Implementing the Tool interface for MapReduce driver

Most of people usually create their MapReduce job using a driver code that is executed though its static main method. The downside of such implementation is that most of your specific configuration (if any) is usually hardcoded. Should you need to modify some of your configuration properties on the fly (such as changing the number of reducers), you would have to modify your code, rebuild your 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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | 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 only 1 reduce task          job.setNumReduceTasks(1);            // 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          int code = job.waitForCompletion(true) ? 0 : 1;          System.exit(code);      }  } |

Your MapReduce job will be executed as follows. You expect only 2 arguments here, inputPath and outputPath, located at respectively index [0] and [1] on your main method String array.

|  |  |
| --- | --- |
| 1 | hadoop jar /path/to/My/jar.jar com.wordpress.hadoopi.ToolMapReduce /input/path /output/path |

In that case, the number of reducers (1) is hardcoded (line #17) and therefore cannot be modified on demand.

With Tool interface

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | 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).

|  |  |
| --- | --- |
| 1 | hadoop jar /path/to/My/jar.jar com.wordpress.hadoopi.ToolMapReduce -D mapred.reduce.tasks=1 /input/path /output/path |

Note that you still have to supply inputPath and outputPath arguments. Basically GenericOptionParser will separate the generic Tools options from the actual job’s arguments. Whatever the number of generic options you might supply, inputPath and outputPath variables will be still located at index [0] and [1], but in your **run method** String array (not in your main method).

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

|  |  |
| --- | --- |
| 1 | conf.set("my.dummy.configuration","foobar"); |

becomes now…

|  |  |
| --- | --- |
| 1 | -D my.dummy.configuration=foobar |

HDFS and JobTracker properties

When I need to submit a jar file remotely to a distant hadoop server, I need to specify the below properties in my driver code.

|  |  |
| --- | --- |
| 1  2  3 | Configuration conf = new Configuration();  conf.set("mapred.job.tracker", "myserver.com:8021");  conf.set("fs.default.name", "<hdfs://myserver.com:8020>"); |

Using Tool interface, this is now out of the box as you can supply both -fs and -jt options from the CLI.

|  |  |
| --- | --- |
| 1 | hadoop jar myjar.jar com.wordpress.hadoopi.ToolMapReduce -fs hdfs://myserver.com:8020 -jt myserver.com:8021 |

Thanks to this Tool implementation, my jar file is now 100% portable, and can be executed both locally or remotely without having to hardcode any specific value.

Generic options supported

Some additional useful options can be supplied from CLI.

-conf specify an application configuration file

-D use value for given property

-fs specify a namenode

-jt specify a job tracker

-files specify comma separated files to be copied to the map reduce cluster

-libjars specify comma separated jar files to include in the classpath.

-archives specify comma separated archives to be unarchived on the compute machines.

Should you need to add a specific library, archive file, etc… this Tool interface might be quite useful.  
As you can see, it is maybe worth to implement this Tool interface in your driver code as it brings added value without any additional complexity.