Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

असतो मा सद्गमय

UNIT- II (Writing MAP-REDUCE Programs)

Helloworld Job:-

step 1: creating a file

$ cat > file.txt

file.txt

hi, How r u?

ctrl+d

step-2: Loading file.txt from local file system to HDFS

$ hadoop fs -put file.txt file

step-3: writing programs → ReducerCode.java.

↙ DriverCode.java ↘ MapperCode.java

step 4: Compiling all above .java files.

$ javac -classpath $HADOOP-HOME/hadoop-core.jar *.java

step 5: creating jar file

$ jar cvf test.jar *.class

step 6: Running test.JAR on file (which is there in your HDFS).

↑ can be file or folder

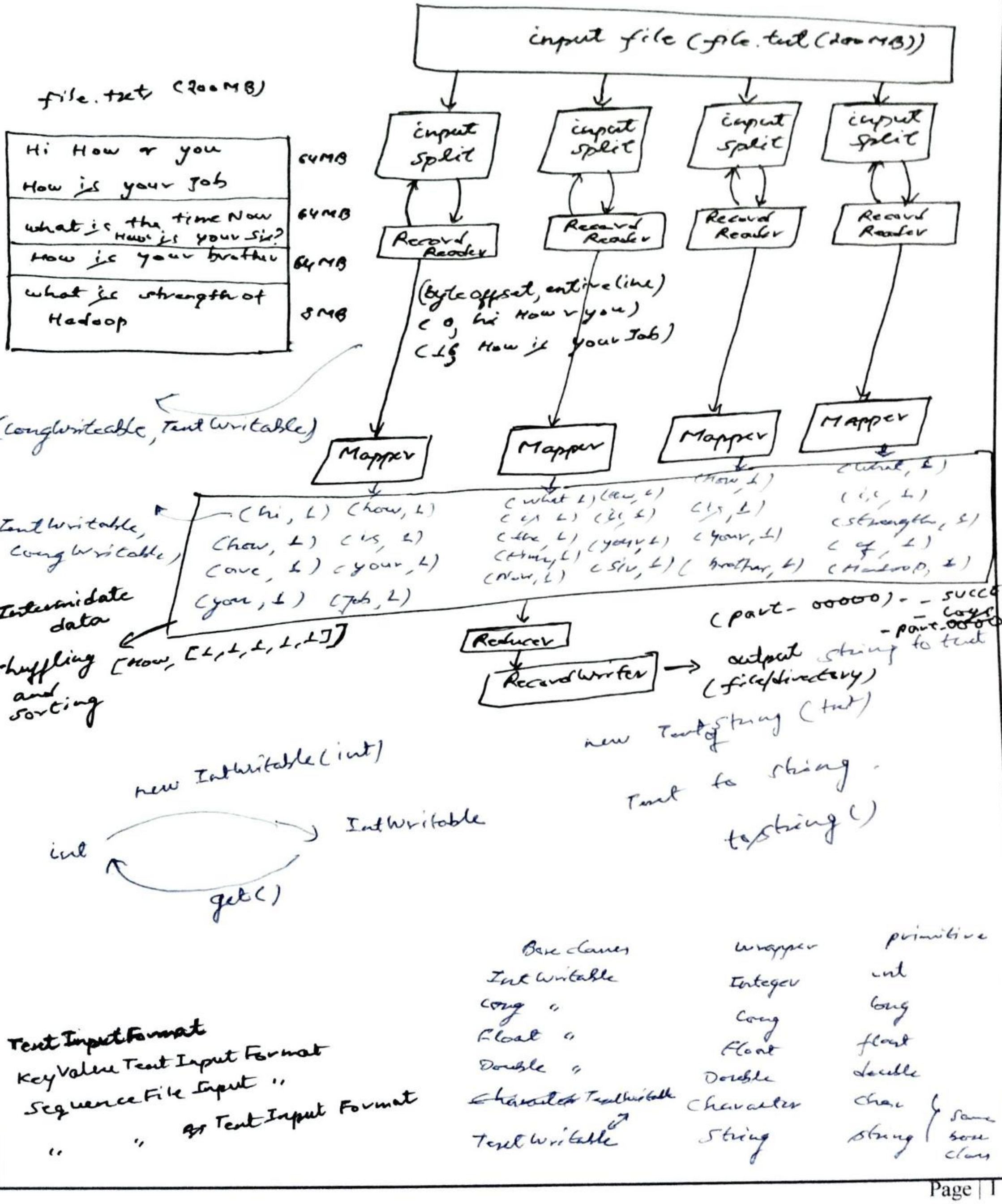
$ hadoop JAR test.JAR DriverCode file TestOutput

↓ (containing Main Method) ↳ only dir

Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

MAPReduce Flow chart:

hadoop jar test.jar DriverCode file.txt
Testoutputfile



Text InputFormat
KeyValue Text Input Format
SequenceFile Input "
" " or Text Input Format

Base classes	wrapper	primitive	
IntWritable	Integer	int	
Long "	Long	long	
Float "	Float	float	
Double "	Double	double	
~~Character~~ TextWritable	Character	char	Same base class
TextWritable	String	String	

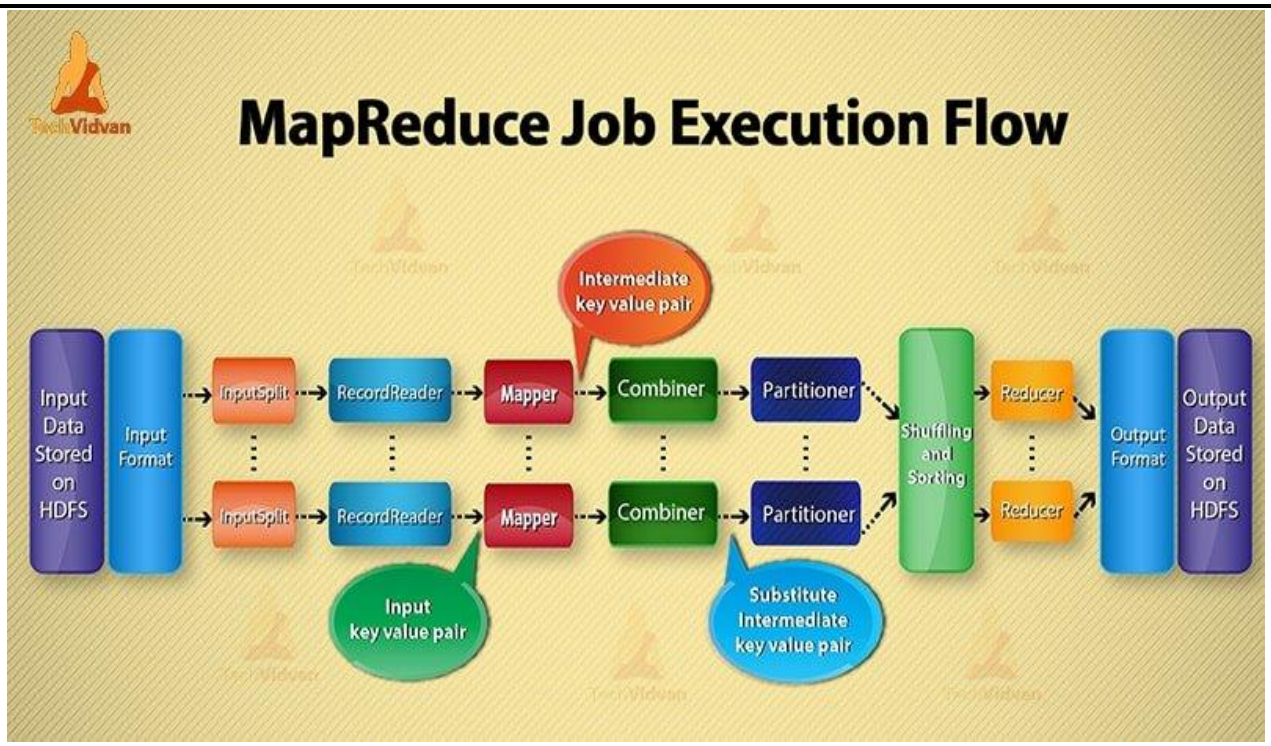
Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Hadoop Map Reduce Job Execution Flow Chart:

Hadoop Map Reduce is the data processing layer. It processes the huge amount of structured and unstructured data stored in HDFS. Map Reduce processes data in parallel by dividing the job into the set of independent tasks. So, parallel processing improves speed and reliability.

Hadoop Map Reduce data processing takes place in 2 phases- Map and Reduce phase.

- Map phase- It is the first phase of data processing. In this phase, we specify all the complex logic/business rules/costly code.
- Reduce phase- It is the second phase of processing. In this phase, we specify light-weight processing like aggregation/summation.



Steps of MAP Reduce Flow Chart: These are the following steps that are performed in the MAP Reduce.

1. Input Files

In input files data for Map Reduce job is stored. In HDFS, input files reside. Input files format is arbitrary. Line-based log files and binary format can also be used.

2. InputFormat:

After that InputFormat defines how to split and read these input files. It selects the files or other objects for input. InputFormat creates InputSplit.

3. InputSplits

It represents the data which will be processed by an individual Mapper. For each split, one map task is created. Thus the number of map tasks is equal to the number of InputSplits. Framework divide split into records, which mapper process.

4. RecordReader

--It communicates with the inputSplit. And then converts the data into key-value pairs suitable for reading by the Mapper.

--RecordReader by default uses TextInputFormat to convert data into a key-value pair.

-- It assigns byte offset to each line present in the file. Then, these key-value pairs are further sent to the mapper for further processing.

5. Mapper

--It processes input record produced by the RecordReader and generates intermediate key-value pairs.

-- The output of the mapper is the full collection of key-value pairs.

6. Combiner

Combiner is Mini-reducer which performs local aggregation on the mapper's output. It minimizes the data transfer between mapper and reducer.

7. Partitioner:

Partitioner comes into the existence if we are working with more than one reducer. It takes the output of the combiner and performs partitioning.

8. Shuffling and Sorting: In this phase all common key-value pair are shuffled on the basis of key and their common key and values are merged.

Then all the mappers finish and shuffle the output on the reducer nodes. Then framework merges this intermediate output and sort.

9. Reducer: Reducer then takes set of intermediate key-value pairs produced by the mappers as the input. After that runs a reducer function on each of them to generate the output.

The output of the reducer is the final output. Then framework stores the output on Output Directory over to HDFS.

10. Record-Writer: It writes these output key-value pair from the Reducer phase to the output files.

11. OutputFormat:

--OutputFormat defines the way how RecordReader writes these output key-value pairs in output files.

--Thus OutputFormat instances write the final output of reducer on HDFS.

Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Map Reduce (Word Count Problem code in Java):

Here, the role of Mapper is to map the keys to the existing values and the role of Reducer is to aggregate the keys of common values. So, everything is represented in the form of Key-value pair.

MapReduce consists of two steps:

- **Map Function** – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

Example – (Map function in Word Count)

Input	Set of data	Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN,BUS, buS, caR, CAR, car, BUS, TRAIN
Output	Convert into another set of data (Key,Value)	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

WordMapper.java:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class WordMapper extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable>{
```

```
    public void map(LongWritable key,Text value,
OutputCollector<Text,IntWritable> output, Reporter r)
    throws IOException{
        String s=value.toString();
        for (String word:s.split(" ")){
            if(word.length()>0){
                output.collect(new Text(word),new IntWritable(1));
            }
        }
    }
}
```

WordReducer.java:

- Reduce Function – Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

Example – (Reduce function in Word Count)

Input (output of Map function)	Set of Tuples	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)
Output	Converts into smaller set of tuples	(BUS,7), (CAR,7), (TRAIN,4)

```
public class WordReducer extends MapReduceBase implements
Reducer<Text,IntWritable,Text,IntWritable>{
    public void reduce(Text key,Iterator<IntWritable> values,
OutputCollector<Text,IntWritable> output,
    Reporter r) throws IOException{
        int count=0;
        while(values.hasNext()){
            IntWritable i=values.next();
            count+=i.get();
        }
        output.collect(key,new IntWritable(count));
    }
}
```

The Map class which extends the public class Mapper<KEYIN,VALUEIN,KEYOUT, VALUEOUT> and implements the Map function.

The Reduce class which extends the public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> and implements the Reduce function.

Driver Code in Map Reduce:

```
public class MyDriver{
    public static void main(String args[]) throws Exception{
        Configuration conf= new Configuration();
        Job job = new Job(conf,"My Word Count Program");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        Path outputPath = new Path(args[1]);
        //Configuring the input/output path from the filesystem into
the job
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        //deleting the output path automatically from hdfs so that
we don't have to delete it explicitly
        outputPath.getFileSystem(conf).delete(outputPath);
        //exiting the job only if the flag value becomes false
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Run the MapReduce code:

The command for running a MapReduce code is:

```
hadoop jar hadoop-mapreduce-example.jar WordCount
/sample/input /sample/output
```

- In the driver class, we set the configuration of our MapReduce job to run in Hadoop.
- We specify the name of the job, the data type of input/output of the mapper and reducer.
- We also specify the names of the mapper and reducer classes.
- The path of the input and output folder is also specified.
- The method setInputFormatClass () is used for specifying how a Mapper will read the input data or what will be the unit of work. Here, we have chosen TextInputFormat so that a single line is read by the mapper at a time from the input text file.

- The main() method is the entry point for the driver. In this method, we instantiate a new Configuration object for the job.

Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Record Reader: An InputFormat is also responsible for creating the **Input Splits** and dividing them into records. The data is divided into the number of splits (typically 64/128mb) in HDFS. This is called as inputsplit, which is the input that is processed by a single map.

InputFormat class calls the **getSplits()** function and computes splits for each file and then sends them to the **JobTracker**, which uses their storage locations to schedule map tasks to process them on the **TaskTrackers**. Map task then passes the split to the **createRecordReader()** method on InputFormat in task tracker to obtain a RecordReader for that split. The RecordReader load's data from its source and converts into key-value pairs suitable for reading by the **mapper.**

How the Record Reader Works: A RecordReader is more than iterator is over records, and map task which uses one record to generate key-value pair, which is passed to the map function.

Then nextKeyValue() will repeat on the context, to populate the key and value objects for the mapper. The key and value is retrieved from the record reader by way of context and passed to the map () method to do its work.

An input to the map function, which is a key-value pair (K, V), gets processed as per the logic mentioned in the map code. When the record gets to the end of the record, the nextKeyValue () method returns false.

Types of Record Reader:

1. **LineRecordReader:** LineRecordReader in Hadoop is the default RecordReader that textInputFormat provides, it treats each line of the input file as the new value, and associated key is byte offset.

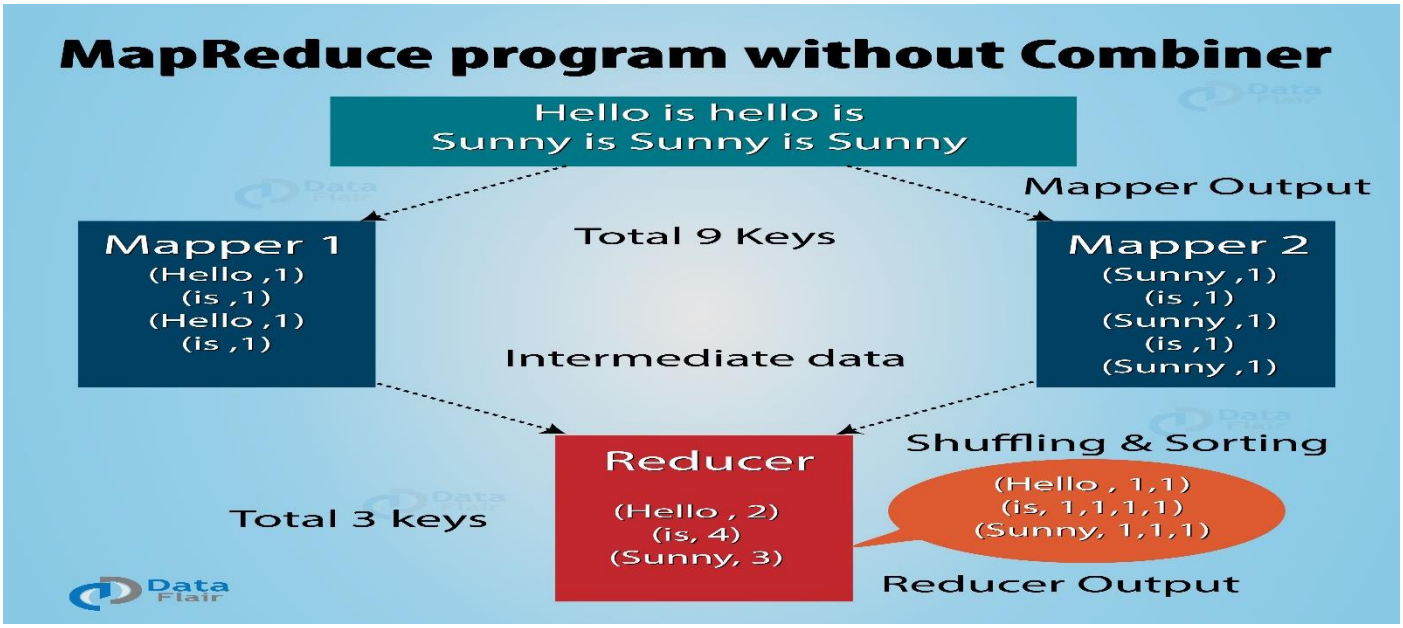
2. **Sequence File Record Reader:** It reads data specified by the header of a sequence file.

Combiner in Hadoop: Combiner is also known as "Mini-Reducer" that summarizes the Mapper output record with the same Key before passing to the Reducer.

On a large dataset when we run MapReduce, job. Therefore, Mapper generates large chunks of intermediate data. Then the framework passes this intermediate data on the Reducer for further processing.

This leads to enormous network congestion. The Hadoop framework provides a function known as Combiner that plays a key role in reducing network congestion.

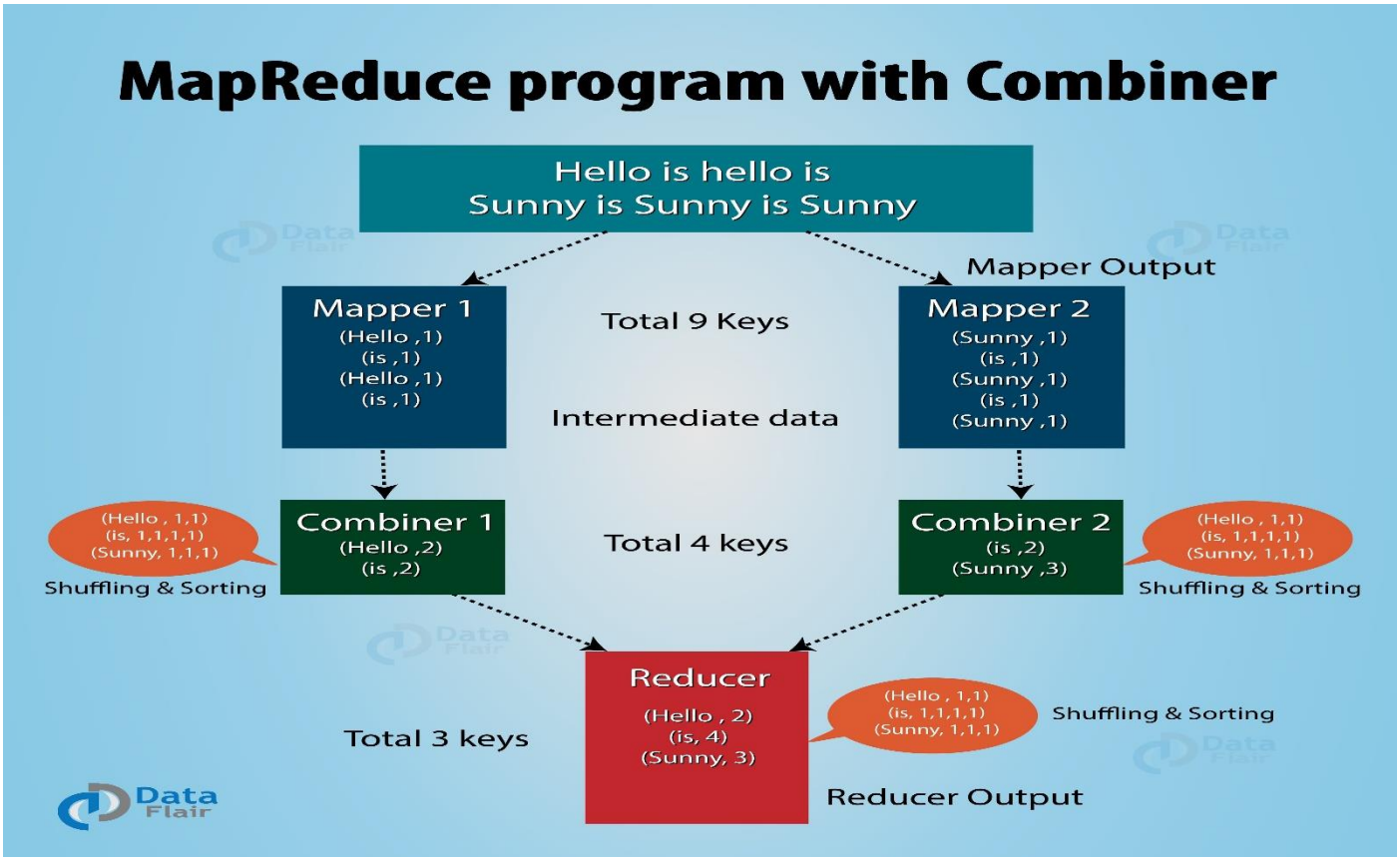
The primary job of Combiner a "Mini-Reducer is to process the output data from the Mapper, before passing it to Reducer. It runs after the mapper and before the Reducer. Its usage is optional.



Here Input is split into two mappers. The framework generates 9 keys from the mappers. Therefore, now we have (9 key/value) intermediate data. Further mapper sends this **key-value** directly

to the reducer. While sending data to the reducer, it consumes some network bandwidth. It takes more time to transfer data to reducer if the size of data is big.

Now in between mapper and reducer if we use a Hadoop combiner, then combiner shuffles intermediate data (9 key/value) before sending it to the reducer and generates 4 key/value pair as an output.



Reducer now needs to process only 4 key/value pair data which is generated from 2 combiners. Thus, reducer gets executed only 4 times to produce final output, which increases the overall performance.

Partitioner in Hadoop: Partitioning of the keys of the intermediate map output is controlled by the Partitioner. By hash function, key (or a subset of the key) is used to derive the partition.

According to the key-value each mapper output is partitioned and records having the same key value go into the same partition (within each mapper), and then each partition is sent to a reducer.

Partition class determines which partition a given (key, value) pair will go. Partition phase takes place after map phase and before reduce phase.

Note: The Default Hadoop Partitioner in Hadoop MapReduce is Hash Partitioner which computes a hash value for the key and assigns the partition based on this result.

--The total number of Partitioner that run in Hadoop is equal to the number of reducers i.e. Partitioner will divide the data according

to the number of reducers which is set by JobConf.setNumReduceTasks() method.

The default Partitioner in Hadoop is the HashPartitioner, which has a method called getPartition. It takes key.hashCode() & Integer.MAX_VALUE and finds the modulus using the number of reduce tasks.

Here is the code for the default Partitioner:

```
public class HashPartitioner<K, V> extends Partitioner<K, V> {
    public int getPartition(K key, V value, int numReduceTasks) {
        return (key.hashCode() & Integer.MAX_VALUE) %
numReduceTasks;
    }
}
```

Creating Custom Partitioner: Here we want to create the custom Partitioner that partition the key-value pair based on the key length.

```
public class MyPartitoner implements Partitioner<Text, IntWritable>{
    public void configure(JobConf conf){

    }
    public int getPartition(Text key,IntWritable value,int
numOfReducer){
        String s=key.toString();
        if(s.length()==1){
            return 0;
        }
        if(s.length()==2){
            return 1;
        }
        if(s.length()==3){
            return 2;
        }
        if(s.length()==4){
            return 3;
        }
    }
}
```

Here we can configure the Partitioner class in the Driver class main method by using this statement:

```
    conf.setPartitionClass(MyPartitoner.Class);
```

we can configure the number of reducer in the Driver class main method by using this statement:

```
conf.setNumReduceTasks (4);
```

We can work with upto part-00000 to part-99999(i.e. 100000) reducers.