to 127
SMALLINT	'2 byte signed integer',	-32, 768 to 32, 767
INT	'4 byte signed integer',	-2,147,483,648 to 2,147,483,647
BIGINT	'8 byte signed integer',	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
FLOAT	'Single precision floating point',	
DOUBLE	'Double precision floating point',	
DECIMAL	'Precise decimal type based on Java BigDecimal Object',	
TIMESTAMP	'YYYY-MM-DD HH:MM:SS.ffffffff" (9 decimal place precision)',	
BOOLEAN	'TRUE or FALSE boolean data type',	
STRING	'Character String data type',	
BINARY	'Data Type for Storing arbitrary	

b. Complex Data Types in Hive

In this category of Hive data types following data types are come-

- Array
- MAP
- STRUCT
- UNION

ARRAY<TINYINT> 'A collection of fields all of the same data type indexed BY an integer',

MAP<STRING,INT> 'A Collection of Key,Value Pairs where the Key is a Primitive Type and the Value can be anything. The chosen data types for the keys and values must remain the same per map',

STRUCT<first : SMALLINT, second : FLOAT, third : STRING>

'A nested complex data structure',

UNIONTYPE<INT,FLOAT,STRING>

'A Complex Data Type that can hold One of its Possible Data Types at Once'

eg.

t1.txt

1001,cse^1,1|2|3,A|50000,3,true

1002,cse^2,1|2,B|40000,good,true

Creating a table t1:

create table t1(id int,class map<string,int>,sections array<int>,hostel

struct<grade:string,fee:double>,rating uniontype<int,string>,exist boolean)

row format delimited

fields terminated by ','

collection items terminated by '|'

map keys terminated by '^'

lines terminated by '\n'

stored as textfile;

Q) Explain different HIVE file formats

The file formats in Hive specify how records are encoded in a file.

The file formats are :

1. **Text File:** The default file format is text file. In this format, each record is a line in the file.
2. **Sequential file:** Sequential files are flat files that store binary key-value pairs. It includes compression support which reduces the CPU, I/O requirement.
3. **RC File (Record Columnar File):** RCFile stores the data in column oriented manner which ensures that Aggregation operation is not an expensive operation.

RC File Instead of only partitioning the table horizontally like the row oriented DBMS, RCFile partitions this table first horizontally and then vertically to serialize the data.

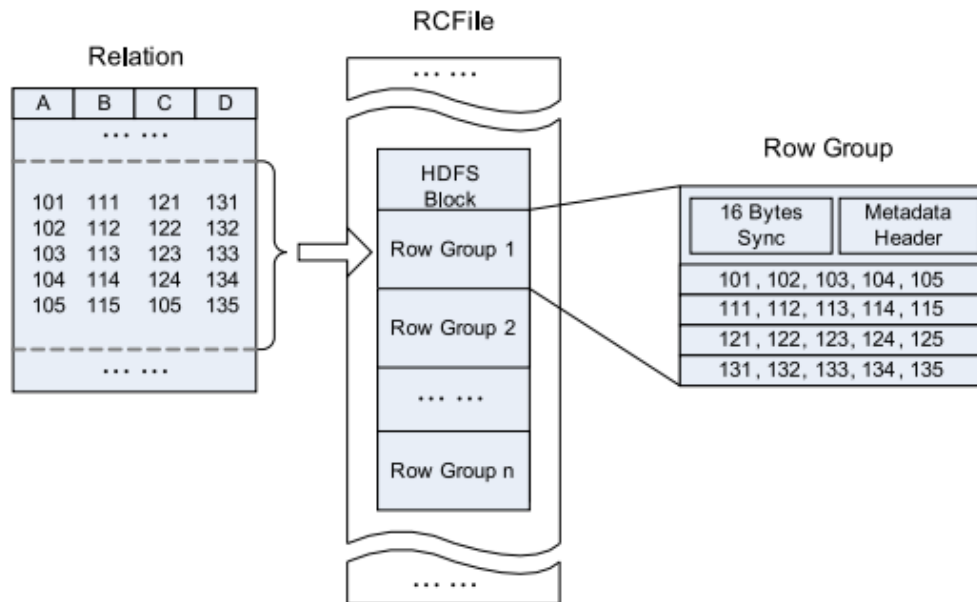


Fig. Hive File Formats