MonALISA Repository User Guide

August 30, 2005

Chapter 1

General Features

1.1 Overview

MonALISA provides an easy mechanism to create clients able to use the discovery mechanism in JINI and to find all the active services. Such clients can subscribe to a set of parameters or filter agents to receive selected information from all the services or any service in particular. This offers the possibility to present global views from the dynamic set of services running in a distributed environment to higher level services.

The received values are stored locally into a relational database (PostgreSQL or MySQL). We are using a parallel multi-resolution storage architecture to optimize long-term queries in the database.

This information can also be used to create WEB repositories able to present a synthetic view of how large distributed systems perform. A servlet engine is used to present historical and real time values, statistics and graphical charts in a flexible way.

Basically, once started, the repository client registers with some predicates and stores the received values in the local database. A predicate from the configuration file has the following pattern: Farm / Cluster / Node / start_time / end_time / function_list. Any number of predicates can be specified in the configuration. These parameters (functions) are then plotted into a large variety of graphical charts or statistics tables, following the configuration files describing the views, and thus offering global or particular perspectives. The repository also uses an interactive map that shows the precise farm positioning along with some of the monitored parameters such as farm usage (number of busy/free nodes), VO utilisation (number of running jobs from each VO), gatekeeper's CPU usage and Internet links between sites (quality, utilisation). Thus, it offers much of the Interactive Client's features, adapted for web and emphasizing on data analysis.

The same mechanism is used to offer access to this information from mobile phones using the Wireless Access Protocol (WAP). Because of the limited capabilities of the mobile devices this service is restricted to real-time monitoring only.

Data can also be accessed through the WebServices interface. Currently there are methods for querying the last known values for some series (real-time information), history information for some series and a given time interval and last known services' configurations.

1.2 Requirements

In order to run the MonALISA Repository, the following program tools must be installed:

- Java 2 Development Kit, version 1.4.2+. You can download it from <http://java.sun.com/j2se/downloads/> We recommend using the latest version of JDK 1.5.0.

The Repository distribution comes with an embeded Tomcat web server and an embeded PostgreSQL database engine.

Chapter 2

Installation

2.1 Installation Instructions

NOTE

Do not try to install the service as root!

Make sure no running program is listening on the ports that PostgreSQL and Tomcat are (or will be) using. The default ports are 5544 for PostgreSQL and 8080 and 8005 for Tomcat. You can change the default configuration using the files in `conf/` dir. Please check these files before trying to install the repository.

Run `install.sh`. Should an error had occured please untar the package, correct the cause of the error and try again. The `install.sh` script will look up the Java home dir. If Java cannot be found in current path then the script asks for one and tests it. The Java home dir and the repository dir are saved in `conf/` directory in the `env.JAVA_HOME` and `env.REPOSITORY_DIR` files. You can edit these files later if you want to relocate the repository or the Java VM that is used.

Start the service using `start.sh`. You can control every service in part using `scripts/*.sh`. Test the web service typing http://localhost:8080/ in your browser's address bar. You should start seeing some values in the charts in a couple of minutes. To stop the service you can use `stop.sh`. `start.sh` and `stop.sh` should add/remove calls to `scripts/verify.sh` in your crontab, for example:

```
* * * * * ~/MLrepository/scripts/verify.sh >> ~/MLrepository/scripts/verify.log 2&>1
```

This script checks if the repository is running ok; if not, the repository is started/restarted.

To change the path to java home dir in the future,you have to edit the `conf/env.JAVA_HOME`. You can move the entire repository to another directory but you have to edit the `conf/env.REPOSITORY_DIR` and write the new path there.

This repository contains some Web Services Client examples. They are default configured to use the localhost:8080 as the web server to which they connect to. To change this behavior you may edit the `conf/env.MONALISA_WS` file. When you edit this file make sure there is no trailing line feed character at the end of the line.

You can keep your repository updated by running the `update.sh` script. Before running this script please BACKUP YOUR REPOSITORY. The script should correctly update the service and the additional software (database, web server etc) and it shouldn't cause any problems at all (at least it never happend for us), but it's better to have a backup in case something doesn't work well. Instead of using this script you can download the latest stable version from the site and manually update the components. If you proceed like this you should NOT touch at least the following folders:

- `/pgsql_store/data`

- `/JStoreClient/conf`
- `/conf`
- `/tomcat/webapps`
- `/tomcat/conf` (or at least modify the new tomcat config files to use the same tcp/ip ports)

2.2 Basic Configuration

The repository options regarding what data is collected and how it is stored can be configured from the `App.properties` file. There are several things you should check before starting the repository.

2.2.1 LUS and Group Configuration

You can select which LookUp Service(s) -LUS- you want to use. Multiple values are to be separated by ",". The default values are the two LUSs that the MonALISA system uses. You shouldn't modify this option in normal conditions.

```
 lia.Monitor.LUSs = monalisa.cern.ch,monalisa.cacr.caltech.edu
```

You have to mention the groups to which the farms monitored by this repository belong. You can collect data from more groups, this is a list of group names separated by ",". The names are case-sensitive so take good care how you write them. "grid3" is just a sample, please change this with your own group before starting the repository.

```
 lia.Monitor.group = grid3
```

2.2.2 Database Configuration

You can specify the database server name, the database name, the database port, a username and a password used to connect to the database. These are the default values that work with the embedded PostgreSQL database. If you have another, standalone, PostgreSQL or MySQL database server running you can provide the details of how to connect to an existing database here. For PostgreSQL the user should have the rights to create the plpgsql language, if it doesn't already exist, and some stored procedures in this language.

```
lia.Monitor.jdbcDriverString = org.postgresql.Driver

lia.Monitor.ServerName = 127.0.0.1

lia.Monitor.DatabaseName = mon_data

lia.Monitor.DatabasePort = 5545

lia.Monitor.UserName = mon_user

lia.Monitor.Pass = mon_pass
```

If you want to take a closer look at the tables stored in the database you cand use the database console, by running the `REP_HOME/scripts/pgsql_console.sh` script.

In the same directory, there are appropiate scripts used to start / stop the database, according to your convenience. You should not shut down the database while the repository is running, but you could for example shut down the data collecting application and create a backup of the database or other maintenance operations.

2.2.3 Logging Options

You may select the level of detail of the log, from INFO to FINEST.

```
handlers= java.util.logging.ConsoleHandler

java.util.logging.ConsoleHandler.level = INFO

java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
```

The log is written to `REP_HOME/JStoreClient/log.out` file.

2.2.4 Store Configuration

The store client has to register with some predicates in order to receive data. A predicate has the following pattern:

```
Farm/Cluster/Node/start_time/end_time/function_list
```

where function_list is a list of parameters needed to be stored, separated by "|". Multiple predicates are separated by ",".. For example:

```
lia.Monitor.JiniClient.Store.predicates=*/Master/*/-1/-1/Load5|%_IN|%_OUT,
              */VO_JOBS/*/-1/-1/Running Jobs|Held Jobs|Idle Jobs|TotalJobs,
              */VO_IO/*,
              */PN%/*/-1/-1/Load_05|Load_51,
              */MonaLisa/*/-1/-1/Cpu_%,
              */ABPing/*/-1/-1/RTT
```

means that you want to store Load5, any parameter having its name ending in _IN and any parameter having its name ending in _OUT from the Master cluster and so on.

You can also specify some global paramters to register with. These are used by the filters on the farms in order to send filtered data (eg: integrated, averages, totals, etc.).

```
lia.Monitor.JiniClient.Store.global_params=Load5,TotalIO_Rate_IN,TotalIO_Rate_OUT,No
```

These parameters are agregated values from all the processing nodes of a service (normally the PN cluster of the services). The options for data storage are explained in the `App.properties` file itself and concern the length of the time interval for data to be stored, number of values to be stored for that inteval, storage mode, etc. The default configuration stores history data for one year with two resolutions: 1 minute and 100 minutes. When displaying a history chart the servlet automatically selects the best data source, e.g. the lowest resolution structure that provides at least the desired number of points for the chart. This is very important when displaying long-term history such as 3 months or 1 year.

2.2.5 Tomcat Configuration

The server runs by default on 8080 port. If you have another application using the same port or simply want to change it, you may do so by editing the file `REP_HOME/tomcat/conf/server.xml`, namely the following tag:

```
<Connector compression="on" port="8080" >
```

and restart the repository by running `REP_HOME/scripts/stop_jstoreclient.sh` and `REP_HOME/scripts/start_jstoreclient.sh` scripts.

You may at any time enhance the content of the repository by adding new servlets or JSPs to the repository. We provide a sample servlet, which makes basic queries to the Cache (the object that holds the last known values for the data series that are collected) and to the database, in order to get historical data, and returns the results. The code for this servlet is located at `REP_HOME/tomcat/webapps/ROOT/WEB-INF/classes/SampleServlet.java` along with a script to compile the code.

To create a new servlet you have 2 options:

- you add your standalone source `*.java` file to `REP_HOME/tomcat/webapps/ROOT/WEB-INF/classes/` directory and compile it using the `compile.sh` script (if you need new classes in the classpath when compiling, you may add them by editing the "CP" variable in the script).
- you may add a JAR containing your servlet classes, to the directory `REP_HOME/tomcat/webapps/ROOT/WEB-INF/lib/`.

Once the servlets uploaded, to activate them you have to add corresponding entries to the `REP_HOME/tomcat/webapps/ROOT/WEB-INF/web.xml` file. You should add "servlet" and "servlet-mapping" tags for each servlet, following the pattern for the already working servlets.

NOTE

After each recompile of your new classes, don't forget to run

```
touch REP_HOME/tomcat/webapps/ROOT/WEB-INF/web.xml
```

so that the server reloads its context.

Another way of producing web pages is to create JSPs. This files do not need special mapping, compiling and so on, but you must be running a JDK after all because Tomcat automatically compiles the code when it detects a change in a JSP. This is a much faster way of developing a web interface but it has its limitations too (cannot define inner classes or static methods, cannot extend another JSP and so on).

Chapter 3

Configuration

Once the repository installed, up and running, you can configure it to display the suitable map(s) with the desired information, plot charts and define statistical views of the Grid. These views are generated by 3 servlets: ”display”, ”stats” and ”genimage” which receive as parameter the filename defining the properties of the specific view. Each of this view will be explained in the following sections. In order to make your views visible in the web repository, you have to modify the web interface, creating links to the servlet calls.

Here is an example for a generic call of a servlet, to create a chart plot or a statistics table: <`http://localhost:8` where:

- servlet - can be ”display”, ”stats” or ”genimage”
- name - the `name.properties` file containing the specific properties of the desired chart/table, without the ”.properties” extension. These configuration files are found in `REP_HOME/tomcat/webapps/ROOT/WEB-INF/conf/` directory.

The ”display” servlet can be used to produce realtime plots such as bar, spider web or pie charts or history charts with lines and shapes or overlapping areas. The realtime plots use the latest available information from the Cache object. This object holds the last known values just for a limited amount of time. You can adjust the data expire time from the

```
lia.web.Cache.RecentData
```

option in the `App.properties` file. By default the data expires after 15 minutes, which means that services that do not report data for more than 15 minutes are displayed as offline. Please see sample_history*, sample_realtime*, sample_pie*, sample_combined* files from `REP_HOME/tomcat/webapps/ROOT/WEB-INF/conf/` for more details on how to define your options.

”stats” servlet is used to produce statistics tables. It is good for showing a two-dimensional matrix of values and from the configuration options you can choose what the lines and the columns mean. Each cell is actually a predicate and you can define agregation functions for these predicates. Some often-used functions are last known value and the average for the last hour values. You can also define summarizing rows such as the sum/min/max/avg/stddev from each column or you can choose colors for the min and max values from each columns and the servlet will automatically create gradient colors for the actual values. You can see the sample_stats.properties file for more information on how to generate such views.

”genimage” is used to display a world map, or any other image, with the services at coordinates relative to the image that is used. You can define predicates that are used to control the size of the circles that represent the services, or the pie composition for each service or the links that unite the services and so on. A sample configuration file is sample_image.properties.

Chapter 4

FAQ

- **`1. How do I create a new chart?`** For each chart there is (at least) one .properties file that describes what needs to be plotted. These files are located in REPOSITORY_HOME/tomcat/webapps/ROOT/WE INF/conf.

  There are many types of charts that can be created, for each one there is a sample configuration file in that folder having each option explained.

  After you create a configuration file you can test it in the browser typing something like:

  http://localhost/display?page=your_new_file

> NOTE
>
> Do NOT specify the .properties extension of the file name. The extension is automatically appended to the name you give as parameter.You can safely call display with any file as parameter, it will redirect the request to the proper servlet.

- **`2. I want to change the look of the pages. How do I do that?`** All the html code that is used to create the pages is located in REPOSITORY_HOME/tomcat/webapps/ROOT/WEB-INF/res. When editing the .res files that contain the actual html code please make sure you don't change the special tags (eg. the ones enclosed by <<: and :>>).

  You can add dynamic html code to some of your pages like this:

  – edit `display/hist.res` or `display/rt.res` and enter a unique tag, we suggest something like <<:page.content.1:>>

  – in one/any .properties file add an option like:

    `page.content.1=html code you want to add`

- **`3. Can I start only the storage client, without the web interface?`** Sure thing, edit `REPOSITORY_HOME/JStoreClient/njGlobal` and comment out the line

  ```
  TOMCAT="true"
  ```

  and restart the repository. To enable the web interface again just uncomment that line and restart the repository.

- **`4. How can I get the data from the repository?`** There are more ways of doing this:

  – You can directly extract the data from the database, but we don't recommend this. We already have three database formats now and we don't guarantee there won't be some other format available in the future :).

  - You can use the WebServices interface that is automatically deployed with the repository. To use this please take a look at some sample programs in `REPOSITORY_HOME/WS-Clients`.
  - You can include the storage client in your own Java application. In `REPOSITORY_HOME/Embedded` is a simple application that starts the MonaLisa client and waits to receive some data. For each data that matches some rules the value that was received and the average value in the last 30 minutes for that data are displayed. The configuration file for this embedded client does not define any database so all the received data is lost. You can either edit the separate configuration file to add database support or implement your own means of storing the data you are interested in.
  - There is also a more direct method of connecting to the repository through TCP/IP, a sample code is provided in the `REPOSITORY_HOME/DirectClient` folder. Please take note that we don't recommend using this method to connect to the repository. The WebService is far more flexible for distance connections while embedding the repository in your application will provide the best possible performance. But this method can have its uses too so feel free to use it if it suits your needs.
  - Implement your own servlet/JSP/WebService that exports some data in your preferred format.

- **`5. How can I put data into the repository?`** We recommend you to write MonALISA service modules and set the repository to store the data that is produced by the service. But if you don't want to have a MonALISA service installed you can directly insert data into the repository. For this you have to write a script/program that writes on the standard output lines with the following format:

```
FARMNAME CLUSTER NODE FUNCTION VALUE TIME

FARMNAME : string
CLUSTER : string
NODE : string
FUNCTION : string
VALUE : double
TIME : longint (absolute time of the event in milliseconds,
                like System.currentTimeMillis() from Java)
```

The columns are separated by a single tab character (\t). A simple sample script that produces correct output:

```
#!/bin/bash

FARM=MyFarm
CLUSTER=LoadAvg
NODE=localhost
TIME=`date "+%s"000`

function process(){
      echo -e "${FARM}\t${CLUSTER}\t${NODE}\tLoad5\t$1\t${TIME}"
      echo -e "${FARM}\t${CLUSTER}\t${NODE}\tLoad10\t$2\t${TIME}"
      echo -e "${FARM}\t${CLUSTER}\t${NODE}\tLoad15\t$3\t${TIME}"
}

process `cat /proc/loadavg`
```

Now you need to modify the `App.properties` file located in `REPOSITORY_HOME/JStoreClient/conf`. You have to add/edit the following options:

```
lia.Monitor.JStore.inserters=N : N is the number of scripts you have
for each script you have to define two options:
lia.Monitor.JStore.execute_X.path=/path/to/your/script.sh
lia.Monitor.JStore.execute_X.time=how often is this script run (in seconds)
```

For example:

```
lia.Monitor.JStore.inserters=2

lia.Monitor.JStore.execute_0.path=/home/monalisa/MLrepository/bin/myscript.sh
lia.Monitor.JStore.execute_0.time=60

lia.Monitor.JStore.execute_1.path=/home/monalisa/MLrepository/bin/myotherscript.
lia.Monitor.JStore.execute_1.time=120
```

This method should be used when no other option is available. Use it on your own risk.