1. Explain the different methods of

● mapper class and

● reducer class

Hadoop was designed to work on key and value pairs only. i.e, the mapper and reducer function can only work on the input file key and value pairs. So, before reaching the mapper part the corresponding input split will be converted into the key value pairs according to the type of the input file. And these implementation is done by a predefined interface in Hadoop framework known as Record Reader.

Record reader is a predefined interface which knows only to read one record (line) at a time from its corresponding input split and it converts the entire record (line) into key, value pairs depending on the file format.

In our case we have considered a text input format file where the input key will be byte offset of the beginning of the line from the beginning of the file and the value will be the entire record (row / line)

Once the Record reader converts a line into key value pairs these key value pairs will be sent to map method via mapper class to perform map code operation on the input splits.

Mapper Class

The Mapper class defines the Map job. Maps input key-value pairs to a set of intermediate key-value pairs. Maps are the individual tasks that transform the input records into intermediate records. The transformed intermediate records need not be of the same type as the input records. A given input pair may map to zero or many output pairs.

Method

map is the most prominent method of the Mapper class. The syntax is defined below −

map(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Mapper.Context context)

This method is called once for each key-value pair in the input split.

Once the Mapper class output is stored in the output context buffer these output key value pairs will be passed to the shuffle and sort phase and are further sent to the reducer phase where the aggregation of the values is performed.

setup() and cleanup() are methods you can override, if you choose, and they are there for you to initialize and clean up your map/reduce tasks. You actually don't have access to any data from the input split directly during these phases. The lifecycle of a map/reduce task is (from a programmer's point of view):

setup -> map -> cleanup

setup -> reduce -> cleanup

What typically happens during setup() is that you may read parameters from the configuration object to customize your processing logic.

What typically happens during cleanup() is that you clean up any resources you may have allocated. There are other uses too, which is to flush out any accumulation of aggregate results.

Maps input key/value pairs to a set of intermediate key/value pairs.

Maps are the individual tasks which transform input records into a intermediate records. The transformed intermediate records need not be of the same type as the input records. A given input pair may map to zero or many output pairs.

The Hadoop Map-Reduce framework spawns one map task for each InputSplit generated by the InputFormat for the job. Mapper implementations can access the Configuration for the job via the JobContext.getConfiguration().

The framework first calls setup(org.apache.hadoop.mapreduce.Mapper.Context), followed by map(Object, Object, org.apache.hadoop.mapreduce.Mapper.Context) for each key/value pair in the InputSplit. Finally cleanup(org.apache.hadoop.mapreduce.Mapper.Context) is called.

All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to a Reducer to determine the final output. Users can control the sorting and grouping by specifying two key RawComparator classes.

The Mapper outputs are partitioned per Reducer. Users can control which keys (and hence records) go to which Reducer by implementing a custom Partitioner.

Users can optionally specify a combiner, via Job.setCombinerClass(Class), to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer.

Applications can specify if and how the intermediate outputs are to be compressed and which CompressionCodecs are to be used via the Configuration.

If the job has zero reduces then the output of the Mapper is directly written to the OutputFormat without sorting by keys

Reducer:

Reducer has 3 primary phases:

Shuffle

The Reducer copies the sorted output from each Mapper using HTTP across the network.

Sort

The framework merge sorts Reducer inputs by keys (since different Mappers may have output the same key).

The shuffle and sort phases occur simultaneously i.e. while outputs are being fetched they are merged.

SecondarySort

To achieve a secondary sort on the values returned by the value iterator, the application should extend the key with the secondary key and define a grouping comparator. The keys will be sorted using the entire key, but will be grouped using the grouping comparator to decide which keys and values are sent in the same call to reduce.The grouping comparator is specified via Job.setGroupingComparatorClass(Class). The sort order is controlled by Job.setSortComparatorClass(Class).

For example, say that you want to find duplicate web pages and tag them all with the url of the "best" known example. You would set up the job like:

Map Input Key: url

Map Input Value: document

Map Output Key: document checksum, url pagerank

Map Output Value: url

Partitioner: by checksum

OutputKeyComparator: by checksum and then decreasing pagerank

OutputValueGroupingComparator: by checksum

Reduce

In this phase the reduce(Object, Iterable, org.apache.hadoop.mapreduce.Reducer.Context) method is called for each <key, (collection of values)> in the sorted inputs.

The output of the reduce task is typically written to a RecordWriter via TaskInputOutputContext.write(Object, Object).

The output of the Reducer is not re-sorted.

Example:

public class IntSumReducer<Key> extends Reducer<Key,IntWritable,

Key,IntWritable> {

private IntWritable result = new IntWritable();

public void reduce(Key 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);

}

}

setup

protected void setup(org.apache.hadoop.mapreduce.Reducer.Context context)

throws IOException,

InterruptedException

Called once at the start of the task.

Throws:

IOException

InterruptedException

reduce

protected void reduce(KEYIN key,

Iterable<VALUEIN> values,

org.apache.hadoop.mapreduce.Reducer.Context context)

throws IOException,

InterruptedException

This method is called once for each key. Most applications will define their reduce class by overriding this method. The default implementation is an identity function.

Throws:

IOException

InterruptedException

cleanup

protected void cleanup(org.apache.hadoop.mapreduce.Reducer.Context context)

throws IOException,

InterruptedException

Called once at the end of the task.

Throws:

IOException

InterruptedException

run

public void run(org.apache.hadoop.mapreduce.Reducer.Context context)

throws IOException,

InterruptedException

Advanced application writers can use the run(org.apache.hadoop.mapreduce.Reducer.Context) method to control how the reduce task works.

Throws:

IOException

InterruptedException