1: The InputSplit has defined a slice of work, but does not describe how to access it. The RecordReader class actually loads the data from its source and converts it into (key, value) pairs suitable for reading by the Mapper. The RecordReader instance is defined by the Input Format.

2: In case of 0 reducer, the output of mapper is the final output. Reduce step is skipped and mapper output will be final output. This job is called map only job.

3: Map side join is used when there is join operation on two files such that one file is much smaller compared to other. Since files are stored in HDFS so in case of map side join, smaller file will get stored by job tracker in each of the machine Destributed cache where blocks of bigger file is present.

Since each mapper task wil run in its own JVM memory, the file from distributed cache of the machine will get transferred in memory where JVM is running for mapper task. This mapper task will perform the join operation of bigger file split input and smaller file.

Each record of split of bigger fike will perform join with records of smaller file. And mapper will create key,value pair of the join operation.

Advantages of using map side join:

Map-side join helps in minimizing the cost that is incurred for sorting and merging in the shuffle and reduce stages.

Map-side join also helps in improving the performance of the task by decreasing the time to finish the task.

Disadvantages of Map-side join:

Map side join is adequate only when one of the tables on which you perform map-side join operation is small enough to fit into the memory. Hence it is not suitable to perform map-side join on the tables which are huge data in both of them.

Reduce side join take place at reduce side. The joining process is completed off memory, so there is no issue related to size of data of two files. Both files can be large.

Reduce side join are straight forward due to the fact that Hadoop sed identical keys to the same reducer, so by default data is organized for us.

4: conf.setMapperClass is used to set Mapper class and all stuffs related to map job such as reading a data and generating key/value pair out of mapper.

5: A counter in MapReduce is a mechanism used for collecting statistical information about the MapReduce job. This information could be useful for diagnosis of a problem in MapReduce job processing.

Counters are similar to putting log message in the code for map or reduce.

Typically, these counters are defined in a program (map or reduce) and are incremented during execution when a particular event or condition (specific to that counter) occurs.

A very good application of counters is to track valid and invalid records from an input dataset.

Two types of counters:

1. Hadoop Built-In counters: There are some built-in counters which exist per job. Below are built-in counter groups-

MapReduce Task Counters - Collects task specific information (e.g., number of input records) during its execution time.

FileSystem Counters - Collects information like number of bytes read or written by a task

FileInputFormat Counters - Collects information of number of bytes read through FileInputFormat

FileOutputFormat Counters - Collects information of number of bytes written through FileOutputFormat

Job Counters - These counters are used by JobTracker. Statistics collected by them include e.g., number of task launched for a job.

2. User Defined Counters

In addition to built-in counters, user can define his own counters using similar functionalities provided by programming languages. For example, in Java 'enum' are used to define user defined counters.

An example MapClass with Counters to count the number of missing and invalid values:

public static class MapClass

extends MapReduceBase

implements Mapper<LongWritable, Text, Text, Text>

{

static enum SalesCounters { MISSING, INVALID };

public void map ( LongWritable key, Text value,

OutputCollector<Text, Text> output,

Reporter reporter) throws IOException

{

//Input string is split using ',' and stored in 'fields' array

String fields[] = value.toString().split(",", -20);

//Value at 4th index is country. It is stored in 'country' variable

String country = fields[4];

//Value at 8th index is sales data. It is stored in 'sales' variable

String sales = fields[8];

if (country.length() == 0) {

reporter.incrCounter(SalesCounters.MISSING, 1);

} else if (sales.startsWith("\"")) {

reporter.incrCounter(SalesCounters.INVALID, 1);

} else {

output.collect(new Text(country), new Text(sales + ",1"));

}

}

}

Here, SalesCounters is a counter defined using 'enum'. It is used to count MISSING and INVALID input records.

In the code snippet, if 'country' field has zero length then its value is missing and hence corresponding counter SalesCounters.MISSING is incremented.

Next, if 'sales' field starts with a " then the record is considered INVALID. This is indicated by incrementing counter SalesCounters.INVALID.

6: There is no custom partitioner in pig.Let us take a scenario where we want to count the population in two cities. I have a data set and sensor list of different cities. I want to count the population by using one mapreduce for two cities. Let us assume that one is Bangalore and the other is Noida. So I need to consider key of Bangalore city similar to Noida through which I can bring the population data of these two cities to one reducer. The idea behind this is some how I have to instruct map reducer program – whenever you find city with the name ‘Bangalore‘ and city with the name ‘Noida’, you create the alias name which will be the common name for these two cities so that you create a common key for both the cities and it get passed to the same reducer. For this, we have to write custom partitioner.

In mapreduce when you create a ‘key’ for city, you have to consider ’city’ as the key. So, whenever the framework comes across a different city, it considers it as a different key. Hence, we need to use customized partitioner. There is a provision in mapreduce only, where you can write your custom partitioner and mention if city = bangalore or noida then pass similar hashcode. However, we cannot create custom partitioner in Pig. As Pig is not a framework, we cannot direct execution engine to customize the partitioner. In such scenarios, MapReduce works better than Pig.

8: The Combiner is a "mini-reduce" process which operates only on data generated by a mapper. The Combiner will receive as input all data emitted by the Mapper instances on a given node. The output from the Combiner is then sent to the Reducers, instead of the output from the Mappers.