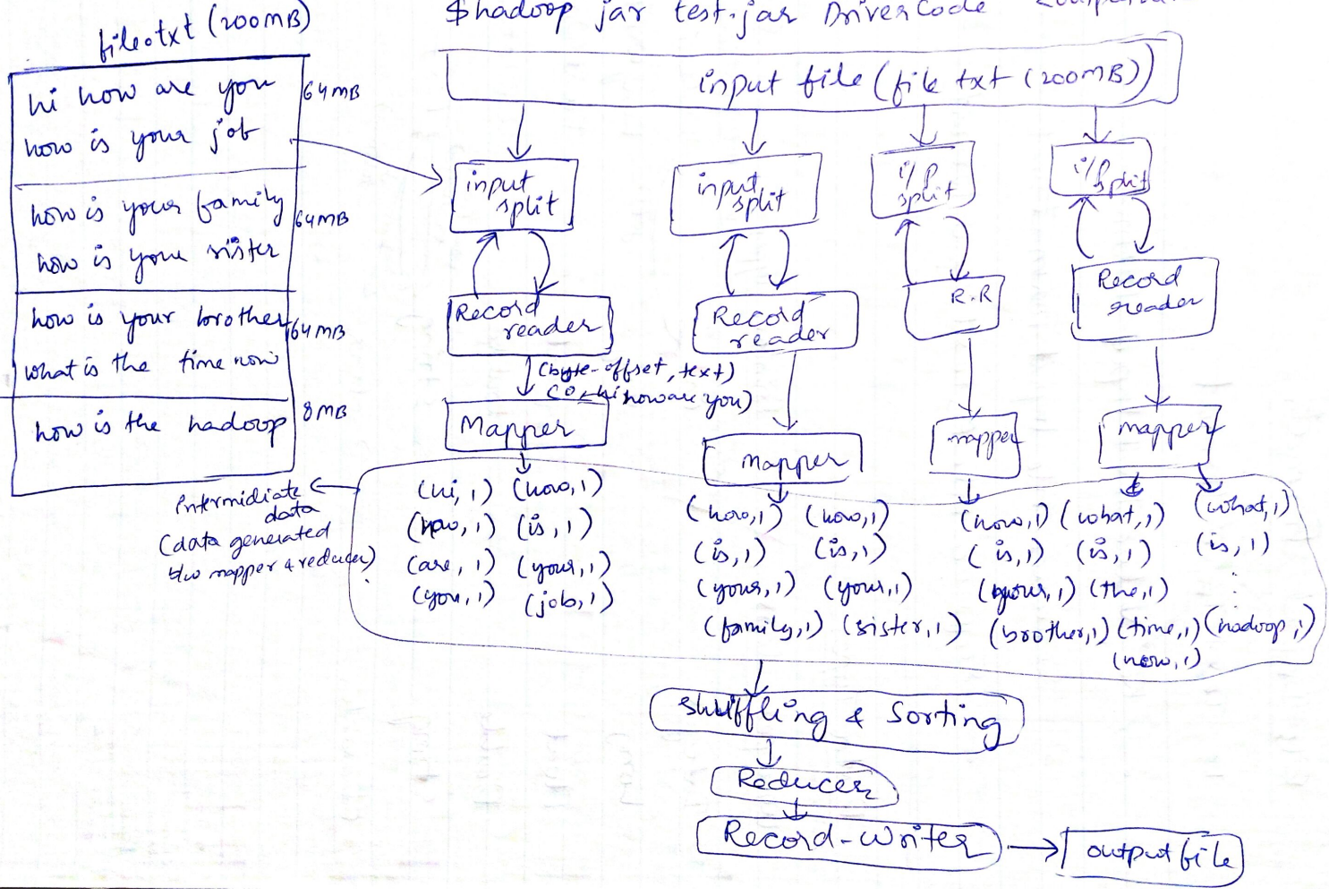
Map-Reduce Flow Chart

Word-Count-Job

$hadoop jar test.jar DriverCode <output dir>

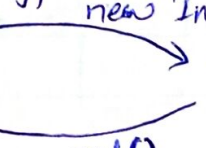


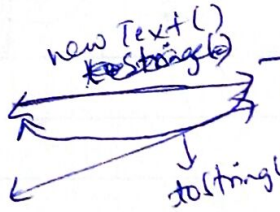
File Input Formats:

1. Text Input Format
2. Key Value Text Input Format
3. Sequence File Input Format
4. Sequence File As Text Input Format.

Data Types:

Wrapper class	Primitive Types	Box Classes
Integer	int	IntWritable
Long	long	LongWritable
Float	float	FloatWritable
Double	double	DoubleWritable
String	String	Text
Character	char	





Process Flow in Word-Count Job:

1. Input file is divided into blocks of 64MB

2. "Record Reader", takes each input split and produces, <key, value> pairs based on "input file", file format

   For example: "TextInputFormat" file generates, < LongWritable, Text > as o/p, where, "Byte Offset of each line is "LongWritable" key & "content of each line" as text.

3. Record reader does it in a loop till all the lines in input split are Completed.

4. Next, Mapper takes the output of Record reader and generates the

Key value pairs of below format

(byteoffset, text)

(0, hi how are you)

↓ Mapper (16, how is your job)

↓

(hi-1) (how-1)
(how-1) (is-1)
(are-1) (your-1)
(you-1) (job-1)

5. Next in the shuffling phase, it combines all the keys to form unique keys. For, example

(hi,1) (how,[1,1]) (are,1) (is,1) (your,1) (your,1) (job,1)

6. Next, in sorting phase, all the keys will be arranged in the sorting order

7. ~~Next:~~
7. Next in Reducer, phase all be reduced to unique keys & their respective counts.

(hi-1) (how,2) (are,1) (is,1) (your,1) (your,1) (job,1)

8. Then using the "RecordWriter", output will be written to file in the output directory

File Formats:

1. In "TextInputFormat", files will be made into (Key, Value) pairs of <Byte-offset, Text>.

* In Hadoop, everything is handled using the <KEY, VALUE> pairs format.

2. In "KeyValueTextInputFormat", file will be split based on tab. For example,

101 \t Ram 20
102 Krish 30 ⇒
103 Rahim 40

<101, (Ram, 20)>
<102, (Krish 30)>
<103, (Rahim 40)>

---

output directory files

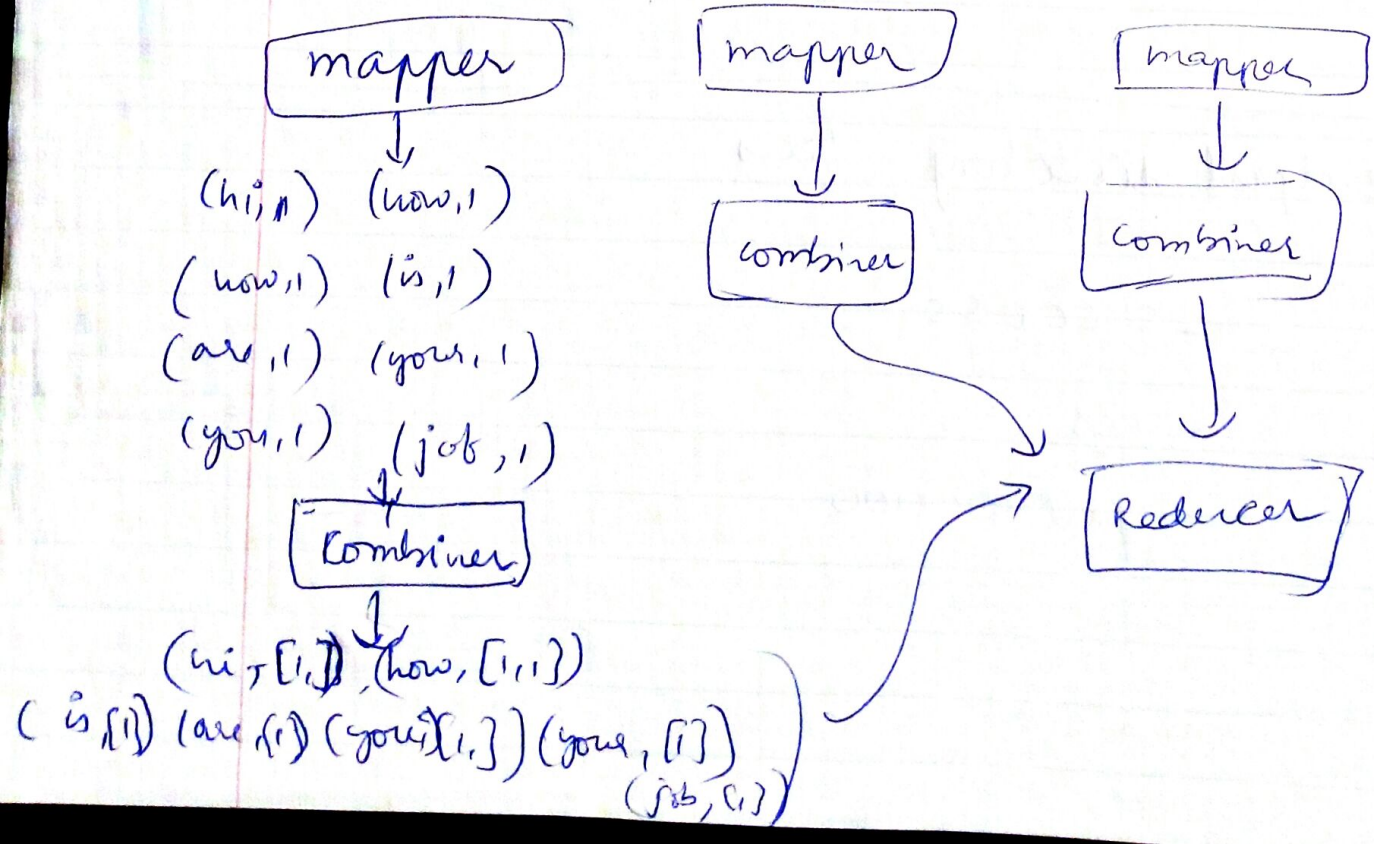
- _SUCCESS
- _logs
- part-00000

Combiners:

1. # of i/p splits = # of mappers.
x. # of reducer = # of o/p files
2. By default only "one reducer"

* To reduce the traffic on the network due to congestion (because of all mappers sending data (key value) pairs to reducers. We use "combiners".

* Combiner is a Mini-Reducer on each mapper node.

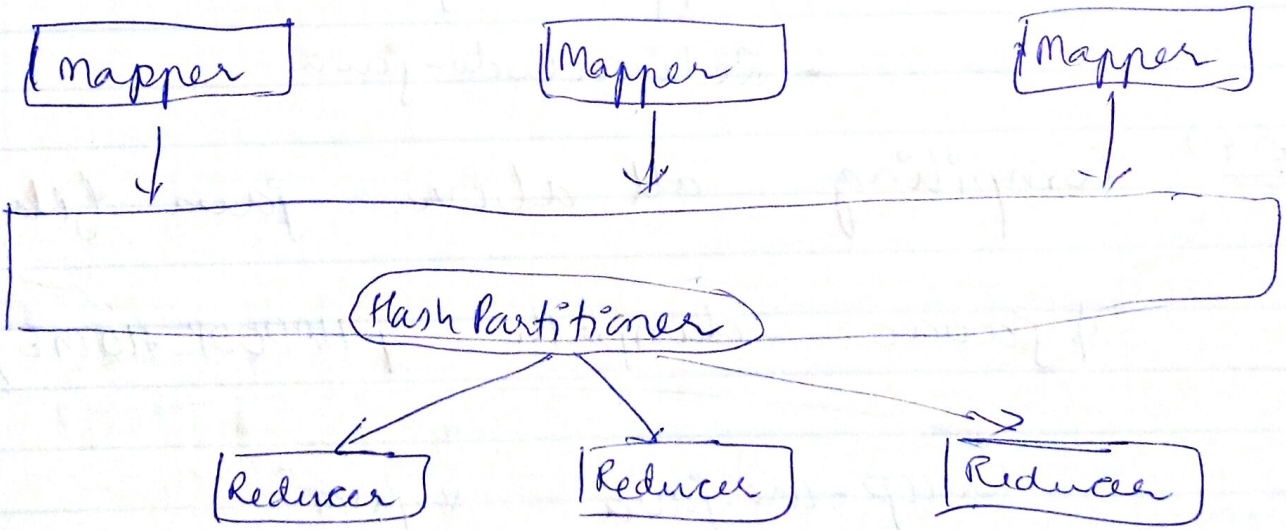


Partitioner

1) It is used to send specific, <Key, value> pairs to a specific Reducer.

2) It improves the performance & readability (for example, all keys of length '2' to one partitioner, keys of length '3' to the other etc...)

3) By default, "Hash Partitioner" is used, where it randomly allocates the <Key, Value> pairs to random reducers.
(based on Hash code of object)



Hello World Job (Sample Program)

Step 1) Create a file, file.txt

vi, (or) cat (or) touch

Step 2) loading file.txt from local file system to HDFS

$ hadoop fs -put file.txt file

Step 3) Writing programs

- Driver Code.java
- Mapper Code.java
- Reducer Code.java

Step 4) Compiling all above java files

$ javac -classpath $HADOOP_HOME/

hadoop-core.java *.java

Step 5) Creating jar files

(Hadoop can only run the jar files)

$ jar cvf test.jar *.class

Step6: Running above test.jar on file(which is there in hdfs).

~~$hadoop jar~~

$ hadoop jar test.jar DriverCode ~~file test output~~

file <test_output>

Hello World Job

Word Count Job

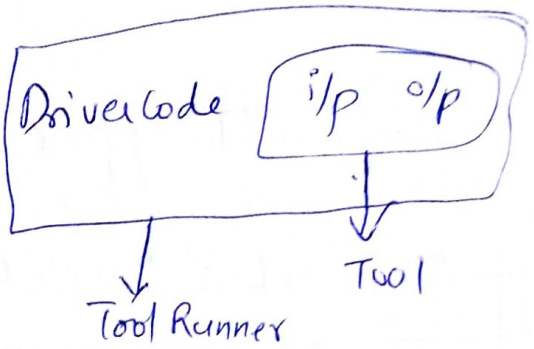
Driver Code :. WordCount.java

takes care of command line arguments

```
public class WordCount extends Configured implements
                Tool {

public int run (string args[]) throws Exception
{
    if (args.length<2)
    {
        System.out.println ("plz give i/p & o/p directories
                            properly")
        return -1;
    }

    JobConf conf = new JobConf (WordCount.class);

    conf.setJobName ("Word Count");
    FileInputFormat.setInputPaths(conf, new
                            Path(args[0]));
    FileOutputFormat.setOuputPath(conf, new
                            Path(args[1]);
```

```
conf.setMapperClass(WordMapper.class);
conf.setReducerClass(WordReducer.class);
conf.setMapOutputKeyClass(Text.class);
conf.setMapOutputValueClass(IntWritable.class);
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);

JobClient.runJob(conf);
return 0;
}

public static void main(String args[]) throws Exception
{
    int exitCode = ToolRunner.run(new
                   WordCount(), args);
    System.exit(exitCode);
}
```

$hadoop jar wc.jar



→

for more information on Main class, look below : ToolRunner class implementation

```
Class ToolRunner
{
    public static int run(Tool tool, String args[])
    {
        tool.run(args);
    }
}
```

Mapper Code :- WordMapper.java

```
public class WordMapper extends
    MapReduceBase implements Mapper<
    LongWritable, Text, Text, IntWritable>
```

LongWritable ← mapper i/p key type
Text ↓ mapper i/p value type
Text ↓ mapper o/p key
IntWritable → mapper o/p value type

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

Reducer Code:- WordReducer.java

```
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 {
{
        while (values.hasNext())
        {
            IntWritable i = values.next();
            count += i.get(); // conversion from
                              obj type to
                              primitive
                              type
        }
        output.collect(key, new IntWritable(count));
    }
}
```

Partitioner code:

```
public class MyPartitioner implements

Partitioner < Text, IntWritable >
{

    public void configure (JobConf conf)
    {

    }

    public int getPartition (Text key,
        IntWritable value, int setNumRedTasks)

{
        String s = key.toString();
        if (s.length() == 1)
        { return 0;
        }
        if (s.length() == 2)
        { return 1;
        }
        if (s.length() == 3)
        { return 2;
        }
        else { return 3; }
```

Configure Partitioner class in Driver code, to run it.

```
conf.setPartitionerClass(MyPartitioner.class);
conf.setNumReduceTasks(4);
```

* Maximum of reducers
= 99999
⇓ as
o/p file can be
part-00000
↓
part-99999.

Hadoop distributed file System (hdfs) commands:

1) hadoop fs

2) hadoop fs -ls
   - list the metadata (directory + files)

3) if multinode cluster ↓
   hadoop fs -ls &hdfs://<hostname>/user/<username>

4) create directory
   $ hadoop fs -mkdir <dir-name>
   → hyphen-added, compared to unix systems

5) Directly we cannot create file in hdfs, we have to create it in local file system & copy to hdfs.

6) copy file to hdfs
   $hadoop fs -put <src> <dest>
   ① local fs.
   (or)
   ~~-copyFromLocal~~
   ② -copyFromLocal
   (or) ③
   -moveFromLoc

⑦ Get the file from hdfs file system to local filesystem

hdfs ↑ local fs ↑

$hadoop fs -get <src> <dest>

(or)

- copyToLocal

(or)

- moveToLocal

8) *writing data directly into hdfs is not possible*

9) Viewing file content in hdfs

$hadoop fs -cat <file_name>

10) remove file from hdfs

$hadoop fs -rm <file_name>

11) remove file forcefully

$hadoop fs -rmr <directory file_name>

12) Copy file from one location to another

$hadoop fs -cp <src> <dest>

13) for moving files

$hadoop fs -mv <src> <dest>

14) Disk Usage:

$hadoop fs -du <file-name>

15) set replication factor

$hadoop fs -setrep <int> <file-name>