Using Raw Hadoop

Sunday, February 28, 2010
8:34 AM

So far, we've been protected from the full complexity of hadoop by using Pig. Let's see what we've been missing!

Hadoop

- Yahoo's open-source MapReduce implementation
- Reference: http://developer.yahoo.com/hadoop/tutorial/

Two main parts

- A **distributed file system** (HDFS) that can store large amounts of data
- A **Map/Reduce programming model** that enables scalable data processing.

Components of a hadoop application

- A **distributed filesystem** that makes data available to multiple compute nodes.
- A **driver** that institutes the MapReduce operation.
- A **mapper** that reads data and maps it.
- A **combiner** that reduces the size of mapped data.
- A **reducer** that summarizes and stores results in the HDFS filesystem.

Note: the output is not a **stream**, but rather, a **distributed file.**

Hadoop MapReduce

- A bit different than Google MapReduce

Five phases of a MapReduce query

- **put**: put data on local disks
- **map**: map keys in data to values; store on mapper.
- **shuffle**: sort ranges of map output to reducers.
  - During shuffle, **combine** reduces size of data.
- **reduce**: store one result per unique key on disk.
- **get**: get results off disk.

For simplicity

Monday, March 01, 2010
5:11 PM

For simplicity,

we describe a typical hadoop example

in which mapper workers are also reducer workers.

In fact, in a real Hadoop implementation, you need to declare three kinds of nodes:

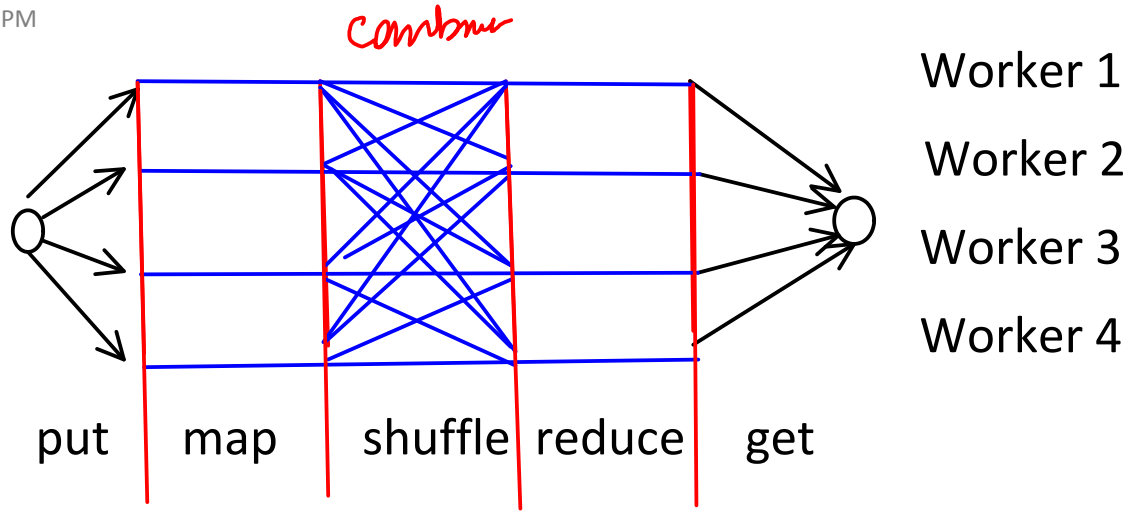
Trackers that start a job and finish it.

Mappers that contain HDFS and do the operation.

Reducers that don't contain HDFS and find final results.

A picture of Hadoop MapReduce Processing

Monday, March 01, 2010
3:07 PM



Often, the same nodes are mappers and reducers.
This picture is a bit deceptive, because phases begin **asynchronously**.
If one mapper gets done mapping early, it will start shuffling **before** the others.
Combiner can reduce data **during** the shuffle; **reducer** is called **after shuffling** is complete.

Understanding shuffle

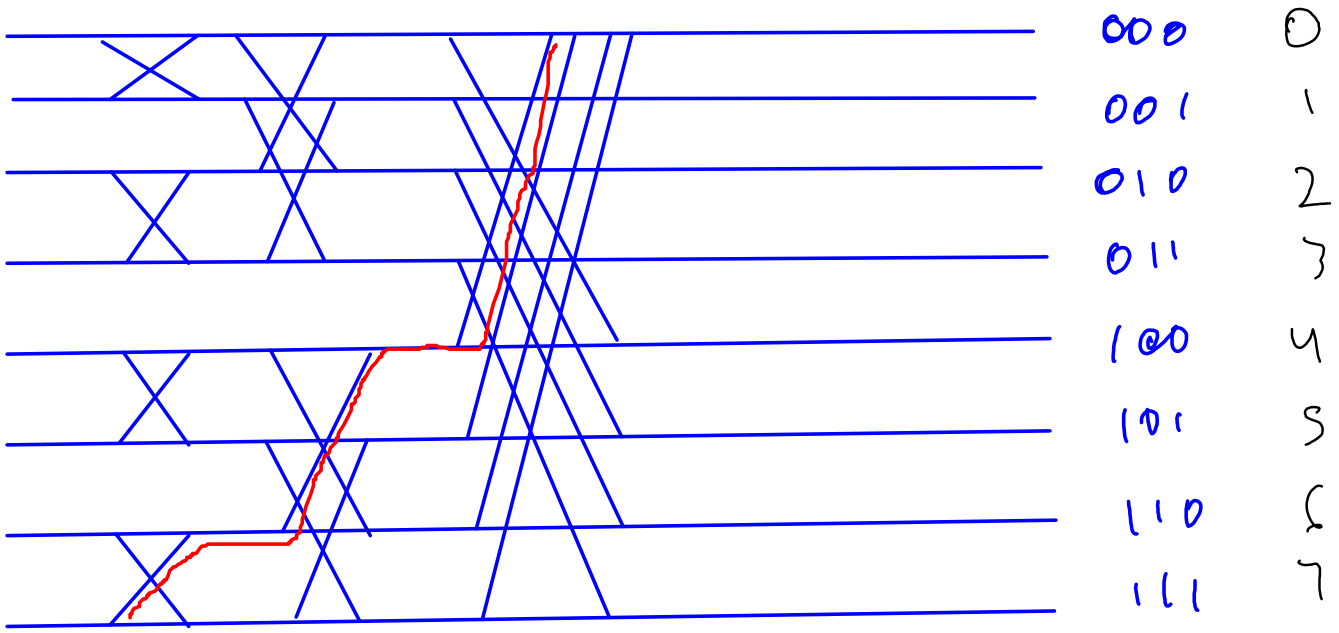
Monday, March 01, 2010
3:22 PM

Understanding shuffle sort

Number processers from 0-n in binary.

Take turns flipping one bit; send to neighbor with that bit flipped.

"Butterfly sort"



Example of shuffle sort

Monday, March 01, 2010
4:46 PM

Understanding butterfly sort

Number processors 0-n.

Map each key to hash h(k) between 0 and n.

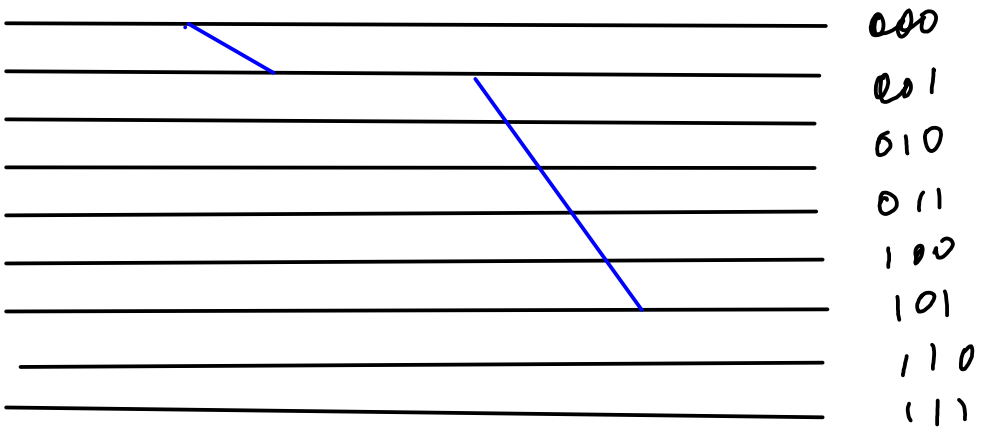
At step s (from 0 to log2(n)),

if k is at processor p and h(k) differs from p in the sth bit, send <k,v> to processor p^(1<<s) (^=xor).

Suppose key k is at worker $0=000_2$ and needs to be at worker $5=h(k)=101_2$ in binary

First worker $0=000_2$ sends it to worker $1 = 001_2$.

Then worker $1=001_2$ sends it to worker $5 = 101_2$.



Theorem: a key always gets to its intended destination in O(log(N)) steps, where N is the number of workers.

An example application

Sunday, February 28, 2010
1:24 PM

An example application: word count

Input: a set of **text files,** concatenated into a big text file

Mapper: for each word w, generate pairs <word, 1>, where w is a key.

Reducer: for each w, sum up 1's to get count of that word.

Driver: orchestrate calling mapper/reducer.

What we don't write

Monday, March 01, 2010
5:13 PM

We provide the mapper, reducer, combiner, and driver. Optionally, we provide input and output classes (to read and write individual files), and key and value classes (to store as map output).
We don't write the overall data reader, key shuffler, or data writer.

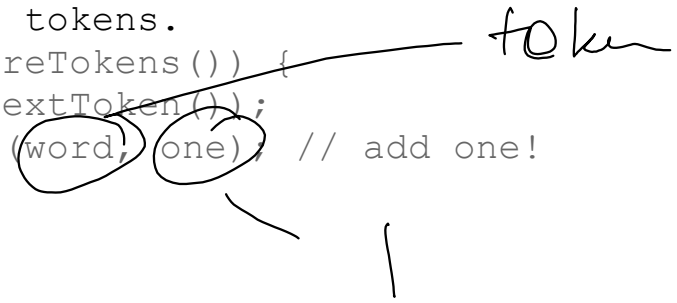
Sunday, February 28, 2010
8:43 PM

The mapper

Input: **shards** (fragments) of the text to be counted.

Output: a list of <w,1> pairs for w a word.

```
// a value of 1 of the appropriate type for reducing
private final IntWritable one = new IntWritable(1);


public void map(WritableComparable key, Writable value,
  OutputCollector<Text, IntWritable> output, Reporter reporter)
throws IOException {
  // WritableComparable means one can apply >, =, < to it.
  // Writable means that one can write it,
  // but might *not* be able to compare it.
  // WritableComparable is a subinterface of Writable.

  // each line is a shard, as a string.
  String line = value.toString();

  // break line into a sequence of tokens
  StringTokenizer itr = new StringTokenizer(line.toLowerCase());

  // iterate over tokens.
  while(itr.hasMoreTokens()) {
   word.set(itr.nextToken());
   output.collect(word, one); // add one!
  }
 }
```



Sunday, February 28, 2010
8:51 PM

Taking the mapper apart:

```
public void map(WritableComparable key, Writable value,
  OutputCollector<Text, IntWritable> output, Reporter
reporter)
```

`WritableComparable:`
an interface requiring support for set, <, ==, >.

`Writable:`
an interface requiring support for just set.
`(WritableComparable extends Writable)`

`OutputCollector:`
a place to put output from the mapping process, e.g., via `output.collect(word, one)`

`Reporter:`
a way to communicate problems to the user.

Picture of mapping

Sunday, February 28, 2010
9:39 PM

"Now is the time for all good gerbils to come to the aid of their country."

becomes:

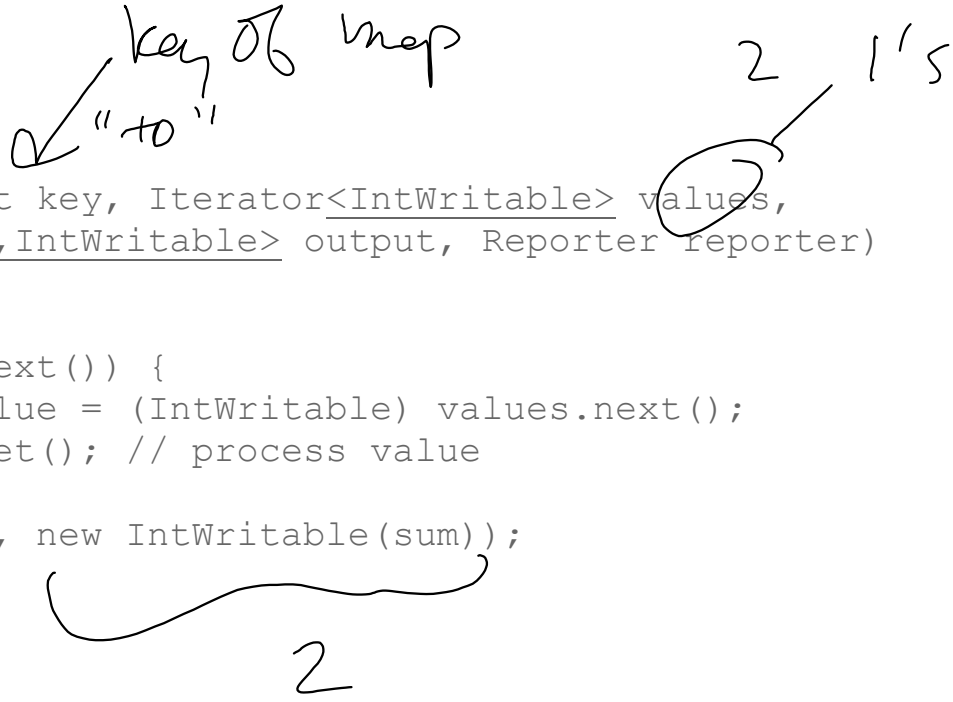
now,1	is,1	the,1	time,1	for,1	all,1	good,1
gerbils,1	to,1	come,1	to,1	the,1	aid,1	of,1
their,1	country,1					

The reducer

Sunday, February 28, 2010
8:49 PM

The reducer:

- Accepts <w, 1> pairs.
- Sums up the 1's.




```
public void reduce(Text key, Iterator<IntWritable> values,
  OutputCollector<Text,IntWritable> output, Reporter reporter)
throws IOException {
    int sum = 0;
    while (values.hasNext()) {
        IntWritable value = (IntWritable) values.next();
        sum += value.get(); // process value
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

Taking the reducer apart

Sunday, February 28, 2010
8:58 PM

Taking the reducer apart:

```
public void reduce(Text key,
Iterator<IntWritable> values,
   OutputCollector<Text,IntWritable> output,
   Reporter reporter)
```

`Text key:`

Word to be counted

`Iterator values:`

Sequence of ones for that word. These are `IntWritable:`

```
IntWritable value = (IntWritable)
values.next();
```

`IntWritable value:`

Mutable version of a value from an iterator, as in:

```
sum += value.get();
```

`OutputCollector output`

Place to put resulting sum, e.g., via

```
output.collect(key, new IntWritable(sum));
```

`Reporter reporter:`

Way to inform user of problems.

Picture of reducing

Sunday, February 28, 2010
9:42 PM

now,1	is,1	the,1	time,1	for,1	all,1	good,1
gerbils,1	to,1	come,1	to,1	the,1	aid,1	of,1
their,1	country,1					

becomes

aid,1	all,1	come,1	country,1	for,1	gerbils,1	good,1	is,1
now,1	of,1	the,2	their,1	time,1	to,2		

A driver

Sunday, February 28, 2010
9:07 PM

A driver

```
public static void main(String[] args) {
  // Create a "configuration" for a job
  JobConf conf = new JobConf(WordCount.class);

  // specify output types
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);

  // specify input and output directories in filesystem
  FileInputPath.addInputPath(conf, new Path("input"));
  FileOutputPath.addOutputPath(conf, new Path("output"));

  // specify a mapper
  conf.setMapperClass(WordCountMapper.class);

  // a reducer produces a final answer.
  conf.setReducerClass(WordCountReducer.class);
  // a combiner produces intermediate values.
  conf.setCombinerClass(WordCountReducer.class);

  try {
   // try to do it.
   JobClient.runJob(conf);
  } catch (Exception e) {
   e.printStackTrace();
  }
}
```

Taking the driver apart

Creating a "job" for the Hadoop cluster:

```
// Create a "configuration" for a job
JobConf conf = new JobConf(WordCount.class);

// … set configuration parts

try {
 // try to do it.
 JobClient.runJob(conf);
} catch (Exception e) {
 e.printStackTrace();
}
```

Configuring a "job":

Sunday, February 28, 2010
9:17 PM

Configuring a "job":

Specify output types:

```
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
```

Specify input and output directories in filesystem

```
FileInputPath.addInputPath(conf, new
Path("input"));
FileOutputPath.addOutputPath(conf, new
Path("output"));
```

Specify a mapper

```
conf.setMapperClass(WordCountMapper.class);
```

Specify a reducer (a combiner is a partial reducer)

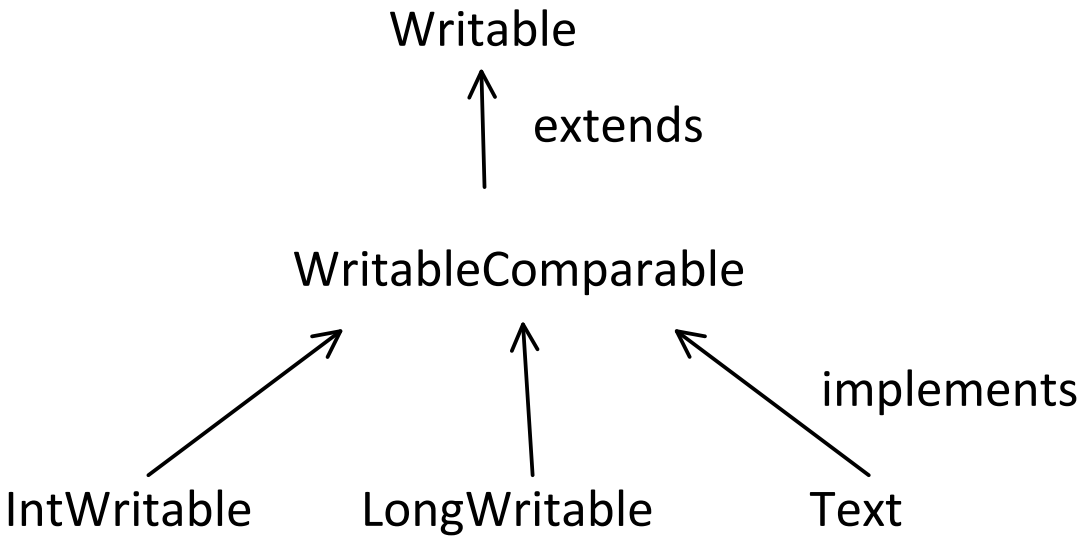
```
conf.setReducerClass(WordCountReducer.class);
conf.setCombinerClass(WordCountReducer.class);
```

Some finer points

- Data are contained in "smart containers"
  - IntWritable is an integer that is writable.
  - LongWritable likewise.
- Keys must be **Comparable.**
- Values must be **Writable.**

The interface hierarchy

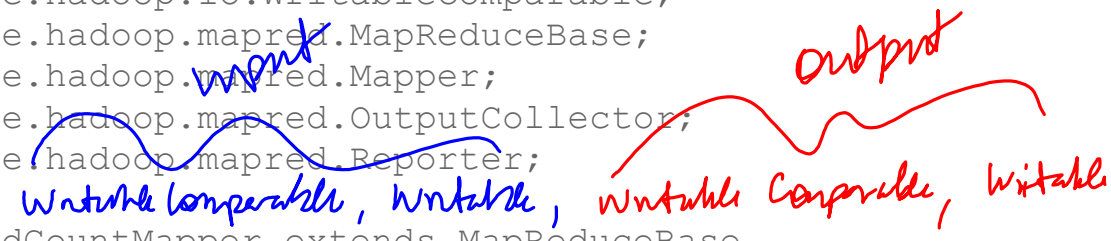
Monday, March 01, 2010
10:31 AM



A complete mapper

Sunday, February 28, 2010
1:11 PM

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WordCountMapper extends MapReduceBase
implements Mapper<LongWritable, Text, Text, IntWritable> {

 private final IntWritable one = new IntWritable(1);
 private Text word = new Text();

 // do a map operation: count words in a given string
 public void map(WritableComparable key, Writable value,
  OutputCollector<Text, IntWritable> output, Reporter reporter)
throws IOException {
  String line = value.toString();
  StringTokenizer itr = new StringTokenizer(line.toLowerCase());
  while(itr.hasMoreTokens()) {
   word.set(itr.nextToken());
   output.collect(word, one); // add one!
  }
 }
}
```

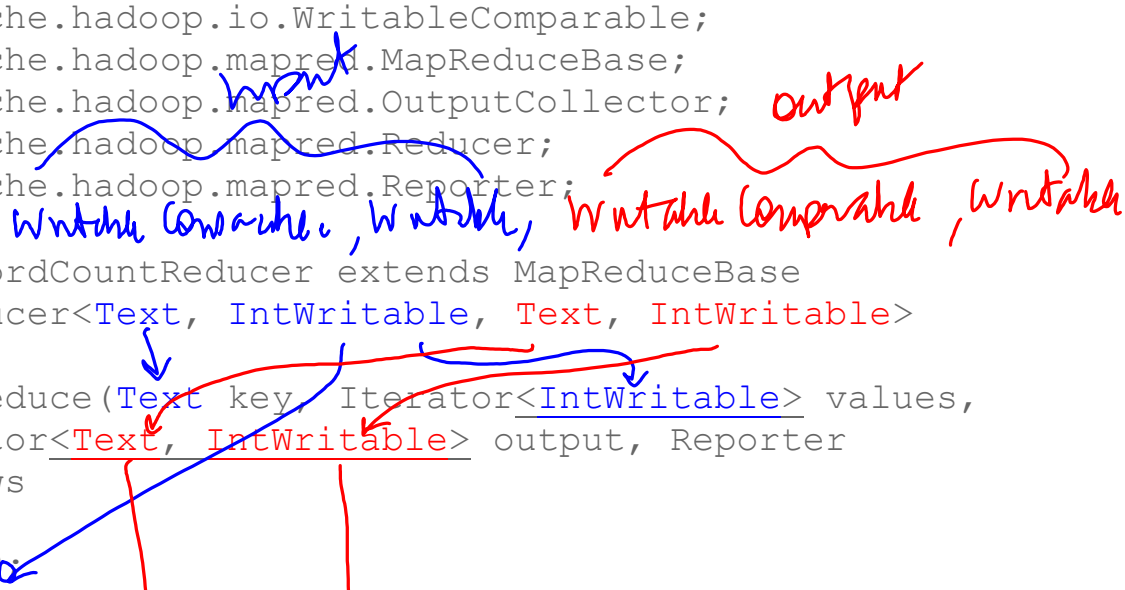


Pasted from <http://developer.yahoo.com/hadoop/tutorial/module3.html>

A complete reducer

Sunday, February 28, 2010
1:16 PM

```
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WordCountReducer extends MapReduceBase
implements Reducer<Text, IntWritable, Text, IntWritable>
{
 public void reduce(Text key, Iterator<IntWritable> values,
  OutputCollector<Text, IntWritable> output, Reporter
reporter) throws
IOException {
    int sum = 0;
    while (values.hasNext()) {
        IntWritable value = (IntWritable) values.next();
        sum += value.get(); // process value
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

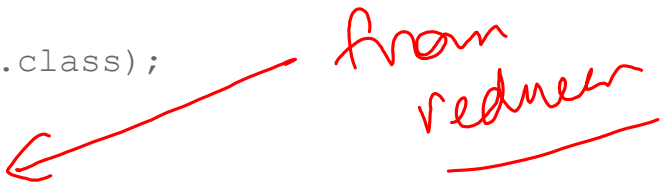


Pasted from <http://developer.yahoo.com/hadoop/tutorial/module3.html>

A complete driver

Sunday, February 28, 2010
1:19 PM

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;

public class WordCount {

 public static void main(String[] args) {

  JobConf conf = new JobConf(WordCount.class);

  // specify output types
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);

  // specify input and output dirs
  FileInputPath.addInputPath(conf, new Path("input"));
  FileOutputPath.addOutputPath(conf, new Path("output"));

  // specify a mapper
  conf.setMapperClass(WordCountMapper.class);

  // specify a reducer
  conf.setReducerClass(WordCountReducer.class);
  conf.setCombinerClass(WordCountReducer.class);

  try {
   JobClient.runJob(conf);
  } catch (Exception e) {
   e.printStackTrace();
  }
 }
}
```



Pasted from <http://developer.yahoo.com/hadoop/tutorial/module3.html>

Some missing details

- How does MapReduce know the input format of data files?
  - conf.setInputFormat(TextInputFormat.class);
  - conf.setOutputFormat(TextOutputFormat.class);

Chained computations

Monday, March 01, 2010
3:47 PM

Suppose you want to search for documents containing three words.

Map outputs key=URL, value=word

Reduce by ignoring all URLs that don't have all three.

Now suppose you want a fourth word too

Start with URLs that contain all three

Map outputs key=URL, value=4th word where we only consider URLs that contain other three.

Result: no repeated work.

Distributed files

Monday, March 01, 2010
11:28 AM

Copy input files to hadoop's distributed file system.

```
hadoop fs -put local_input input
```

Run an instance of an algorithm:

```
hadoop jar example.jar arguments...
```

Copy the output files from the distributed filesystem to the local filesytem and examine them:

```
hadoop fs -get output local_output
cat local_output/*
```

Typically,

Input is any number of text files.

Output is one file `part-00000` containing the result.

What Pig does

- Automatically insures that Java code is type-conformal (via schemas)
- Automatically writes the java code and creates a jar.
- Schemas control input/output form.
- Automatically puts the output in a local form.
- Uses a lot of temporary files.