Implementing the Tool interface for MapReduce driver

Usually people run MapReduce job using a driver code that is executed though its static main method. The downside here, all the parameters are usually hardcoded. In case, if you want to run MR job with different number of reducer than what mentioned in the code, you would have to modify your code, rebuild jar file and redeploy your application.

This can be avoided by implementing the Tool interface in your MapReduce driver code.

Hadoop Configuration

By implementing the Tool interface and extending Configured class, you can easily set your hadoop Configuration object via the GenericOptionsParser, thus through the command line interface. This makes your code definitely more portable (and additionally slightly cleaner) as you do not need to hardcode any specific configuration anymore.

Let’s take a couple of example with and without the use of Tool interface.

Without Tool interface

```
public class ToolMapReduce {

	public static void main(String[] args) throws Exception {

		// Create configuration
		Configuration conf = new Configuration();

		// Create job
		Job job = new Job(conf, "Tool Job");
		job.setJarByClass(ToolMapReduce.class);

		// Setup MapReduce job
		job.setMapperClass(Mapper.class);
		job.setReducerClass(Reducer.class);

		// Set No of Reduce Task to One
		job.setNumReduceTasks(1);

		// Specify Output key / value
		job.setOutputKeyClass(LongWritable.class);
		job.setOutputValueClass(Text.class);

		// Input
		FileInputFormat.addInputPath(job, new Path(args[0]));
		job.setInputFormatClass(TextInputFormat.class);

		// Output
		FileOutputFormat.setOutputPath(job, new Path(args[1]));
		job.setOutputFormatClass(TextOutputFormat.class);

		// Execute job
		int code = job.waitForCompletion(true) ? 0 : 1;
		System.exit(code);
	}
}
```

With Tool interface – Dynamic Properties

```
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class ToolMapReduce extends Configured implements Tool {
	public static void main(String[] args) throws Exception {
		int res = ToolRunner.run(new Configuration(), new ToolMapReduce(), args);
		System.exit(res);
	}

	@Override
	public int run(String[] args) throws Exception {

		// When implementing tool
		Configuration conf = this.getConf();

		// Create job
		Job job = new Job(conf, "Tool Job");
		job.setJarByClass(ToolMapReduce.class);

		// Setup MapReduce job
		// Do not specify the number of Reducer
		job.setMapperClass(Mapper.class);
		job.setReducerClass(Reducer.class);

		// Specify key / value
		job.setOutputKeyClass(LongWritable.class);
		job.setOutputValueClass(Text.class);

		// Input
		FileInputFormat.addInputPath(job, new Path(args[0]));
		job.setInputFormatClass(TextInputFormat.class);

		// Output
		FileOutputFormat.setOutputPath(job, new Path(args[1]));
		job.setOutputFormatClass(TextOutputFormat.class);

		// Execute job and return status
		return job.waitForCompletion(true) ? 0 : 1;
	}
}
```

ToolsRunner execute your MapReduce job through its static run method. In this example we do not need to hardcode the number of reducers anymore as it can be specified directly from the CLI (using the “-D” option).

```
hadoop jar /path_to_your_jar_file/ur_jar.jar
com.wordpress.hadoopi.ToolMapReduce -D
mapred.reduce.tasks=1 /HDFS_input_path /HDFS_outputpath
```

This -D option can be used for any “official” or custom property values.

```
conf.set("my.dummy.configuration","foobar");
```

becomes now…

```
-D my.dummy.configuration=foobar
```