



Denodo MapReduce Custom Wrapper

Revision 20131001

NOTE

This document is confidential and proprietary of **Denodo Technologies**.
No part of this document may be reproduced in any form by any means without prior written authorization of **Denodo Technologies**.

Copyright © 2013
Denodo Technologies Proprietary and Confidential

CONTENTS

| | |
|--|-----------|
| 1 INTRODUCTION..... | 4 |
| 2 MAPREDUCE..... | 5 |
| 3 MAPREDUCE CUSTOM WRAPPER..... | 7 |
| 3.1 USING MAPREDUCE CUSTOM WRAPPER..... | 7 |
| 3.2 WHY USING SSH TO EXECUTE MAPREDUCE JOBS?..... | 10 |
| 3.3 WHEN NOT TO USE MAPREDUCE CUSTOM WRAPPER..... | 10 |
| 4 DEVELOPING A MAPREDUCE JOB..... | 11 |
| 5 EXECUTING MAPREDUCE CUSTOM WRAPPER..... | 13 |
| 6 EXTENDING MAPREDUCE CUSTOM WRAPPER..... | 16 |
| 7 TROUBLESHOOTING..... | 18 |



1 INTRODUCTION

mapreduce-customwrapper is a Virtual DataPort custom wrapper for running **map and reduce operations** using **Hadoop**.

The custom wrapper connects to the Hadoop machine **via SSH**, executes a MapReduce job and reads the results from **HDFS**.

2 MAPREDUCE

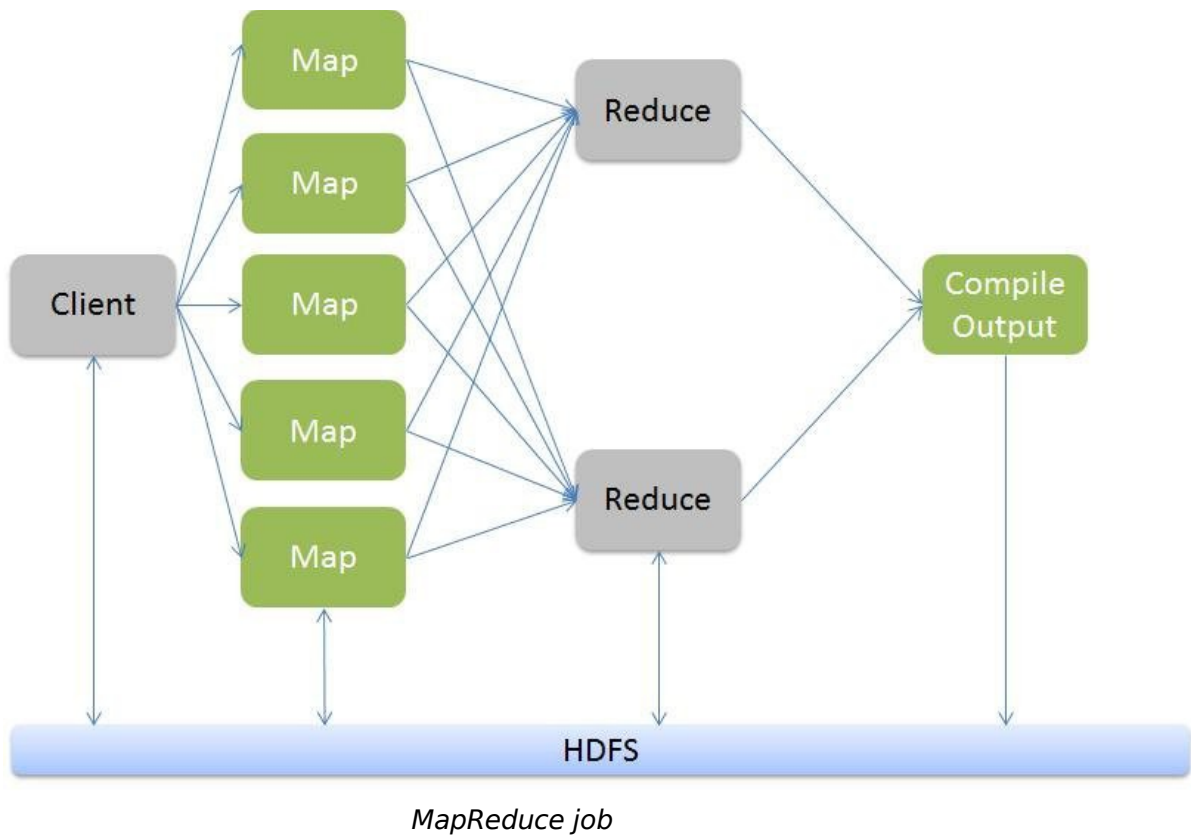
MapReduce is a programming model which uses parallelism to handle **large data sets** in **distributed systems**. This model provides fault tolerance by re-scheduling failed tasks and also considers I/O locality to reduce the network traffic. The process consists of:

1. The master node splits the input file into smaller files and forwards each one to slave nodes.
2. Each slave node operates the **map** function and produces intermediate key-value pairs.
3. The **reduce** function accepts an intermediate key and a set of values for this key and merges these values to get the result.

The model is inspired by the map and reduce functions commonly used in functional programming. Furthermore, the key contribution of the MapReduce framework are not the map and reduce functions, but:

- A simple abstraction for processing any size of data that fits into this programming model. Simple since it hides the details of parallelization, fault-tolerance, locality optimization and load balancing.
- Fault tolerance. The master node monitors workers and restarts chunks of work when necessary.
- Data locality, reducing network overhead.

MapReduce is supported by multiple implementations, being **Apache Hadoop** one of the most popular.



3 MAPREDUCE CUSTOM WRAPPER

MapReduce custom wrapper connects to the machine where Hadoop is running via SSH and executes a MapReduce job running the following command from the shell:

```
hadoop jar <jar> main-class job-parameters
```

When the job finishes the wrapper reads the output stored in an HDFS directory.

3.1 USING MAPREDUCE CUSTOM WRAPPER

Base views created from the MapReduceSSHWrapper need the following **mandatory** parameters:

1. Host: Name of the Hadoop machine or its IP address.
2. Port: Port number to connect to the Hadoop machine, default is 22.
3. User: Username to log into the Hadoop machine.
4. Path to jar in host: Path to the job jar on the Hadoop machine.
5. Main class in jar: Name of the main class. This value is mandatory to avoid having to create runnable jars; jars can contain several main classes, therefore several MapReduce jobs.
6. MapReduceJobHandler implementation: Class implementing the MapReduceJobHandler interface. You can use one of the distributed with the custom wrapper:
 1. SequenceFileOutputMapReduceJobHandler, for jobs whose output is written to a **Sequence file**.
 2. MapFileOutputMapReduceJobHandler, for jobs whose output is written to a **Map file**.
 3. DelimitedFileOutputMapReduceJobHandler, for jobs whose output is written to a **delimited file**.
 4. AvroFileOutputMapReduceJobHandler, for jobs whose output is written to an **Avro file**.

or create your own implementation and link it to the data source of the custom wrapper (see [Extending MapReduce custom wrapper](#)).

Five **optional** parameters:

1. Connection timeout: Amount of time, in milliseconds, that the wrapper will wait for the “hadoop jar...” command to complete. Configure a value of 0 or leave the box blank to wait indefinitely.

2. MapReduce job parameters: Parameters required by the MapReduce job. They will be added to the command “hadoop jar...”. MapReduceJobHandler implementations could add some parameters too.
3. Key class: Key class name implementing org.apache.hadoop.io.Writable interface. This parameter is **mandatory** when using SequenceFileOutputMapReduceJobHandler or MapFileOutputMapReduceJobHandler.
4. Value class: Value class name implementing org.apache.hadoop.io.Writable interface. This parameter is **mandatory** when using SequenceFileOutputMapReduceJobHandler or MapFileOutputMapReduceJobHandler.
5. Separator: Delimiter between key/value pairs. This parameter is **mandatory** when using DelimitedFileOutputMapReduceJobHandler.


There are also parameters that are **mutually exclusive**:

1. Password: Password associated with the username.
2. Key file: Key file created with PuTTY key file generator.
 - Passphrase: Passphrase associated with the key file.

and

1. Avro schema path: **HDFS path** to the Avro schema file.
2. Avro schema JSON: JSON of the Avro schema.

Only one of these two latter parameters is **mandatory** when using AvroFileOutputMapReduceJobHandler.

| | | |
|----------------------------------|---|--|
| Host IP | <input type="text" value="192.168.153.128"/> | |
| Host port | <input type="text" value="22"/> | |
| User | <input type="text" value="root"/> | |
| Password | <input type="password" value="•••••"/> | |
| Key file | <input type="text" value="None"/>  | <input type="button" value="Configure"/> |
| Passphrase | <input type="text"/> | |
| Host timeout | <input type="text" value="30000000"/> | |
| Path to jar in host | <input type="text" value="/hadoop/jars/hadooptestwordcount-1.0-SNAPSHOT"/> | |
| Main class in jar | <input type="text" value="test.MapReduceDriver1"/> | |
| MapReduce job parameters | <input type="text" value="2.168.153.128,8021,/user/sandbox/input/index.html"/> | |
| MapReduceJobHandler implement... | <input type="text" value="andler.SequenceFileOutputMapReduceJobHandler"/> | |
| Key class | <input type="text" value="org.apache.hadoop.io.Text"/> | |
| Value class | <input type="text" value="org.apache.hadoop.io.IntWritable"/> | |
| Separator | <input type="text"/> | |
| Avro schema path | <input type="text"/> | |
| Avro schema JSON | <input type="text"/> | |

MapReduceSSHWrapper base view edition

The image above shows the creation of a base view for the MapReduce job *WordCount*. Note that the jar location in the machine is not accessed through HDFS. The job parameters are: the NameNode IP and port, the JobTracker IP and port and the HDFS path to the input file.

3.1.1 Supported values for Key class and Value class

Classes provided for the base view parameters Key class and Value class should implement `org.apache.hadoop.io.Writable` interface. Value class also supports arrays, i.e. `org.apache.hadoop.io.IntWritable[]` represents an `ArrayWritable` of `IntWritable`.

The Writable classes supported by the custom wrapper are:

- `org.apache.hadoop.io.NullWritable`
- `org.apache.hadoop.io.Text`
- `org.apache.hadoop.io.IntWritable`

- `org.apache.hadoop.io.LongWritable`
- `org.apache.hadoop.io.BooleanWritable`
- `org.apache.hadoop.io.DoubleWritable`
- `org.apache.hadoop.io.FloatWritable`
- `org.apache.hadoop.io.ByteWritable`
- An array of any of the previous classes (written as `org.apache.hadoop.io.IntWritable[], ...`).

3.2 WHY USING SSH TO EXECUTE MAPREDUCE JOBS?

The custom wrapper executes a MapReduce job by connecting to the Hadoop machine via SSH and running the command “`hadoop jar...`”. This may seem a complicate way to execute jobs because Hadoop API allows to submit and run jobs programmatically.

When using Hadoop API:

1. Job jars have to be added to the Denodo Platform and linked to the data source created for the custom wrapper.
2. Base views have to be configured with the job name.

While when using SSH:

1. Job jars have to be stored the in the Hadoop machine.
2. Base views have to be configured with the path to the job jar in the Hadoop machine.

Consequently, SSH is the preferred way as it avoids Denodo Platform having dependencies with the MapReduce jobs.

3.3 WHEN NOT TO USE MAPREDUCE CUSTOM WRAPPER

MapReduce custom wrapper is synchronous: it invokes a MapReduce job, wait until the job has finished and then reads the output from HDFS. For relatively fast MapReduce jobs synchronous execution would be desirable. However, in situations where a MapReduce job is expected to take a large amount of time asynchronous execution should be preferred.

Denodo Connect provides two components for achieve asynchronous execution of MapReduce jobs:

- SSH custom wrapper: executes a MapReduce job connecting to the Hadoop machine via SSH.
- HDFS custom wrapper: reads the output files of the job stored in HDFS.

4 DEVELOPING A MAPREDUCE JOB

There are some of the issues to consider when developing a MapReduce job for this custom wrapper:

1. A MapReduce job should be packaged in a jar containing at least one main class. The jar does not need to be executable as the name of the main class is provided as a parameter to the custom wrapper; this way the jar can contain more than one main class and therefore more than one job.
2. The job jar should be located in an accessible path in the machine running Hadoop. The jar is not stored in HDFS but in the machine file system.
3. The job receives the parameters in the following order: first those indicated in the property MapReduce job parameters, then those returned by getJobParameters method in MapReduceJobHandler.
4. The following property should be added to the job configuration:
 1. `conf.set("mapreduce.fileoutputcommitter.marksuccessfuljobs", "false");`
to disable the creation of the success file in the output folder, as the custom wrapper only expects the output files in that folder.

Below there is the skeleton of the main class of a MapReduce job.

```
public class MapReduceDriverExample {

    public static void main(String[] args) {

        String namenodeIp = args[0];
        String namenodePort = args[1];
        String jobtrackerIp = args[2];
        String jobtrackerPort = args[3];
        String input = args[4];
        String output = args[5];
        String customParam1 = args[6];

        JobConf conf = new JobConf(test.MapReduceDriverExample.class);
        conf.setJobName("example-" + System.nanoTime());
        conf.set("defaultFS", "hdfs://" + namenodeIp + ":" +
namenodePort);
        conf.set("mapreduce.jobtracker.address", jobtrackerIp + ":"
+ jobtrackerPort);

        //Remove SUCCESS file from output directory
        conf.set("mapreduce.fileoutputcommitter.marksuccessfuljobs",
"false");
    }
}
```

```
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);

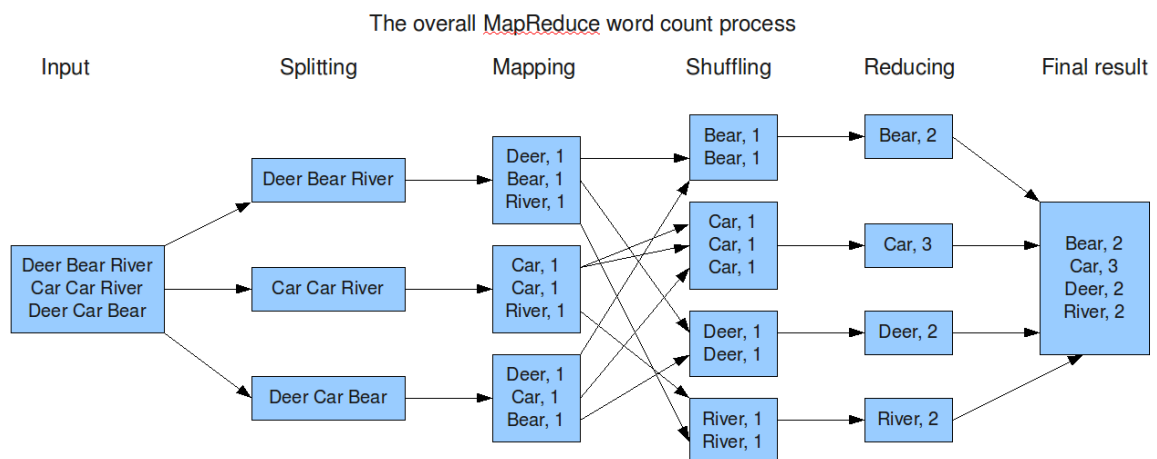
// Set your Mapper and Reducer classes
conf.setMapperClass(test.Mapper1.class);
conf.setReducerClass(test.Reducer1.class);
conf.setOutputFormat(SequenceFileOutputFormat.class);

// Input and output are in hdfs
FileInputFormat.setInputPaths(conf, new Path(input));
FileOutputFormat.setOutputPath(conf, new Path(output));
try {
    RunningJob rj = JobClient.runJob(conf);
} catch (Exception e) {
    // Place here your code to handle the exception
}
}
```

MapReduce job main class skeleton

5 EXECUTING MAPREDUCE CUSTOM WRAPPER

WordCount is a MapReduce job that counts the number of times each word appears in the input files.





MapReduce job for counting words

For more information see <http://wiki.apache.org/hadoop/WordCount>.

The steps for executing the *WordCount* job with the MapReduce custom wrapper are:

1. Add the custom wrapper jar to the VDP installation.
2. Create the custom data source.
3. Implement the MapReduce job that counts the times every word appears. For the main class follow the skeleton displayed in the [Developing a MapReduce job](#) section.
4. Copy the jar to the Hadoop machine. In the example `/hadoop/jars/hadooptestwordcount-1.0-SNAPSHOT.jar`.
5. Create the base view:

| | |
|----------------------------------|---|
| Host IP | <input type="text" value="192.168.153.128"/> |
| Host port | <input type="text" value="22"/> |
| User | <input type="text" value="root"/> |
| Password | <input type="password" value="•••••"/> |
| Key file | <input type="text" value="None"/>  |
| Passphrase | <input type="text"/> |
| Host timeout | <input type="text" value="30000000"/> |
| Path to jar in host | <input type="text" value="/hadoop/jars/hadooptestwordcount-1.0-SNAPSHOT"/> |
| Main class in jar | <input type="text" value="test.MapReduceDriver1"/> |
| MapReduce job parameters | <input type="text" value="2.168.153.128,8021,/user/sandbox/input/index.html"/> |
| MapReduceJobHandler implement... | <input type="text" value="andler.SequenceFileOutputMapReduceJobHandler"/> |
| Key class | <input type="text" value="org.apache.hadoop.io.Text"/> |
| Value class | <input type="text" value="org.apache.hadoop.io.IntWritable"/> |
| Separator | <input type="text"/> |
| Avro schema path | <input type="text"/> |
| Avro schema JSON | <input type="text"/> |



MapReduceSSHWrapper base view edition

6. Copy the input file from the local disk to HDFS. In the example /user/sandbox/input/index.html.
7. Execute the base view and see the results:

| Execute view haddopcw_view | |
|--------------------------------------|-------|
| Total rows received: 131 (shown 131) | |
| KEY ▼ | VALUE |
| You | 4 |
| you | 2 |
| writing, | 2 |
| work | 2 |
| WITHOUT | 2 |
| with | 4 |
| WARRANTIES | 2 |
| version="1.0"?> | 2 |
| Version | 2 |
| value | 1 |
| user | 3 |
| use | 2 |
| Unless | 2 |
| under | 8 |
| type="text/xml" | 2 |
| to | 7 |
| tmp | 1 |
| this | 6 |
| The | 2 |

WordCount job base view result

6 EXTENDING MAPREDUCE CUSTOM WRAPPER

The custom wrapper provides an API to customize the parameters sent to the command “hadoop jar...” and how data is being read from the job’s output. The interface that defines this behaviour is `MapReduceJobHandler` and contains the following methods:

1. `getJobParameters`, returns the parameters that will be passed to the command “hadoop jar <jar> main-class **job-parameters**”, besides those indicated in the base view parameter `MapReduce job parameters`.
2. `getSchema`, describes the structure of the data of the output of the job.
3. `getOutputReader`, returns the class implementing `MapReduceJobOutputReader` interface that is in charge of retrieving data from the job’s output.

The interface `MapReduceJobOutputReader` contains the methods:

- `read`, reads the next value from the output.
- `close`, closes the reader and releases any resources associated with it.

MapReduce custom wrapper includes some implementations for the previous interfaces that may fit your needs:

1. `DelimitedFileOutputMapReduceJobHandler`, for jobs whose output is written to a **delimited file**.
2. `SequenceFileOutputMapReduceJobHandler`, for jobs whose output is written to a **Sequence file**.
3. `MapFileOutputMapReduceJobHandler`, for jobs whose output is written to a **Map file**.
4. `AvroFileOutputMapReduceJobHandler`, for jobs whose output is written to an **Avro file**.

There are some points you have to keep in mind in case you use these classes:

1. The output folder is named as “/denodo_output_X”, where x is a random value.
2. The job output folder is deleted after the custom wrapper execution.
3. `MapReduce job parameters` should contain the NameNode IP and port as the first and second values.
4. The call to run the MapReduce job is:

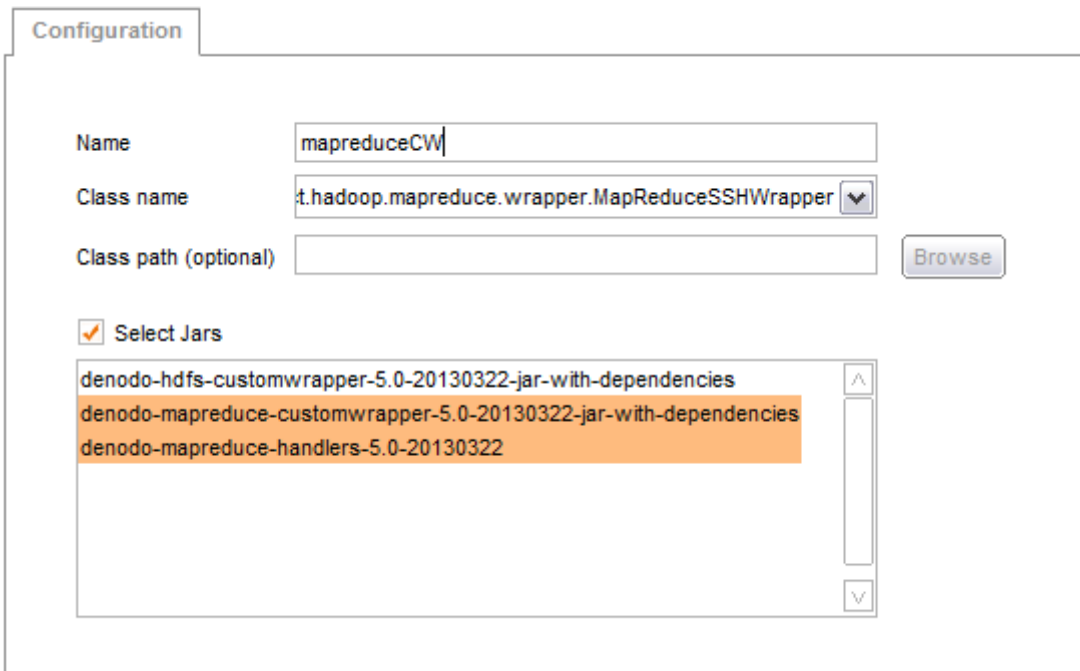
```
hadoop jar <jar> main-class job-parameters output-dir
```

where `job-parameters` is a comma-separated list, i.e. `192.168.153.128,8020,192.168.153.128,8021,/user/sandbox/input/index.html` and `output-dir` is the job output path, i.e. `/denodo_output_3482473427`.

For the wrapper to be able to use these custom classes they should be added to Denodo Platform.

To summarize, these are the recommended steps for the development of your own classes:

1. Implement MapReduceJobHandler class and/or MapReduceJobOutputReader.
2. Package those classes and their dependencies in a jar.
3. Add the jar to Denodo Platform using File>Extensions>Jar management menu.
4. Modify the custom data source to add the new jar.
5. Configure the base view with the new implementation of MapReduceJobHandler.
6. Test the base view.



MapReduceSSHWrapper data source edition

7 TROUBLESHOOTING

Symptom

Error message: "Server IPC version 7 cannot communicate with client version 4".

Resolution

It is a version mismatch problem. IPC version 4 is for Hadoop 1.0, which is the version distributed in the custom wrapper artifact `denodo-mapreduce-customwrapper-5.0-20130902-jar-with-dependencies`, whereas version 7 is for Hadoop 2.0.

To solve the problem you should use the custom wrapper artifact `denodo-mapreduce-customwrapper-5.0-xxx` and copy the dependencies of Hadoop v2 to the `$DENODO_PLATFORM_HOME/extensions/thirdparty/lib` directory.

Symptom

Error message: "denodo_output_XXX/_logs is not a file".

Resolution

Job logs refers to the events and configuration for a completed job. They are retained to provide interesting information for the user running a job. These job history files are stored on the local filesystem of the jobtracker in a history subdirectory of the logs directory.

A second copy is also stored for the user in the `_logs/history` subdirectory of the job's output directory. This is the directory the custom wrapper is complaining about because it expects only output files.

By setting to none the property `hadoop.job.history.user.location` in the `mapred-site.xml` config file or in the mapreduce job configuration no user job history is saved in the job's output directory.