Distributed Computing with GridGain and UNICORE

Bastian Baranski, baranski@52north.org

January 8, 2010

Contents

1 Introduction

This document describes how to configure the 52° North Web Processing Service (WPS)[1] so that processes could be performed distributed and potentially parallel in a distributed Grid and Cloud Computing environment either via the GridGain[2] or the UNICORE[3] framework.

If you are a beginner in Grid and Cloud Computing and if you are not familiar with the UNICORE framework, it is strongly recommended to start with the GridGain framework.

If you have any further questions or helpful suggestions, please contact the author of this document or write a mail to the 52° North Geoprocessing mailing list at

geoprocessingservices@52north.org

2 Prerequisites

Checkout and compile the current WPS sources from

`http://52north.org/svn/geoprocessing/main/WPS/trunk/WPS/`

[1]`http://www.52north.org/wps`
[2]`http://www.gridgain.com`
[3]`http://www.unicore.eu`

For more information about building, configuring and creating your own WPS processes please follow the general WPS installation tutorial at

`http://52north.org/download/Geoprocessing/documents/Website/WPS_Tutorial.pdf`

In generally, you have to be familiar to some degree with the Java programming language and some corresponding development tools like Subversion (Version Controlling), Maven (Project Management and Build Automation) and Eclipse (Software Development Environment).

3 GridGain

This section describes how to execute your WPS processes on a local cluster by utilizing the GridGain framwork. GridGain is an open source and Java-based MapReduce implementation.

3.1 Installation

Firstly, you have to setup a local GridGain infrastructure and prepare each node in that infrastructure for executing the WPS processes.

Therefore, you have to donwload the GridGain binaries from the project homepage at

`http://www.gridgain.com/`

Follow the installation instructions at

`http://www.gridgainsystems.com/wiki/display/GG15UG/Installation+Instructions`

or the fancy and helpful screencasts at

`http://gridgain.com/screencasts.html`

to setup your local cluster of GridGain nodes.

In most cases the setup process of the GridGain infrastructure is no big deal and typically there is no complex configuration required. For testing purposes one local running instance of GridGain is fairly enough, you don't have to setup necessarily a complete cluster of GridGain nodes.

After setting up the local GridGain infrastructure you have to prepare each node in the infrastructure for executing the WPS processes. Therefore you have to copy all libraries from the directory

`$WPS_WEBAPP/WEB-INF/lib/` [4]

to the directory

`$GRIDGAIN_HOME/lib/ext/`

on **each** of the GridGain nodes.

[4]The `$WPS_WEBAPP` string either points to the webapp directory of your local WPS installation or to the automatically generated target directory of the `52n-wps-webapp` module in the WPS sources. That depends on wether you use the pre-compiled binaries of the WPS or if you build the WPS binaries directly from the source code.

This process must be repeated each time you have modified the original source code of the WPS. If you have many nodes in your local GridGain infrastrcuture, that process could be very time consuming and should be automated somehow (e.g. with a shell script).

3.2 Development

Secondly, you have to develop an algorithm that could be distributed at a GridGain infrastructure.

The typical development process of such a distributable algorithm is exemplified on basis of the `GridGainSimpleBufferAlgorithm` algorithm that is delivered by default with the WPS source code and that encapsulates the functionality of the classical `SimpleBuffer` algorithm.

The source code of the `GridGainSimpleBufferAlgorithm` algorithm is located at

`52n-wps-gridgain/src/main/java/org/n52/wps/gridgain/algorithm/GridGainSimpleBufferAlgorithm.java`

The WPS process description of the `GridGainSimpleBufferAlgorithm` algorithm is located at

`52n-wps-gridgain/src/main/resources/org/n52/wps/gridgain/algorithm/GridGainSimpleBufferAlgorithm.`

The source code of the encapsulated `SimpleBufferAlgorithm` algorithm is located at

`52n-wps-server/src/main/java/org/n52/wps/server/algorithm/SimpleBufferAlgorithm.java`

```java
public class GridGainSimpleBufferAlgorithm extends AbstractGridGainAlgorithm
{
  public GridGainSimpleBufferAlgorithm()
  {
    super(new org.n52.wps.server.algorithm.SimpleBufferAlgorithm());
  }

  public List<Map<String, List<IData>>> split(Map<String, List<IData>> inputData)
  {
    (...)
  }

  public Map<String, IData> merge(List<Map<String, IData>> outputData);
  {
    (...)
  }
```

Listing 1: A brief overview of the `GridGainSimpleBufferAlgorithm` algorithm.

To develop an algorithm that could be distributed at a GridGain infrastructure you have to proceed the following steps.

1. Each distributable algorithm is based on a classical algorithm implementation either of the `IAlgorithm` interface or the abstract `AbstractObservaleAlgorithm`. Implement such a classical algorithm as it is described in detail in the general WPS installation tutorial (e.g. the `SimpleBufferAlgorithm` algorithm).

2. Create a new class (e.g. the `GridGainSimpleBufferAlgorithm` class) that extends the abstract `AbstractGridGainAlgorithm` class.

3. In the constructor of the new class (e.g. the `GridGainSimpleBufferAlgorithm` class) you have to call the constructor of the superclass with an instance of the previously developed classical algorithm (e.g. the `SimpleBufferAlgorithm` class) as an argument.

4. Implement the missing `split(...)` function. Following the MapReduce[5][6] and Divide and Conquer[7] approach, this function splits the (potentially huge) input data set into smaller chunks, so that several (potentially smaller) parts of the algorithm could be performed parallel on different nodes in the Grid. If the algorithm cannot or should not be scheduled in parallel, just return a list with one element (even exactly the normal input data).

5. Implement the missing `merge(...)` function. If smaller chunks of the algorithm are executed on more then one node in parallel, the resulting data sets have to be merged. If the algorithm cannot or should not be scheduled in parallel, just return the first element of the argument (even exactly the one and only resulting data).

6. Create a WPS process description file for the new distributable algorithm. This process descriptions is most likely very similar to the WPS process description file of the classical and encapsulated algorithm implementation (just the `Identifier` element must match the package and classname of the new class).

However, the new algorithm encapsulates the original functionality of the embedded classical algorithm (provided by its `run(...)` method) and could be now distributed at a GridGain infrastructure. Furthermore, the algorithm could be executed on more than one node in parallel to increase the computational performance (dependes on the amount of returned elements of the new `split(...)` method).

3.3 Configuration

Thirdly, you have to enable the WPS to load your developed process during startup.

Therefore, you have to modify the WPS configuration file located at

`$WPS_WEBAPP/WEB-INF/conf/wps_config.xml`

By default there is a disabled algorithm repository entry in the WPS configuration file for the the previously introduced `GridGainSimpleBufferAlgorithm` algorithm. To enable the GridGain algorithm repository just uncomment the default entry.

```
<Repository name="LocalGridGainAlgorithmRepository" className="org.n52.wps.gridgain
    .GridGainAlgorithmRepository">
  <Property name="Algorithm">org.n52.wps.gridgain.algorithm.
      GridGainSimpleBufferAlgorithm</Property>
</Repository>
```

Listing 2: The default GridGain algorithm repository entry in the WPS configuration file.

You can load more than one process during startup just by inserting additional `Algorithm` property elements pointing to different process implementations.

3.4 Testing

Finally, you have to check if your process implementation and configuration was successful.

One possibility to check if the process implementation and configuration was successful is to execute the developed process directly from within a Desktop GIS client (e.g. via the uDig 52° North WPS Plugin as described in the general WPS installation tutorial).

[5] http://en.wikipedia.org/wiki/MapReduce
[6] http://labs.google.com/papers/mapreduce.html
[7] http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm

Another solution is to execute the `ExecuteTest` class that is delivered by default with the WPS source code and that submits an example WPS Execute request document (for the the previously introduced `GridGainSimpleBufferAlgorithm` algorithm) to a defined URL.

The source code of the `ExecuteTest` class is located at

`52n-wps-gridgain/src/test/java/org/n52/wps/gridgain/ExecuteTest.java`

The example WPS Execute request document for the exemplified `GridGainSimpleBufferAlgorithm` is located at

`52n-wps-gridgain/src/main/resources/org/n52/wps/gridgain/ExecuteDocument.xml`

Please note, that you have to adjust the URL in the `ExecuteTest` class to your local environment. Furthermore, you have to replace the WFS reference URL in the example WPS Execute request document with an existing WFS. For testing purposes, the author of this document propose to install locally a simple Geoserver[8] solution that provides the data that is used in the example WPS Execute request document.

4 UNICORE

This section describes how to execute your WPS processes on a local cluster by utilizing the open source and Java-based UNICORE middleware. UNICORE offers a ready-to-run Grid system including client and server software.

4.1 Installation

Firstly, you have to setup a local UNICORE infrastructure by installing the server components and the GPE Application Client.

Therefore you have to donwload the UNICORE binaries from the project homepage at

`http://www.unicore.eu/`

Follow the installation instructions at

`http://www.unicore.eu/XXX`

and the general tutorials at

`http://www.unicore.eu/documentation/`

to setup your local cluster of UNCIORE nodes by installing the server components and the GPE Application Client.

4.2 Development

Secondly, you have to develop an algorithm that could be distributed at a UNICORE infrastructure.

The typical development process of such a distributable algorithm is exemplified on basis of the `UnicoreSimpleBufferAlgorithm` algorithm that is delivered by default with the WPS source code

[8]`http://geoserver.org`

and that encapsulates the functionality of the classical `SimpleBuffer` algorithm.

The source code of the `UnicoreSimpleBufferAlgorithm` algorithm is located at

`52n-wps-unicore/src/main/java/org/n52/wps/unicore/algorithm/UnicoreSimpleBufferAlgorithm.java`

The WPS process description of the `UnicoreSimpleBufferAlgorithm` algorithm is located at

`52n-wps-unicore/src/main/resources/org/n52/wps/unicore/algorithm/UnicoreSimpleBufferAlgorithm.xml`

The source code of the encapsulated `SimpleBufferAlgorithm` algorithm is located at

`52n-wps-server/src/main/java/org/n52/wps/server/algorithm/SimpleBufferAlgorithm.java`

To develop an algorithm that could be distributed at a UNICORE infrastructure you have to proceed the same devlopment steps as describes in detail in the development section of the GridGain framework (Section 2.2). But you have to create a new class that extends the abstract `AbstractUnicoreAlgorithm` class instead of the abstract `AbstractGridGainAlgorithm` class.

```
public class UnicoreSimpleBufferAlgorithm extends AbstractUnicoreAlgorithm
{
  public UnicoreSimpleBufferAlgorithm()
  {
    super(new org.n52.wps.server.algorithm.SimpleBufferAlgorithm());
  }

  public List<Map<String, List<IData>>> split(Map<String, List<IData>> inputData)
  {
    (...)
  }

  public Map<String, IData> merge(List<Map<String, IData>> outputData);
  {
    (...)
  }
```

Listing 3: A brief overview of the UnicoreSimpleBufferAlgorithm algorithm.

However, the development process for embedding existing classical algorithms into distributable algorithms and for splitting and merging the input and output data are completely equal to the GridGain framework. That's why there is actually no additional time and effort required to switch between the GridGain and UNICORE framework.

4.3 Configuration

Thirdly, you have to enable the WPS to load your developed process during startup.

Therefore, you have to modify the WPS configuration file at

`$WPS_WEBAPP/WEB-INF/conf/wps_config.xml`

By default there is a disabled algorithm repository entry in the WPS configuration file for the the previously introduced UnicoreSimpleBufferAlgorithm algorithm. To enable the UNICORE algorithm repository just uncomment the default entry.

```
<Repository name="LocalUnicoreAlgorithmRepository" className="org.n52.wps.unicore.
    UnicoreAlgorithmRepository">
```

```
  <Property name="Algorithm">org.n52.wps.unicore.algorithm.
      UnicoreSimpleBufferAlgorithm</Property>

  <Property name="Registry">https://localhost:8080/DEMO-SITE/services/Registry?res=
      default_registry</Property>
  <Property name="Keystore">/home/username/.gpe4unicore/keystore.jks</Property>
  <Property name="Registry">https://localhost:8080/DEMO-SITE/services/Registry?res=
      default_registry</Property>
  <Property name="Keystore">/home/username/.gpe4unicore/keystore.jks</Property>
  <Property name="Type">jks</Property>
  <Property name="Alias">demo user</Property>
  <Property name="Password">unicore</Property>

  <Property name="OverwriteRemoteFile">false</Property>
  <Property name="CompressInputData">true</Property>
</Repository>
```

Listing 4: The default UNICORE algorithm repository entry in the WPS configuration file.

You can load more than one process during startup just by inserting additional `Algorithm` property elements pointing to different process implementations.

Furthermore, there is a specific set of properties that must be specified for a successful connection to a UNICORE installation. The following properties must be defined.

- The WPS needs additional information about where to find the UNICORE server installation. The `Registry` property points to a UNCIORE gateway.
- The WPS needs security information to access a UNCIORE server installation. Typically a keystore will hold all the certificates you need to access the grid. During the installation process of the GPE Application Client such a keystore - including all personal and CA certificates - will be created. The `Keystore` property points to that keystore.
- The `Type` property specifies the type of the used keystore. During the installation process of the GPE Application Client the keystore is created automatically and so you typically don't have to change the value ("jks" for a Java Keystore) for that property.
- The `Alias` property specifies the alias of your personal certificate. During the installation process of the GPE Application Client such a certificate is created automatically and so you typically don't have to change the value ("demo user") for that property.
- The `Password` property specifies the password for accessing the keystore. During the installation process of the GPE Application Client you have to specify the password and so you typically don't have to change the value for that property ("unicore" is proposed as the default password during the GPE Application Client installation process).
- Before submitting a job to the Grid, the latest WPS libraries are copied to the home directory of the Grid user. The `OverwriteRemoteFile` property defines, if existing remote files will be overridden. Normally the value of that property should be 'false'. But if you made any modifications to the WPS source code, you have to set that value to 'true', so that once the modified libraries will be updated at the users home directory.
- The `CompressInputData` property defines, if the input data will be compressed before the submission to each Grid node. Normally the value for that property should be 'true', but sometimes (e.g. for testing purposes) it could be helpful to set the value for that property to 'false'.

The property settings in the default UNICORE algorithm repository entry match the resulting environment after installing the standard UNICORE server and client components following the tutorials from the project homepage.

4.4 Testing

Finally, you have to check if your process implementation and configuration was successful.

You have the same possibilities to check if the process implementation and configuration was successful as described in the testing chapter for the GridGain framework.
The source code of the `ExecuteTest` class is located at

```
52n-wps-unicore/src/test/java/org/n52/wps/unicore/ExecuteTest.java
```

The example WPS Execute request document for the exemplified `UnicoreSimpleBufferAlgorithm` is located at

```
52n-wps-unicore/src/main/resources/org/n52/wps/unicore/ExecuteDocument.xml
```

Please note, that you again have to adjust your settings in the `ExecuteTest` class and the example WPS Execute request document to your local environment.