MRUnit Tutorial

Setup development environment

1. Download the latest version of MRUnit jar from Apache website: https://repository.apache.org/content/repositories/releases/org/apache/mrunit/mrunit/. For example if you are using the Hadoop version 1.0.3, download mrunit-x.x.x-incubating-hadoop2.jar.
2. Include the jar in your IDE classpath. Also download the latest verison of mokito (http://code.google.com/p/mockito/) and JUnit jar and add them to your class path of development environment.

Using from Maven add dependency like.

<dependency>
<groupId>org.apache.mrunit</groupId>
<artifactId>mrunit</artifactId>
<version>0.9.0-incubating</version>
<classifier>hadoop1</classifier>
</dependency>

Use Classifier as hadoop2 if you are using Hadoop 2 version

Writing MRUnit test cases

MRUnit testing framework is based on JUnit and it can test Map Reduce programs written on 0.20 , 0.23.x , 1.0.x , 2.x version of Hadoop.

Following is an example to use MRUnit to unit test a Map Reduce program that does SMS CDR (call details record) analysis.

The records look like

1. CDRID;CDRType;Phone1;Phone2;SMS Status Code
   655209;1;796764372490213;804422938115889;6
   353415;0;356857119806206;287572231184798;4
   835699;1;252280313968413;889717902341635;0

The MapReduce program analyzes these records, finds all records with CDRType as 1, and note its corresponding SMS Status Code. For example, the Mapper outputs are

```
6, 1
0, 1
```

The Reducer takes these as inputs and output number of times a particular status code has been obtained in the CDR records.

The corresponding Mapper and Reducer are

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

  private Text status = new Text();
  private final static IntWritable addOne = new IntWritable(1);

  /**
   * Returns the SMS status code and its count
   */
  protected void map(LongWritable key, Text value, Context context)
      throws java.io.IOException, InterruptedException {

    //655209;1;796764372490213;804422938115889;6 is the Sample record format
    String[] line = value.toString().split(";");
    // If record is of SMS CDR
    if (Integer.parseInt(line[1]) == 1) {
      status.set(line[4]);
      context.write(status, addOne);
    }
  }
}
```

The corresponding Reducer code is

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

  protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws java.io.IOException,
InterruptedException {
    int sum = 0;
    for (IntWritable value : values) {
      sum += value.get();
    }
    context.write(key, new IntWritable(sum));
  }
}
```

The MRUnit test class for the Mapper, Reduce and full MapReduce is

```
import java.util.ArrayList;
import java.util.List;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.apache.hadoop.mrunit.mapreduce.MapReduceDriver;
import org.apache.hadoop.mrunit.mapreduce.ReduceDriver;
import org.junit.Before;
import org.junit.Test;

public class SMSCDRMapperReducerTest {
  MapDriver<LongWritable, Text, Text, IntWritable> mapDriver;
  ReduceDriver<Text, IntWritable, Text, IntWritable> reduceDriver;
  MapReduceDriver<LongWritable, Text, Text, IntWritable, Text, IntWritable> mapReduceDriver;

  @Before
  public void setUp() {
    SMSCDRMapper mapper = new SMSCDRMapper();
    SMSCDRReducer reducer = new SMSCDRReducer();
    mapDriver = MapDriver.newMapDriver(mapper);
    reduceDriver = ReduceDriver.newReduceDriver(reducer);
    mapReduceDriver = MapReduceDriver.newMapReduceDriver(mapper, reducer);
  }

  @Test
  public void testMapper() {
    mapDriver.withInput(new LongWritable(), new Text(
        "655209;1;796764372490213;804422938115889;6"));
    mapDriver.withOutput(new Text("6"), new IntWritable(1));
    mapDriver.runTest();
  }

  @Test
  public void testReducer() {
    List<IntWritable> values = new ArrayList<IntWritable>();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    reduceDriver.withInput(new Text("6"), values);
    reduceDriver.withOutput(new Text("6"), new IntWritable(2));
    reduceDriver.runTest();
  }

  @Test
  public void testMapReduce() {
    mapReduceDriver.withInput(new LongWritable(), new Text(
              "655209;1;796764372490213;804422938115889;6"));
    List<IntWritable> values = new ArrayList<IntWritable>();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));
    mapReduceDriver.withOutput(new Text("6"), new IntWritable(2));
    mapReduceDriver.runTest();
  }
}
```

Run the test class as JUnit class and it will pass or fail the test depending upon if the mapper is correctly written or not.

Testing Counters

One common use of self-created Counter is to track malformed records in the input.

For example, when the input CDR record is not SMS type, the Mapper can ignore that record and increase the counter.

The revised Mapper with Counter is shown below.

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

  private Text status = new Text();
  private final static IntWritable addOne = new IntWritable(1);

  static enum CDRCounter {
    NonSMSCDR;
  };

  /**
   * Returns the SMS status code and its count
   */
  protected void map(LongWritable key, Text value, Context context) throws java.io.IOException,
InterruptedException {

    String[] line = value.toString().split(";");
    // If record is of SMS CDR
    if (Integer.parseInt(line[1]) == 1) {
      status.set(line[4]);
      context.write(status, addOne);
    } else {// CDR record is not of type SMS so increment the counter
      context.getCounter(CDRCounter.NonSMSCDR).increment(1);
    }
  }
}
```

The revised testMapper() method:

```
public void testMapper() {
    mapDriver.withInput(new LongWritable(), new Text(
        "655209;0;796764372490213;804422938115889;6"));
    //mapDriver.withOutput(new Text("6"), new IntWritable(1));
    mapDriver.runTest();
      assertEquals("Expected 1 counter increment", 1, mapDriver.getCounters()
              .findCounter(CDRCounter.NonSMSCDR).getValue());
  }
```

When the CDR record is of non SMS type out counter should be incremented by Mapper class , we are checking this by assertion that it is really incremented by one.

Similarly you can test counters for Reducer and its Counter.

Passing arguments for Testing

In next part we would see how to pass arguments to test class using the Configuration class

Configuration parameters are fetched using

Configuration.get() methods in Mapper and Reducer classes

Declare new Configruation object for your test class

```
Configuration conf = new Configuration();
```

In setUp() method add following

```
mapDriver.setConfiguration(conf);
conf.set("myParameter1", "20");
conf.set("myParameter2", "23");
```

Your test class would pass on these parameters to the mappers.

Happy Testing