Developing a MapReduce Application

Learning Objectives

- The Configuration File
- Development Environment Setup
- Write a Unit Test
- Run a Job

Hadoop Configuration File – 1

Example 5-1. A simple configuration file, configuration-1.xml

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>color</name>
    <value>yellow</value>
    <description>Color</description>
  </property>

  <property>
    <name>size</name>
    <value>10</value>
    <description>Size</description>
  </property>

  <property>
    <name>weight</name>
    <value>heavy</value>
    <final>true</final>
    <description>Weight</description>
  </property>

  <property>
    <name>size-weight</name>
    <value>${size},${weight}</value>
    <description>Size and weight</description>
  </property>
</configuration>
```

Hadoop Configuration File – 2

Assuming this configuration file is in a file called *configuration-1.xml*, we can access its properties using a piece of code like this:

```
Configuration conf = new Configuration();
conf.addResource("configuration-1.xml");
assertThat(conf.get("color"), is("yellow"));
assertThat(conf.getInt("size", 0), is(10));
assertThat(conf.get("breadth", "wide"), is("wide"));
```

Hadoop Configuration File – 3

Example 5-2. A second configuration file, configuration-2.xml

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>size</name>
    <value>12</value>
  </property>

  <property>
    <name>weight</name>
    <value>light</value>
  </property>
</configuration>
```

Resources are added to a `Configuration` in order:

```
Configuration conf = new Configuration();
conf.addResource("configuration-1.xml");
conf.addResource("configuration-2.xml");
```

Hadoop Configuration File – 4

Properties defined in resources that are added later override the earlier definitions. So the `size` property takes its value from the second configuration file, *configuration-2.xml*:

```
assertThat(conf.getInt("size", 0), is(12));
```

However, properties that are marked as `final` cannot be overridden in later definitions. The `weight` property is `final` in the first configuration file, so the attempt to override it in the second fails, and it takes the value from the first:

```
assertThat(conf.get("weight"), is("heavy"));
```

Variable Expansion

- Properties can be defined in terms of other properties, or system properties.

```
assertThat(conf.get("size-weight"), is("12,heavy"));
```

System properties take priority over properties defined in resource files:

```
System.setProperty("size", "14");
assertThat(conf.get("size-weight"), is("14,heavy"));
```

Configuring the Development Environment

- Assume the *conf* directory contains three configuration files: hadoop-local.xml, hadoop-localhost.xml, and hadoop-cluster.xml.

	Local	Pseudo-distributed	Distributed
fs.default.name	file:///	hdfs://localhost/	hdfs://namenode/
mapred.job.tracker	local	localhost:8021	jobtracker:8021

- To specify which configuration file you are using

```
% hadoop fs -conf conf/hadoop-localhost.xml -ls .
```

GenericOptionsParser

GenericOptionsParser

- Is a class that interprets common Hadoop command-line options and sets them on a Configuration object for your application to use as desired.
- Is usually not used directly.
- Is indirectly used through the Tool interface, which uses GenericOptionsParser internally.

A Tool Implementation

Example 5-4. An example Tool implementation for printing the properties in a Configuration

```
public class ConfigurationPrinter extends Configured implements Tool {

  static {
    Configuration.addDefaultResource("hdfs-default.xml");
    Configuration.addDefaultResource("hdfs-site.xml");
    Configuration.addDefaultResource("mapred-default.xml");
    Configuration.addDefaultResource("mapred-site.xml");
  }

  @Override
  public int run(String[] args) throws Exception {
    Configuration conf = getConf();
    for (Entry<String, String> entry: conf) {
      System.out.printf("%s=%s\n", entry.getKey(), entry.getValue());
    }
    return 0;
  }

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new ConfigurationPrinter(), args);
    System.exit(exitCode);
  }
}
```

Show the Properties

and to set them on the `Configuration` instance. We can see the effect of picking up the properties specified in *conf/hadoop-localhost.xml* by running the following command:

```
% mvn compile
% export HADOOP_CLASSPATH=target/classes/
% hadoop ConfigurationPrinter -conf conf/hadoop-localhost.xml \
  | grep mapred.job.tracker=
mapred.job.tracker=localhost:8021
```

`GenericOptionsParser` also allows you to set individual properties. For example:

```
% hadoop ConfigurationPrinter -D color=yellow | grep color
color=yellow
```

Test For Mapper

Example 5-5. Unit test for MaxTemperatureMapper

```
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.junit.*;

public class MaxTemperatureMapperTest {

  @Test
  public void processesValidRecord() throws IOException, InterruptedException {
    Text value = new Text("0043011990999991950051518004+68750+023550FM-12+0382" +
                                  // Year ^^^^
        "99999V0203201N00261220001CN9999999N9-00111+99999999999");
                                // Temperature ^^^^^
    new MapDriver<LongWritable, Text, Text, IntWritable>()
      .withMapper(new MaxTemperatureMapper())
      .withInputValue(value)
      .withOutput(new Text("1950"), new IntWritable(-11))
      .runTest();
  }
}
```

The First Version of a Mapper

Example 5-6. First version of a Mapper that passes MaxTemperatureMapperTest

```
public class MaxTemperatureMapper
  extends Mapper<LongWritable, Text, Text, IntWritable> {

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature = Integer.parseInt(line.substring(87, 92));
    context.write(new Text(year), new IntWritable(airTemperature));
  }
}
```

Test Missing Value

the line and writes them to the Context. Let's add a test for missing values, which in the raw data are represented by a temperature of +9999:

```
@Test
public void ignoresMissingTemperatureRecord() throws IOException,
    InterruptedException {
  Text value = new Text("0043011990999991950051518004+68750+023550FM-12+0382" +
                                  // Year ^^^^
      "99999V0203201N00261220001CN9999999N9+99991+99999999999");
                                // Temperature ^^^^^
  new MapDriver<LongWritable, Text, Text, IntWritable>()
    .withMapper(new MaxTemperatureMapper())
    .withInputValue(value)
    .runTest();
}
```

A Fix to Handling Missing Value

The existing test fails with a `NumberFormatException`, as `parseInt()` cannot parse integers with a leading plus sign, so we fix up the implementation (version 2) to handle missing values:

```
@Override
public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {

  String line = value.toString();
  String year = line.substring(15, 19);
  String temp = line.substring(87, 92);
  if (!missing(temp)) {
      int airTemperature = Integer.parseInt(temp);
      context.write(new Text(year), new IntWritable(airTemperature));
  }
}

private boolean missing(String temp) {
  return temp.equals("+9999");
}
```

Test For Reducer

The reducer has to find the maximum value for a given key. Here's a simple test for this feature, which uses a ReduceDriver:

```
@Test
public void returnsMaximumIntegerInValues() throws IOException,
    InterruptedException {
  new ReduceDriver<Text, IntWritable, Text, IntWritable>()
    .withReducer(new MaxTemperatureReducer())
    .withInputKey(new Text("1950"))
    .withInputValues(Arrays.asList(new IntWritable(10), new IntWritable(5)))
    .withOutput(new Text("1950"), new IntWritable(10))
    .runTest();
}
```

Reducer Passes the Test

Example 5-7. Reducer for the maximum temperature example

```
public class MaxTemperatureReducer
  extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values,
      Context context)
      throws IOException, InterruptedException {

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

A Driver to Run MapReduce – p.157

Example 5-8. Application to find the maximum temperature

```
public class MaxTemperatureDriver extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.printf("Usage: %s [generic options] <input> <output>\n",
          getClass().getSimpleName());
      ToolRunner.printGenericCommandUsage(System.err);
      return -1;
    }

    Job job = new Job(getConf(), "Max temperature");
    job.setJarByClass(getClass());

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setCombinerClass(MaxTemperatureReducer.class);
    job.setReducerClass(MaxTemperatureReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    return job.waitForCompletion(true) ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new MaxTemperatureDriver(), args);
    System.exit(exitCode);
  }
}
```

Fixing the Mapper – 1

Example 5-9. A class for parsing weather records in NCDC format

```
public class NcdcRecordParser {

  private static final int MISSING_TEMPERATURE = 9999;

  private String year;
  private int airTemperature;
  private String quality;

  public void parse(String record) {
    year = record.substring(15, 19);
    String airTemperatureString;
    // Remove leading plus sign as parseInt doesn't like them
    if (record.charAt(87) == '+') {
      airTemperatureString = record.substring(88, 92);
    } else {
      airTemperatureString = record.substring(87, 92);
    }
    airTemperature = Integer.parseInt(airTemperatureString);
    quality = record.substring(92, 93);
  }

  public void parse(Text record) {
    parse(record.toString());
  }

  public boolean isValidTemperature() {
    return airTemperature != MISSING_TEMPERATURE && quality.matches("[01459]");
  }

  public String getYear() {
    return year;
  }

  public int getAirTemperature() {
    return airTemperature;
  }
}
```

Fixing the Mapper – 2

Example 5-10. A Mapper that uses a utility class to parse records

```
public class MaxTemperatureMapper
  extends Mapper<LongWritable, Text, Text, IntWritable> {

  private NcdcRecordParser parser = new NcdcRecordParser();

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    parser.parse(value);
    if (parser.isValidTemperature()) {
      context.write(new Text(parser.getYear()),
          new IntWritable(parser.getAirTemperature()));
    }
  }
}
```

Testing the Driver

Example 5-11. A test for MaxTemperatureDriver that uses a local, in-process job runner

```
@Test
public void test() throws Exception {
  Configuration conf = new Configuration();
  conf.set("fs.default.name", "file:///");
  conf.set("mapred.job.tracker", "local");

  Path input = new Path("input/ncdc/micro");
  Path output = new Path("output");
  FileSystem fs = FileSystem.getLocal(conf);
  fs.delete(output, true); // delete old output

  MaxTemperatureDriver driver = new MaxTemperatureDriver();
  driver.setConf(conf);

  int exitCode = driver.run(new String[] {
      input.toString(), output.toString() });
  assertThat(exitCode, is(0));

  checkOutput(conf, output);
}
```

Packaging and Launching a Job

- Packaging – package a job's classes into a *JAR* file.

Creating a job JAR file is conveniently achieved using a build tool such as Ant or Maven. The following Maven command, for example, will create a JAR file called *hadoop-examples.jar* in the project directory containing all of the compiled classes:

```
% mvn package -DskipTests
```

- This class will also use *Eclipse* to create a *JAR* file
- Launching – run the driver, specifying the driver class, input and output folders.

```
% unset HADOOP_CLASSPATH
% hadoop jar hadoop-examples.jar v3.MaxTemperatureDriver \
  -conf conf/hadoop-cluster.xml input/ncdc/all max-temp
```

View Job Information

- Hadoop comes with a web UI for viewing job information.
- It is useful for finding job's progress, statistics, and logs.
- The UI is at *http://jobtracker-host:50030*
- It has a jobtracker/home page and a job page.

The JobTracker/Home Page

Quick Links

ip-10-250-110-47 Hadoop Map/Reduce Administration

State: RUNNING
Started: Sat Apr 11 08:11:53 EDT 2009
Version: 0.20.0, r763504
Compiled: Thu Apr 9 05:18:40 UTC 2009 by ndaley
Identifier: 200904110811

Cluster Summary (Heap Size is 53.75 MB/888.94 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
53	30	2	11	88	88	16.00	0

Scheduling Information

Queue Name	Scheduling Information
default	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_200904110811_0002	NORMAL	root	Max temperature	47.52%	101	48	15.25%	30	0	NA

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_200904110811_0001	NORMAL	gonzo	word count	100.00%	14	14	100.00%	30	30	NA

Failed Jobs

none

Local Logs

Log directory, Job Tracker History

Hadoop, 2009.

The Job Page

Hadoop job_200904110811_0002 on ip-10-250-110-47

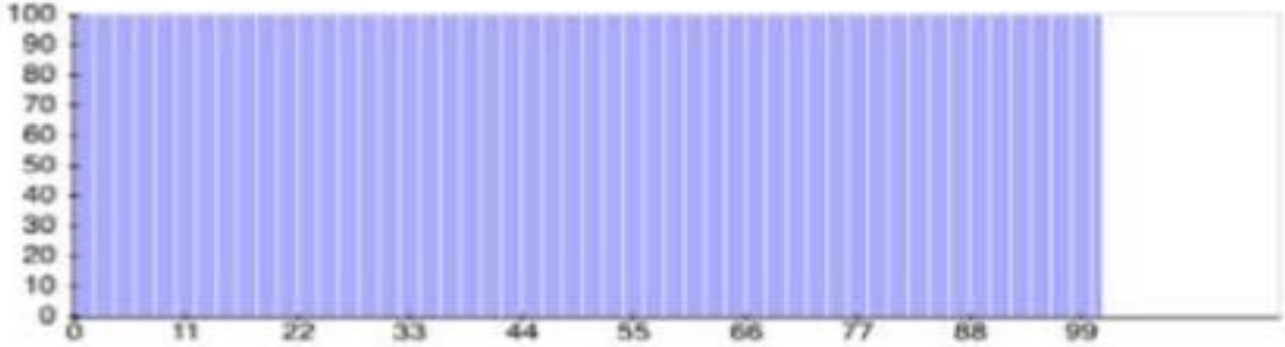
User: root
Job Name: Max temperature
Job File: hdfs://ip-10-250-110-47.ec2.internal/mnt/hadoop/mapred/system/job_200904110811_0002/job.xml
Job Setup: Successful
Status: Running
Started at: Sat Apr 11 08:15:53 EDT 2009
Running for: 5mins, 38sec
Job Cleanup: Pending

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	101	0	0	101	0	0 / 26
reduce	70.74%	30	0	13	17	0	0 / 0

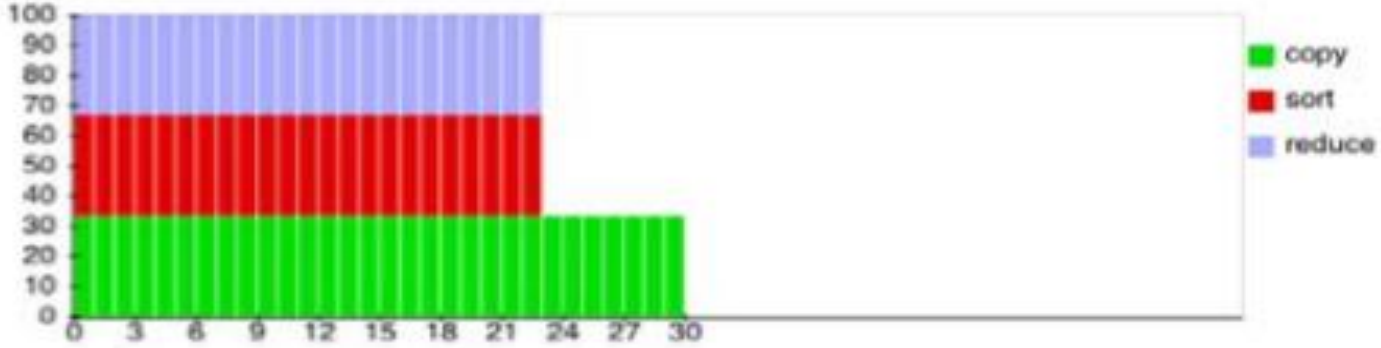
	Counter	Map	Reduce	Total
Job Counters	Launched reduce tasks	0	0	32
	Rack-local map tasks	0	0	82
	Launched map tasks	0	0	127
	Data-local map tasks	0	0	45
FileSystemCounters	FILE_BYTES_READ	12,665,901	564	12,666,465
	HDFS_BYTES_READ	33,485,841,275	0	33,485,841,275
	FILE_BYTES_WRITTEN	988,084	564	988,648
	HDFS_BYTES_WRITTEN	0	360	360
Map-Reduce Framework	Reduce input groups	0	40	40
	Combine output records	4,489	0	4,489
	Map input records	1,209,901,509	0	1,209,901,509
	Reduce shuffle bytes	0	18,397	18,397
	Reduce output records	0	40	40
	Spilled Records	9,378	42	9,420
	Map output bytes	10,282,306,995	0	10,282,306,995
	Map input bytes	274,600,205,558	0	274,600,205,558
	Map output records	1,142,478,555	0	1,142,478,555
	Combine input records	1,142,482,941	0	1,142,482,941
	Reduce input records	0	42	42

Completion Graph

Map Completion Graph - close



Reduce Completion Graph - close



Go back to JobTracker

Hadoop, 2009.

Retrieve Results

- The **–getmerge** option gets all the files in the directory and merges them into a single file on the local filesystem.

```
% hadoop fs -getmerge max-temp max-temp-local
% sort max-temp-local | tail
1991        607
1992        605
1993        567
1994        568
1995        567
1996        561
1997        565
1998        568
1999        568
2000        558
```

- The –cat option prints the output files to the console.

```
% hadoop fs -cat max-temp/*
```

Debugging a Job – p.171

want to find out what the source data causing the anomalous output looks like:

```
public class MaxTemperatureMapper
  extends Mapper<LongWritable, Text, Text, IntWritable> {

  enum Temperature {
    OVER_100
  }

  private NcdcRecordParser parser = new NcdcRecordParser();

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    parser.parse(value);
    if (parser.isValidTemperature()) {
      int airTemperature = parser.getAirTemperature();
      if (airTemperature > 1000) {
        System.err.println("Temperature over 100 degrees for input: " + value);
        context.setStatus("Detected possibly corrupt record: see logs.");
        context.getCounter(Temperature.OVER_100).increment(1);
      }
      context.write(new Text(parser.getYear()), new IntWritable(airTemperature));
    }
  }
}
```

Debugging a Job – 2

Hadoop map task list for job_200904110811_0003 on ip-10-250-110-47

Completed Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_200904110811_0003_m_000043	100.00%	hdfs://ip-10-250-110-47.ec2.internal/user/root/input/ncdc/all/1949.gz:0+220338475	11-Apr-2009 09:00:06	11-Apr-2009 09:01:25 (1mins, 18sec)		10
task_200904110811_0003_m_000044	100.00%	Detected possibly corrupt record: see logs.	11-Apr-2009 09:00:06	11-Apr-2009 09:01:28 (1mins, 21sec)		11
task_200904110811_0003_m_000045	100.00%	hdfs://ip-10-250-110-47.ec2.internal/user/root/input/ncdc/all/1970.gz:0+208374610	11-Apr-2009 09:00:06	11-Apr-2009 09:01:28 (1mins, 21sec)		10

Debugging a Job – 3

By following one of the links to the logfiles for the successful task attempt (you can see the last 4 KB or 8 KB of each logfile, or the entire file), we can find the suspect input record that we logged (the line is wrapped and truncated to fit on the page):

```
Temperature over 100 degrees for input:
0335999999433181957042302005+37950+139117SAO  +0004RJSN V020113590031500703569999994
33201957010100005+35317+139650SAO +000899999V02002359002650076249N004000599+0067...
```

This record seems to be in a different format from the others. For one thing, there are spaces in the line, which are not described in the specification.

Debugging a Job – 4

When the job has finished, we can look at the value of the counter we defined to see how many records over 100°C there are in the whole dataset. Counters are accessible via the web UI or the command line:

```
% hadoop job -counter job_200904110811_0003 'v4.MaxTemperatureMapper$Temperature' \
  OVER_100
3
```

The `-counter` option takes the job ID, counter group name (which is the fully qualified classname here), and the counter name (the enum name). There are only three malformed records in the entire dataset of over a billion records. Throwing out bad records

Hadoop Logs

Table 5-2. Types of Hadoop logs

Logs	Primary audience	Description	Further information
System daemon logs	Administrators	Each Hadoop daemon produces a logfile (using log4j) and another file that combines standard out and error. Written in the directory defined by the HADOOP_LOG_DIR environment variable.	"System logfiles" on page 309 and "Logging" on page 352
HDFS audit logs	Administrators	A log of all HDFS requests, turned off by default. Written to the namenode's log, although this is configurable.	"Audit Logging" on page 346
MapReduce job history logs	Users	A log of the events (such as task completion) that occur in the course of running a job. Saved centrally on the jobtracker and in the job's output directory in a *_logs/history* subdirectory.	"Job History" on page 167
MapReduce task logs	Users	Each tasktracker child process produces a logfile using log4j (called *syslog*), a file for data sent to standard out (*stdout*), and a file for standard error (*stderr*). Written in the *userlogs* subdirectory of the directory defined by the HADOOP_LOG_DIR environment variable.	This section