CountLinks

CountLinks is a MapReduce job that simply counts the outbound links on a page. If you are not familiar with MapReduce please read about it on the Hadoop site.

This tutorial will show how to create a MapReduce task, configure a job to run it and how to feed data in and get data out. The task is simple to keep extra code to a minimum.

Create the Mapper

The mapper recieves the initial data and performs some operation on it. The map method that you supply get called by the framework. The first value is the key that we will use to identify the results associated with this document. If multiple map operations have the same key they will be combined and sent to the reducer. The second is the value. In our case the value is actually a ParseData so we will cast it to get access to it. OutputCollector is supplied bt the framework and you want to write the results of your map implementation as a key/value pair using OutputCollector.collect(). Don't worry about the Reporter for now.

How do I know what will be passed to the map function? This map function processes data from the parse_data directory of a nutch segment. You will see later how we set this up but for now I know that the parse_data is a SequenceFile that has the format of (Url,ParseData) as a key/value pair. The key (Url) is a Hadoop Text object. Each of these pairs will be passed to the map function. Depending upon what data you need to process you may need to examine the Nutch code to determine the format of the data.

{{{ public static class CounterMapper extends MapReduceBase implements Mapper
{
public void map(WritableComparable key, Writable value, OutputCollector collector, Reporter reporter) throws IOException {
// TODO Auto-generated method stub
ParseData data = (ParseData)value;

IntWritable outboundLinkCount = new IntWritable(data.getOutlinks().length);

collector.collect(key, outboundLinkCount);
}

public void close() throws IOException {
// TODO Auto-generated method stub
super.close();
}

public void configure(JobConf arg0) {
// TODO Auto-generated method stub
super.configure(arg0);
}

}
}}}

Create the Reducer

Our reducer does very little. We already know that the iterator will hold 1 item, which is an IntWritable that we created in the map function. To see a reducer that combines multiple items checkout the WordCount example in Hadoop source code. We simply write the data back to the Output collector.

```
        public static class CounterReducer extends MapReduceBase implements Reducer
        {

                public void reduce(WritableComparable url, Iterator iterator, OutputCollector output, Reporter
reporter) throws IOException {
                        IntWritable linkCount = (IntWritable)iterator.next();
                        output.collect(url, linkCount);
                }

                public void close() throws IOException {
                        // TODO Auto-generated method stub
                        super.close();
                }

                public void configure(JobConf arg0) {
                        // TODO Auto-generated method stub
                        super.configure(arg0);
                }


        }
```

Configure the job to run

So the last thing that we need to do is setup the job to run. The program is called with the parameters input directory and output directory respectively. The input directory is a specific segment. To process more data you would need to enumerate your segments and add them as paths for MapReduce. That is beyond the scope of this tutorial.

First we get the configuration for Nutch. Then we perform basic job setup. We set the input and out file formats. We set the format for the output key/value pair. Then we set the Classes that we created so that MapReduce can call them. Finally we pass the input and output directories and start the job.

```
        public static void main(String[] args) throws IOException{
            Configuration config = NutchConfiguration.create();

            JobConf jobConfig = new NutchJob(config);
            jobConfig.setJobName("countlinks");

            jobConfig.setInputFormat(SequenceFileInputFormat.class);

            jobConfig.setOutputFormat(MapFileOutputFormat.class);

            // the keys are words (strings)
            jobConfig.setOutputKeyClass(Text.class);
            // the values are counts (ints)
            jobConfig.setOutputValueClass(IntWritable.class);

            jobConfig.setMapperClass(CounterMapper.class);
            jobConfig.setCombinerClass(CounterReducer.class);
            jobConfig.setReducerClass(CounterReducer.class);

            jobConfig.setInputPath(new Path((String) args[0], ParseData.DIR_NAME));
            jobConfig.setOutputPath(new Path((String) args[1]));

            JobClient.runJob(jobConfig);
        }
```

TutorialOneCompleteSourceListing