How-to Implement Algorithms for DataMiner

Gianpaolo Coro, Giancarlo Panichi
Istituto di Scienza e Tecnologie dell'Informazione A. Faedo
{coro,panichi}@isti.cnr.it

In this document we describe how to implement an algorithm that should run on the D4Science DataMiner[1]. DataMiner is a cross-usage service that provides users and services with tools for performing data mining operations. Specifically, it offers a unique access to perform data mining and statistical operations on heterogeneous data, which may reside either at client side, in the form of comma-separated values files, or be remotely hosted, possibly in a database. The DataMiner service is able to take inputs and execute the operation requested by a client or a user, by invoking the most suited computational facility from a set of available computational resources. Executions can run either on multi-core machines or on different computational platforms, such as D4Science and other different private and commercial Cloud providers.

Prerequisites

IDE: Eclipse Java EE IDE for Web Developers. Version: 3.7+

We advice to also follow this demonstration video:

http://i-marine.eu/Content/eTraining.aspx?id=e1777006-a08c-49ad-b2e6-c13e094f27d4

Step by Step

Let's start by creating a project using the eclipse IDE that is mavenized according to the D4Science e-Infrastructure indications[2]. After having mavenized the project in eclipse a user has to add dependencies.

[1] https://wiki.gcube-system.org/gcube/DataMiner_Manager

[2] http://gcube.wiki.gcube-system.org/gcube/index.php/Creating_gCube_Maven_components:_How-Toindications

Maven coordinates

The maven artifact coordinates are:

```
<dependency>
        <groupId>org.gcube.dataanalysis</groupId>
        <artifactId>ecological-engine</artifactId>
        <version>(1.6.1-SNAPSHOT,2.0.0-SNAPSHOT)</version>
</dependency>
```

Let's start creating a new call which implements a basic algorithm; it will be executed by Dataminer. The next step is to extend a basic interface: "StandardLocalExternalAlgorithm". The following snippet shows unimplemented interface methods that we are going to fulfill.

```
public class SimpleAlgorithm extends StandardLocalExternalAlgorithm
    {

        @Override
        public void init() throws Exception {
                // TODO Auto-generated method stub
        }
        @Override
        public String getDescription() {
                // TODO Auto-generated method stub
                return null;
        }
        @Override
        protected void process() throws Exception {
                // TODO Auto-generated method stub

        }
        @Override
        protected void setInputParameters() {
                // TODO Auto-generated method stub

        }
        @Override
        public void shutdown() {
                // TODO Auto-generated method stub
        }
        @Override
        public StatisticalType getOutput() {
                return null;
        }
}
```

init() is the initialization method. In this simple example we need to initialize the logging facility and we use the logger from the ecological engine library. In case the algorithm uses a database, we have to open its connection in this method.

shutdown() closes the database connection. In the getDescription() method we add a simple description for the algorithm.

Customize input visualization

String input parameters

The user's input is obtained in the setInputParameters() method by calling the method addStringInput with the following parameters: name of the variable; description for the variable; default value.

User input is retrieved using getInputParameter() passing the name used as parameter into setInputParameters().

```
protected void setInputParameters() {
                addStringInput(NameOfVariable, "Description", "
                    DefaultInput");
```

The input parameter will be automatically passed by DataMiner to the procedure. In particular, to process the method we can retrieve this parameter by name, i.e. the one we set in the addStringInput method.

```
@Override
protected void process() throws Exception {
....
String userInputValue = getInputParameter(NameOfVariable);
}
```

Combo box input parameter

In order to obtain a combo box we have to define a enumerator that contains the possible choices that could be selected in the combo box and you have to pass it to the method "addEnumerateInput" as follows:

```
public enum Enum {
FIRST_ENUM,
SECOND_ENUM
}

protected void setInputParameters() {
addEnumerateInput(Enum.values(), variableName, "Description",
                                                Enum.FIRST_ENUM.name());
}
```

addEnumerateInput parameters are respectively: values of the declared enumerator; the name of variable used to extract value insert by user; the description; the default value visualized in the comboBox.

Importing input from the Statistical Manager database

Users can upload data in the DataMiner "Data Space" panel. After the uploading of a file (for example a CSV file), it's possible to use the uploaded data as input for

an algorithm. In order to select the columns values of a table that is extrapolated from a CSV file, an algorithm developer builds the methods in the following way:

```
@Override
protected void setInputParameters() {
List<TableTemplates> templates = new ArrayList<TableTemplates>();
templates.add(TableTemplates.GENERIC);
InputTable tinput = new InputTable(templates, "Table","Table␣
    Description");
ColumnTypesList columns = new ColumnTypesList("Table","Columns", "
    Selceted␣Columns␣Description", false);
inputs.add(tinput);
inputs.add(columns);
DatabaseType.addDefaultDBPars(inputs);

}

@Override
protected void process() throws Exception {
{
config.setParam("DatabaseDriver", "org.postgresql.Driver");
SessionFactory dbconnection = DatabaseUtils.initDBSession(config);
String[] columnlist = columnnames.split(AlgorithmConfiguration.
    getListSeparator());
List<Object> speciesList = DatabaseFactory.executeSQLQuery("select␣
    " + columnlist[0]+ "␣from␣" + tablename, dbconnection);
}
```

Algorithms using databases

In order to use a database it is required to call, into setInputParameters(), the method addRemoteDatabaseInput(). An important step is to pass as first parameter the name of the Runtime Resource addressing the database. DataMiner automatically retrieves thew following parameters from the runtime resource: url, username and password. Into the process method, before the initialisation of the database connection, url,user and password can be retrieved using the getInputParameter method. Each of these parameters is retrieved using a string and passing it into addRemoteDatabaseInput as parameters.

```
@Override
protected void setInputParameters() {
...
addRemoteDatabaseInput("Obis2Repository", urlParameterName,
    userParameterName, passwordParameterName, "driver", "dialect");

@Override
protected void process() throws Exception {
...

String databaseJdbc = getInputParameter(urlParameterName);
String databaseUser = getInputParameter(userParameterName);
String databasePwd = getInputParameter(passwordParameterName);
```

```
connection = DriverManager.getConnection(databaseJdbc, databaseUser
    ,databasePwd);
...

}
```

Customising the output

The last step is to set and to specify output of procedure. For this purpose the user overrides the getOutput() method which returns a StatisticalType. The first output parameter to be instantiated is a PrimitiveType object that wraps a string; so, the type to set is a string. A name and a description are associated to the output value. A second output can be instantiated of another PrimitiveType as well. To such scope, a map is used, which keeps the order of the parameter used to store the output.

String output

In ordert to have a string as output, a user should create a PrimitiveType as follows:

```
@Override
public StatisticalType getOutput() {
...
PrimitiveType val = new PrimitiveType(String.class.getName(),
    myString , PrimitiveTypes.STRING, stringName, defaultValue);
return val;

}
```

Bar Chart Output

In order to create an Histogram Chart, a user should fulfill a DafaultCategory-Dataser object and use it to create chart:

```
DefaultCategoryDataset dataset;
...
dataset.addValue(...);
...

@Override
public StatisticalType getOutput() {
...
HashMap<String, Image> producedImages = new HashMap<String, Image
    >();
JFreeChart chart = HistogramGraph.createStaticChart(dataset);
Image image = ImageTools.toImage(chart.createBufferedImage(680,
    420));
```

```
producedImages.put("Species Observations", image);
...
}
```

Timeseries Chart Output

In order to create a TimeSeries Chart, a user has to fulfill a DafaultCategoryDataser object and use it to create the chart. The second parameter of createStatiChart method is the format of the time-stamp:

```
DefaultCategoryDataset dataset;
...
dataset.addValue(...);
...
@Override
public StatisticalType getOutput() {
...
HashMap<String, Image> producedImages = new HashMap<String, Image
    >();
JFreeChart chart = TimeSeriesGraph.createStaticChart(dataset, "yyyy
    ");
Image image = ImageTools.toImage(chart.createBufferedImage(680,
    420));
producedImages.put("TimeSeries chart", image);
...
}
```

File Output

In order to create a file as output, that the user can download, the following code is required:

```
protected String fileName;
protected BufferedWriter out;

@Override
protected void process() throws Exception {
fileName = super.config.getPersistencePath() + "results.csv";
out = new BufferedWriter(new FileWriter(fileName));
out.write(results);
out.newLine();
}

@Override
public StatisticalType getOutput() {
...
PrimitiveType file = new PrimitiveType(File.class.getName(), new
    File(fileName), PrimitiveTypes.FILE, "Description ", "Default
    value");
map.put("Output",file);
...
}
```

Algorithm testing

In the following, a template example is reported to test an algorithm in the Eclipse IDE. The same Factory exists for Clusterers, Evaluators, Modellers and Generators.

The developer should download the following folder: http://goo.gl/aWlPV5 and put it locally to the code. For new algorithms just an edit is required, of one among the Transducers, Clusterers, Evaluators, Modellers or Generators files, adding your class.

Only the file corresponding to the category of the agorithm should be edited.

```
package org.gcube.dataanalysis.ecoengine.test.regression;

import java.util.List;

import org.gcube.dataanalysis.ecoengine.configuration.
    AlgorithmConfiguration;
import org.gcube.dataanalysis.ecoengine.evaluation.bioclimate.
    InterpolateTables.INTERPOLATIONFUNCTIONS;
import org.gcube.dataanalysis.ecoengine.interfaces.
    ComputationalAgent;
import org.gcube.dataanalysis.ecoengine.interfaces.Transducerer;
import org.gcube.dataanalysis.ecoengine.processing.factories.
    TransducerersFactory;

public class TestTransducers {

public static void main(String[] args) throws Exception {
System.out.println("TEST 1");
List<ComputationalAgent> trans = null;
trans = TransducerersFactory.getTransducerers(testConfigLocal());
trans.get(0).init();
Regressor.process(trans.get(0));
trans = null;
}

private static AlgorithmConfiguration testConfigLocal() {

 AlgorithmConfiguration config = Regressor.getConfig();
 config.setAgent("OCCURRENCES_DUPLICATES_DELETER");

 config.setParam("longitudeColumn", "decimallongitude");
 config.setParam("latitudeColumn", "decimallatitude");
 config.setParam("recordedByColumn", "recordedby");
 config.setParam("scientificNameColumn", "scientificname");
 config.setParam("eventDateColumn", "eventdate");
 config.setParam("lastModificationColumn", "modified");
 config.setParam("OccurrencePointsTableName", "
     whitesharkoccurrences2");
 config.setParam("finalTableName", "
     whitesharkoccurrencesnoduplicates");
 config.setParam("spatialTolerance", "0.5");
 config.setParam("confidence", "80");
```

```
return config;
}

}
```

Properties Files and Deployment

In order to deploy an algorithm, the developer should create:

- the jar corresponding to the eclipse Java project containing the algorithm;
- a property file containing the name you want the algorithm to be displayed on the GUI and the classpath to algorithm class. E.g. MY_ALGORITHM=org.gcube.cnr.Myalgorithm

Thereafter, these files should be provided to the D4Science infrastructure team. They will install the algorithm onto DataMiner and an interface will be automatically generated.

Complete Example with multiple outputs

In the following complete example, inside the src/main/java folder, the package org.gcube.dataanalysis.myAlgorithms exists that contains the class SimpleAlgorithm implements the SIMPLE_ALGORITHM = org.gcube.dataanalysis.myrAlgorithms.SimpleAlgorithm algorithm.

```
public class AbsoluteSpeciesBarChartsAlgorithm  extends
StandardLocalExternalAlgorithm  {
        LinkedHashMap<String, StatisticalType> map = new
            LinkedHashMap<String, StatisticalType>();
        static String databaseName = "DatabaseName";
        static String userParameterName = "DatabaseUserName";
        static String passwordParameterName = "DatabasePassword";
        static String urlParameterName = "DatabaseURL";
        private String firstSpeciesNumber="Species# :";
        private String yearStart="Starting year :";
        private String yearEnd="Ending year :";
        private int speciesNumber;
        private DefaultCategoryDataset defaultcategorydataset;
        @Override
        public void init() throws Exception {
                AnalysisLogger.getLogger().debug("Initialization");
        }

        @Override
        public String getDescription() {
                return "Algorithm returning bar chart of most
                    observed species in a specific years range (
                    with respect to the OBIS database)";
```

```
}

@Override
protected void process() throws Exception {
        defaultcategorydataset = new DefaultCategoryDataset
            ();
        String driverName = "org.postgresql.Driver";
        String tmp=getInputParameter(firstSpeciesNumber);

        speciesNumber = Integer.parseInt(tmp);
        Class driverClass = Class.forName(driverName);
        Driver driver = (Driver) driverClass.newInstance();
        String databaseJdbc = getInputParameter(
            urlParameterName);
        String year_start = getInputParameter(yearStart);
        String year_end = getInputParameter(yearEnd);

        String databaseUser = getInputParameter(
            userParameterName);
        String databasePwd = getInputParameter(
            passwordParameterName);
        Connection connection = null;
        connection = DriverManager.getConnection(
            databaseJdbc, databaseUser,
                        databasePwd);
        Statement stmt = connection.createStatement();
        String query = "SELECT  tname, sum(count)AS count
            FROM public.count_species_per_year WHERE year::
            integer >="
                        + year_start
                        + "AND year::integer <="
                        + year_end
                        + "GROUP BY tname ORDER BY count
                            desc;";
        ResultSet rs = stmt.executeQuery(query);
        int i =0;
        String s = "Species";
                while (rs.next()&& i<speciesNumber) {

                        String tname = rs.getString("tname"
                            );
                        String count = rs.getString("count"
                            );
                        int countOcc=Integer.parseInt(count
                            );

                        // First output (list of string)
                        PrimitiveType val = new
                            PrimitiveType(String.class.
                            getName(), count,
                            PrimitiveTypes.STRING, tname,
                            tname);
                        map.put(tname, val);
                        if(i<16)
                        defaultcategorydataset.addValue(
                            countOcc,s,tname);
                        else
```

```
                                    break;
                            i++;

        }
        connection.close();



}

@Override
protected void setInputParameters() {
        addStringInput(firstSpeciesNumber,
                        "Number of shown species", "10");
        addStringInput(yearStart, "Starting year of
            observations",
                        "1800");
        addStringInput(yearEnd, "Ending year of
            observations", "2020");
        addRemoteDatabaseInput("Obis2Repository",
            urlParameterName,
                        userParameterName,
                            passwordParameterName, "driver"
                            , "dialect");


}

@Override
public void shutdown() {
        AnalysisLogger.getLogger().debug("Shutdown");
}


@Override
public StatisticalType getOutput() {
        PrimitiveType p = new PrimitiveType(Map.class.
            getName(), PrimitiveType.
            stringMap2StatisticalMap(outputParameters),
            PrimitiveTypes.MAP, "Discrepancy Analysis","");
        AnalysisLogger.getLogger().debug("MapsComparator:
            Producing Gaussian Distribution for the errors"
            );
        //build image:
        HashMap<String, Image> producedImages = new HashMap
            <String, Image>();

        JFreeChart chart = HistogramGraph.createStaticChart
            (defaultcategorydataset);
     Image image = ImageTools.toImage(chart.
         createBufferedImage(680, 420));
     producedImages.put("Species Observations", image);

        PrimitiveType images = new PrimitiveType(HashMap.
            class.getName(), producedImages, PrimitiveTypes
            .IMAGES, "ErrorRepresentation", "Graphical
            representation of the error spread");
```

```
                //end build image
                AnalysisLogger.getLogger().debug("Bar Charts
                    Species Occurrences Produced");
                //collect all the outputs

                map.put("Result", p);
                map.put("Images", images);

                //generate a primitive type for the collection
                PrimitiveType output = new PrimitiveType(HashMap.
                    class.getName(), map, PrimitiveTypes.MAP, "
                    ResultsMap", "Results Map");


                return output;
        }

}
```

Integrating R Scripts

DataMiner supports R scripts integration. This section explains how to integrate R scripts that will be executed by one single powerful machine in sequential mode. The calculation will be distributed on one of the machines that make up the DataMiner system (DM), and the DataMiner will automatically account for multi-users requests management. This section does not deal with parallel processing enabled for the script, which will be discussed later.

In the Eclipse IDE project, the developer should download the following configuration folder: http://goo.gl/bNKrZK and then add the following maven dependency:

```
<dependency>
   <groupId>org.gcube.dataanalysis</groupId>
   <artifactId>ecological-engine-smart-executor</artifactId>
   <version>[1.0.0-SNAPSHOT,2.0.0)</version>
</dependency>
```

Then the developer should copy an R script inside the cfg folder. The DM framework assumes that the R file (i) accepts an input file whose name is hard-coded in the script, (ii) produces an output file whose name is hard-coded in the script, (iii) requires an R context made up of user's variables, (iv) possibly requires custom adjustment to the code.

The DM framework facilitates the call to the script by adding context variables "on the fly" and managing multi-user synchronous calls. This mechanism is performed by generating new on-the-fly temporary R scripts for each user. The DM will be also responsible for distributing the script on one powerful machine.

Required packages are assumed to be preinstalled on the backend system. One example of an algorithm calling a complex interpolation model is the following:

```
package org.gcube.dataanalysis.executor.rscripts;

import java.io.File;
import java.util.HashMap;
import java.util.LinkedHashMap;

import org.gcube.contentmanagement.lexicalmatcher.utils.
    AnalysisLogger;
import org.gcube.dataanalysis.ecoengine.datatypes.PrimitiveType;
import org.gcube.dataanalysis.ecoengine.datatypes.StatisticalType;
import org.gcube.dataanalysis.ecoengine.datatypes.enumtypes.
    PrimitiveTypes;
import org.gcube.dataanalysis.ecoengine.interfaces.
    StandardLocalExternalAlgorithm;
import org.gcube.dataanalysis.executor.util.RScriptsManager;

public class SGVMS_Interpolation extends
    StandardLocalExternalAlgorithm {

        private static int maxPoints = 10000;
        public enum methodEnum { cHs, SL};
        RScriptsManager scriptmanager;
        String outputFile;

        @Override
        public void init() throws Exception {
                AnalysisLogger.getLogger().debug("Initializing␣
                    SGVMS_Interpolation");
        }

        @Override
        public String getDescription() {
                return "An␣interpolation␣method␣relying␣on␣the␣
                    implementation␣by␣the␣Study␣Group␣on␣VMS␣(SGVMS
                    ).␣The␣method␣uses␣two␣interpolation␣approached
                    ␣to␣simulate␣vessels␣points␣at␣a␣certain␣
                    temporal␣resolution.␣The␣input␣is␣a␣file␣in␣
                    TACSAT␣format␣uploaded␣on␣the␣Statistical␣
                    Manager.␣The␣output␣is␣another␣TACSAT␣file␣
                    containing␣interpolated␣points." +
                                "The␣underlying␣R␣code␣has␣been␣
                                    extracted␣from␣the␣SGVM␣
                                    VMSTools␣framework.␣This␣
                                    algorithm␣comes␣after␣a␣
                                    feasibility␣study␣(http://goo.
                                    gl/risQre)␣which␣clarifies␣the␣
                                    features␣an␣e-Infrastructure␣
                                    adds␣to␣the␣original␣scripts.␣
                                    Limitation:␣the␣input␣will␣be␣
                                    processed␣up␣to␣"+maxPoints+"␣
                                    vessels␣trajectory␣points.";
        }

        @Override
        protected void process() throws Exception {
```

```
status = 0;
//instantiate the R Script executor
scriptmanager = new RScriptsManager();
//this is the script name
String scriptName = "interpolateTacsat.r";
//absolute path to the input, provided by the SM
String inputFile = config.getParam("InputFile");

AnalysisLogger.getLogger().debug("Starting SGVM
    Interpolation -> Config path "+config.
    getConfigPath()+" Persistence path: "+config.
    getPersistencePath());
//default input and outputs
String defaultInputFileInTheScript = "tacsat.csv";
String defaultOutputFileInTheScript = "
    tacsat_interpolated.csv";
//input parameters: represent the context of the
    script. Values will be assigned in the R
    environment.
LinkedHashMap<String,String> inputParameters = new
    LinkedHashMap<String, String>();
inputParameters.put("npoints",config.getParam("
    npoints"));
inputParameters.put("interval",config.getParam("
    interval"));
inputParameters.put("margin",config.getParam("
    margin"));
inputParameters.put("res",config.getParam("res"));
inputParameters.put("fm",config.getParam("fm"));
inputParameters.put("distscale",config.getParam("
    distscale"));
inputParameters.put("sigline",config.getParam("
    sigline"));
inputParameters.put("minspeedThr",config.getParam("
    minspeedThr"));
inputParameters.put("maxspeedThr",config.getParam("
    maxspeedThr"));
inputParameters.put("headingAdjustment",config.
    getParam("headingAdjustment"));
inputParameters.put("equalDist",config.getParam("
    equalDist").toUpperCase());
//add static context variables
inputParameters.put("st", "c(minspeedThr,
    maxspeedThr)");
inputParameters.put("fast", "TRUE");
inputParameters.put("method", "\""+config.getParam(
    "method")+"\"");

AnalysisLogger.getLogger().debug("Starting SGVM
    Interpolation -> Input Parameters: "+
    inputParameters);
//if other code injection is required, put the
    strings to substitute as keys and the
    substituting ones as values
HashMap<String,String> codeInjection = null;
//force the script to produce an output file,
```

```
            otherwise generate an exception
        boolean scriptMustReturnAFile = true;
        boolean uploadScriptOnTheInfrastructureWorkspace =
            false; //the Statistical Manager service will
            manage the upload
        AnalysisLogger.getLogger().debug("SGVM␣
            Interpolation->␣Executing␣the␣script␣");
        status = 10;
        //execute the script in multi-user mode
        scriptmanager.executeRScript(config, scriptName,
            inputFile, inputParameters,
            defaultInputFileInTheScript,
            defaultOutputFileInTheScript, codeInjection,
            scriptMustReturnAFile,
            uploadScriptOnTheInfrastructureWorkspace);
        //assign the file path to an output variable for
            the SM
        outputFile = scriptmanager.currentOutputFileName;
        AnalysisLogger.getLogger().debug("SGVM␣
            Interpolation->␣Output␣File␣is␣"+outputFile);
        status = 100;
}

@Override
protected void setInputParameters() {
        //declare the input parameters the user will set:
            they will basically correspond to the R context
        inputs.add(new PrimitiveType(File.class.getName(),
            null, PrimitiveTypes.FILE, "InputFile", "Input␣
            file␣in␣TACSAT␣format.␣E.g.␣http://goo.gl/
            i16kPw"));
        addIntegerInput("npoints", "The␣number␣of␣pings␣or␣
            positions␣required␣between␣each␣real␣or␣actual␣
            vessel␣position␣or␣ping", "10");
        addIntegerInput("interval", "Average␣time␣in␣
            minutes␣between␣two␣adjacent␣datapoints", "120"
            );
        addIntegerInput("margin", "Maximum␣deviation␣from␣
            specified␣interval␣to␣find␣adjacent␣datapoints␣
            (tolerance)", "10");
        addIntegerInput("res", "Number␣of␣points␣to␣use␣to␣
            create␣interpolation␣(including␣start␣and␣end␣
            point)", "100");
        addEnumerateInput(methodEnum.values(), "method","
            Set␣to␣cHs␣for␣cubic␣Hermite␣spline␣or␣SL␣for␣
            Straight␣Line␣interpolation", "cHs");
        addDoubleInput("fm", "The␣FM␣parameter␣in␣cubic␣
            interpolation", "0.5");
        addIntegerInput("distscale", "The␣DistScale␣
            parameter␣for␣cubic␣interpolation", "20");
        addDoubleInput("sigline", "The␣Sigline␣parameter␣in
            ␣cubic␣interpolation", "0.2");
        addDoubleInput("minspeedThr", "A␣filter␣on␣the␣
            minimum␣speed␣to␣take␣into␣account␣for␣
            interpolation", "2");
        addDoubleInput("maxspeedThr", "A␣filter␣on␣the␣
            maximum␣speed␣to␣take␣into␣account␣for␣
```

```
                interpolation", "6");
            addIntegerInput("headingAdjustment", "Parameter to
                adjust the choice of heading depending on its
                own or previous point (0 or 1). Set 1 in case
                the heading at the endpoint does not represent
                the heading of the arriving vessel to that
                point but the departing vessel.", "0");
            inputs.add(new PrimitiveType(Boolean.class.getName
                (), null, PrimitiveTypes.BOOLEAN, "equalDist",
                "Whether the number of positions returned
                should be equally spaced or not", "true"));
        }

        @Override
        public StatisticalType getOutput() {
            //return the output file by the procedure to the SM
            PrimitiveType o = new PrimitiveType(File.class.
                getName(), new File(outputFile), PrimitiveTypes
                .FILE, "OutputFile", "Output file in TACSAT
                format.");
            return o;
        }

        @Override
        public void shutdown() {
            //in the case of forced shutdown, stop the R
                process
            if (scriptmanager!=null)
                    scriptmanager.stop();
            System.gc();
        }

}
```

Testing the integrated algorithm

In order to test the above algorithm, the developer should just modify the "transducerers.properties" file inside the cfg folder by adding the following string:

SGVM_INTERPOLATION = org.gcube.dataanalysis.executor.rscripts.SGVMS_Interpolation

which will assign a name to the algorithm. Then a test class for this algorithm will be the following:

```
package org.gcube.dataanalysis.executor.tests;

import java.util.List;

import org.gcube.dataanalysis.ecoengine.configuration.
    AlgorithmConfiguration;
import org.gcube.dataanalysis.ecoengine.datatypes.PrimitiveType;
import org.gcube.dataanalysis.ecoengine.datatypes.StatisticalType;
import org.gcube.dataanalysis.ecoengine.interfaces.
    ComputationalAgent;
```

```
import org.gcube.dataanalysis.ecoengine.processing.factories.
    TransducerersFactory;
import org.gcube.dataanalysis.ecoengine.test.regression.Regressor;

public class TestSGVMInterpolation {

        public static void main(String[] args) throws Exception {
                // setup the configuration
                AlgorithmConfiguration config = new
                    AlgorithmConfiguration();
                // set the path to the cfg folder and to the
                    PARALLEL_PROCESSING folder
                config.setConfigPath("./cfg/");
                config.setPersistencePath("./PARALLEL_PROCESSING");
                //set the user's inputs. They will passed by the SM
                     to the script in the following way:
                config.setParam("InputFile", "<absolute␣path␣to␣the
                    ␣file>/tacsatmini.csv"); //put the absolute
                    path to the input file
                config.setParam("npoints", "10");
                config.setParam("interval", "120");
                config.setParam("margin", "10");
                config.setParam("res", "100");
                config.setParam("method", "SL");
                config.setParam("fm", "0.5");
                config.setParam("distscale", "20");
                config.setParam("sigline", "0.2");
                config.setParam("minspeedThr", "2");
                config.setParam("maxspeedThr", "6");
                config.setParam("headingAdjustment", "0");
                config.setParam("equalDist", "true");

                //set the scope and the user (optional for this
                    test)
                config.setGcubeScope( "/gcube/devsec/devVRE");
                config.setParam("ServiceUserName", "test.user");

                //set the name of the algorithm to call, as is is
                    in the transducerer.properties file
                config.setAgent("SGVM_INTERPOLATION");

                //recall the transducerer with the above name
                List<ComputationalAgent> transducers =
                    TransducerersFactory.getTransducerers(config);
                ComputationalAgent transducer =transducers.get(0);
                //init the transducerer
                transducer.init();
                //start the process
                Regressor.process(transducer);
                //retrieve the output
                StatisticalType st = transducer.getOutput();
                System.out.println("st:"+((PrimitiveType)st).
                    getContent());
        }

}
```

Enabling Cloud Computing for R Scripts

In the case of a process running in the Infrastructure and using Cloud computing, a developer should have to extend the ActorNode class, define how to setup the process, chunkize the input space, run the script and perform the Reduce phase. These steps are performed using the following methods respectively:

- setup(AlgorithmConfiguration config)
- getNumberOfRightElements()
- getNumberOfLeftElements()
- postProcess(boolean manageDuplicates, boolean manageFault)

```
package org.gcube.dataanalysis.executor.nodes.algorithms;

public class LWR extends ActorNode {

        public String destinationTable;
        public String destinationTableLabel;
        public String originTable;
        public String familyColumn;
        public int count;

        public float status = 0;

        //specify the kind of parallel process: the following
            performs a matrix-to-matrix comparison
        @Override
        public ALG_PROPS[] getProperties() {
                ALG_PROPS[] p = { ALG_PROPS.
                    PHENOMENON_VS_PARALLEL_PHENOMENON };
                return p;
        }

        @Override
        public String getName() {
                return "LWR";
        }

        @Override
        public String getDescription() {
                return "An algorithm to estimate Length-Weight
                    relationship parameters for marine species,
                    using Bayesian methods. Runs an R procedure.
                    Based on the Cube-law theory.";
        }

        @Override
        public List<StatisticalType> getInputParameters() {
                List<TableTemplates> templateLWRInput = new
                    ArrayList<TableTemplates>();
                templateLWRInput.add(TableTemplates.GENERIC);
                InputTable p1 = new InputTable(templateLWRInput, "
                    LWR_Input", "Input table containing taxa and
                    species information", "lwr");
```

```
        ColumnType p3 = new ColumnType("LWR_Input", "
            FamilyColumn", "The␣column␣containing␣Family␣
            information", "Family", false);
        ServiceType p4 = new ServiceType(ServiceParameters.
            RANDOMSTRING, "RealOutputTable", "name␣of␣the␣
            resulting␣table", "lwr_");
        PrimitiveType p2 = new PrimitiveType(String.class.
            getName(), null, PrimitiveTypes.STRING, "
            TableLabel", "Name␣of␣the␣table␣which␣will␣
            contain␣the␣model␣output", "lwrout");

        List<StatisticalType> parameters = new ArrayList<
            StatisticalType>();
        parameters.add(p1);
        parameters.add(p3);
        parameters.add(p2);
        parameters.add(p4);

        DatabaseType.addDefaultDBPars(parameters);

        return parameters;
}

@Override
public StatisticalType getOutput() {
        List<TableTemplates> template = new ArrayList<
            TableTemplates>();
        template.add(TableTemplates.GENERIC);
        OutputTable p = new OutputTable(template,
            destinationTableLabel, destinationTable, "
            Output␣lwr␣table");
        return p;
}

@Override
public void initSingleNode(AlgorithmConfiguration config) {

}

@Override
public float getInternalStatus() {
        return status;
}

private static String scriptName = "UpdateLWR_4.R";

//the inputs delivered by the SM are: the index and number
    of elements to take from the left and right tables, the
     indication on if the same requeste was yet asked to
    another worker node (in the case of errors), the
    sandobox folder in which the script will be executed,
    the configuration of the algorithm
@Override
public int executeNode(int leftStartIndex, int
    numberOfLeftElementsToProcess, int rightStartIndex, int
     numberOfRightElementsToProcess,  boolean duplicate,
    String sandboxFolder, String
```

```
nodeConfigurationFileObject, String
logfileNameToProduce) {
    String insertQuery = null;
    try {
            status = 0;
            //reconstruct the configuration
            AlgorithmConfiguration config =
                Transformations.restoreConfig(
                nodeConfigurationFileObject);
            config.setConfigPath(sandboxFolder);
            System.out.println("Initializing␣DB");
            //take the parameters and possibly
                initialize connection to the DB
            dbconnection = DatabaseUtils.initDBSession(
                config);
            destinationTableLabel = config.getParam("
                TableLabel");
            destinationTable = config.getParam("
                RealOutputTable");
            System.out.println("Destination␣Table:␣"+
                destinationTable);
            System.out.println("Destination␣Table␣Label
                :␣"+destinationTableLabel);
            originTable = config.getParam("LWR_Input");
            familyColumn = config.getParam("
                FamilyColumn");
            System.out.println("Origin␣Table:␣"+
                originTable);

            // take the families to process
            List<Object> families = DatabaseFactory.
                executeSQLQuery(DatabaseUtils.
                getDinstictElements(originTable,
                familyColumn, ""), dbconnection);

            // transform the families into a string
            StringBuffer familiesFilter = new
                StringBuffer();
            familiesFilter.append("Families␣<-␣Fam.All[
                ");

            int end = rightStartIndex +
                numberOfRightElementsToProcess;
            //build the substitution string
            for (int i = rightStartIndex; i < end; i++)
                 {
                    familiesFilter.append("Fam.All␣==␣
                        \"" + families.get(i) + "\"");
                    if (i < end - 1)
                            familiesFilter.append("␣|␣"
                                );
            }
            familiesFilter.append("]");

            //substitution to perform in the script
            String substitutioncommand = "sed␣-i␣'s/
                Families␣<-␣Fam.All[Fam.All==␣\"
```

```
        Acanthuridae\" | Fam.All == \"Achiridae
        \"]/" + familiesFilter + "/g' " + "
        UpdateLWR_Test2.R";
System.out.println("Preparing for
        processing the families names: "+
        familiesFilter.toString());

substituteInScript(sandboxFolder+scriptName
        ,sandboxFolder+"UpdateLWR_Tester.R","
        Families <- Fam.All[Fam.All == \"
        Acanthuridae\" | Fam.All == \"Achiridae
        \"]",familiesFilter.toString());
//for test only

System.out.println("Creating local file
        from remote table");
// download the table in csv format to feed
      the procedure
DatabaseUtils.
        createLocalFileFromRemoteTable(
        sandboxFolder+"RF_LWR.csv", originTable
        , ",", config.getDatabaseUserName(),
        config.getDatabasePassword(),config.
        getDatabaseURL());

String headers = "Subfamily,Family,Genus,
        Species,FBname,SpecCode,AutoCtr,Type,a,
        b,CoeffDetermination,Number,LengthMin,
        Score,BodyShapeI";
System.out.println("Adding headers to the
        file");

String headerscommand = "sed -i '1s/^/"+
        headers+"\\n/g' "+"RF_LWR2.csv";
// substitute the string in the RCode
addheader(sandboxFolder+"RF_LWR.csv",
        sandboxFolder+"RF_LWR2.csv",headers);
System.out.println("Headers added");
System.out.println("Executing R script " +
        "R --no-save < UpdateLWR_Tester.R");
// run the R code: it can be alternatively
        made with the methods of the previous
        example
Process process = Runtime.getRuntime().exec
        ("R --no-save");
BufferedWriter bw = new BufferedWriter(new
        OutputStreamWriter(process.
        getOutputStream()));
bw.write("source('UpdateLWR_Tester.R')\n");
bw.write("q()\n");
bw.close();
BufferedReader br = new BufferedReader(new
        InputStreamReader(process.
        getInputStream()));
String line = br.readLine();
System.out.println(line);
while (line!=null){
```

```
                        line = br.readLine();
                        System.out.println(line);
                }
                process.destroy();
                System.out.println("Appending csv to table"
                    );
                // transform the output into table
                StringBuffer lines = readFromCSV("LWR_Test1
                    .csv");
                insertQuery = DatabaseUtils.
                    insertFromBuffer(destinationTable,
                    columnNames, lines);
                DatabaseFactory.executeSQLUpdate(
                    insertQuery, dbconnection);
                System.out.println("The procedure was
                    successful");
                status = 1f;
        } catch (Exception e) {
                e.printStackTrace();
                System.out.println("warning: error in node
                    execution " + e.getLocalizedMessage());
                System.out.println("Insertion Query: "+
                    insertQuery);
                System.err.println("Error in node execution
                    " + e.getLocalizedMessage());
                return -1;
        } finally {
                if (dbconnection != null)
                        try {
                                dbconnection.close();
                        } catch (Exception e) {
                        }
        }
        return 0;
}

//setup phase of the algorithm
@Override
public void setup(AlgorithmConfiguration config) throws
    Exception {

        destinationTableLabel = config.getParam("TableLabel
            ");
        AnalysisLogger.getLogger().info("Table Label: "+
            destinationTableLabel);
        destinationTable = config.getParam("RealOutputTable
            ");
        AnalysisLogger.getLogger().info("Uderlying Table
            Name: "+destinationTable);
        originTable = config.getParam("LWR_Input");
        AnalysisLogger.getLogger().info("Original Table: "+
            originTable);
        familyColumn = config.getParam("FamilyColumn");
        AnalysisLogger.getLogger().info("Family Column: "+
            familyColumn);
        haspostprocessed = false;
```

```
        AnalysisLogger.getLogger().info("Initializing␣DB␣
            Connection");
        dbconnection = DatabaseUtils.initDBSession(config);
        List<Object> families = DatabaseFactory.
            executeSQLQuery(DatabaseUtils.
            getDinstictElements(originTable, familyColumn,
            ""), dbconnection);
        count = families.size();

        //create the table were the script will write the
            output
        DatabaseFactory.executeSQLUpdate(String.format(
            createOutputTable, destinationTable),
            dbconnection);
        AnalysisLogger.getLogger().info("Destination␣Table␣
            Created!␣Addressing␣" + count + "␣species");
}

@Override
public int getNumberOfRightElements() {
        return count; //each Worker node has to get all the
             elements in the right table
}

@Override
public int getNumberOfLeftElements() {
        return 1; //each Worker node has to get only one
            element in the left table
}

@Override
public void stop() {

        //if has not postprocessed, then abort the
            computations by removing the database table
        if (!haspostprocessed){
                try{
                        AnalysisLogger.getLogger().info("
                            The␣procedure␣did␣NOT␣correctly
                            ␣postprocessed␣....Removing␣
                            Table␣"+destinationTable+"␣
                            because␣of␣computation␣stop!");
                        DatabaseFactory.executeSQLUpdate(
                            DatabaseUtils.
                            dropTableStatement(
                            destinationTable), dbconnection
                            );
                }catch (Exception e) {
                        AnalysisLogger.getLogger().info("
                            Table␣"+destinationTable+"␣did␣
                            not␣exist");
                }
        }
        else
                AnalysisLogger.getLogger().info("The␣
                    procedure␣has␣correctly␣postprocessed:␣
                    shutting␣down␣the␣connection!");
```

```
                if (dbconnection != null)
                        try {
                                dbconnection.close();
                        } catch (Exception e) {
                        }
        }

        boolean haspostprocessed = false;
        @Override
        public void postProcess(boolean manageDuplicates, boolean
            manageFault) {
                haspostprocessed=true;
        }

}
```

Related Links

DataMiner Tutorial:

https://wiki.gcube-system.org/gcube/DataMiner_Manager

How to interact with DataMiner by means of a thin client:

https://wiki.gcube-system.org/gcube/How_to_Interact_with_the_Statistical_Manager_by_client

Currently integrated algorithms:

https://wiki.gcube-system.org/gcube/Statistical_Manager_Algorithms