Using GPLMT for Large-Scale Deployment and Experimentation - User Guide

Matthias Wachs and Omar Tarabai

October 16, 2013

Contents

Abstract

GNUnet Parallel Large-scale Management Tool (`GPLMT`) is a tool to deploy and run experiments remotely on a large number of systems in parallel. In this user guide we describe how to use `GPLMT` and the graphical user interface to deploy, run and monitor software and experiments and retrieve results on different heterogeneous testbeds. We describe how to setup, configure GPLMT and use both the command line interface and the web based user interface to prepare systems, deploy software and experiments, run experiments retrieve results from the nodes.

Chapter 1

Introduction

The GNUnet Parallel Large-scale Management Tool (`GPLMT`) is a tool set to deploy, run, control and monitor experiments. It is designed to enable experimentation on different testbeds by providing a generic interface to interact with the testbeds and supports the experimentation on a large number of testbed nodes in parallel. The tool interacts with the testbeds to obtain information about the testbed environment and uses this information when conducting experiments.

Experiments are be defined in a extensible description language and be stored in a file. These Experiments can then be executed on the desired testbeds and without modification transfered to a different testbed. With this approach experiments can be easily repeated and experimentation results reproduced on the same or a different testbed.

`GPLMT` can be used as a stand alone command line tool, allowing the experimenter to use it to test and prepare his experiments or run his experiments as a batch-orientated in script. In addition a comfortable graphical user interface, tightly integrated with our Zabbix based monitoring platform, is provided allowing users to use the tool in a way convenient for end users. The user interface is web based, allowing platform independent and remote access to the experimentation infrastructure.

In this user guide we describe both the use of the command line interface in Section 3 as well as the use of the web interface in Section 4.

1.1 Features

The key features of GPLMT are:

- Generic multi target support
- Easy experiment repetition
- Extensible architecture
- Experiment description language

- Multi User Support
- Supports full experiment life cycle:
  Allocation, Preparation, Deployment, Execution, Control, Result Retrieval

1.2 Experimentation Work Flow

GPLMT is a tool designed to help an experimenter to perform distributed networking experimentation tasks on various differing testbeds and on a large number of experimentation nodes in parallel. GPLMT ensures reproducibility of experimentation results and provides the possibility of transferring experiments between testbeds.

With the common work flow to run a distributed network experiment the experimenter starts with defining the experiment he wants to conduct. He then allocates the required resources on the testbed, prepares the testbed by deploying software and data to the testbed and involved nodes. When the testbed is prepared the experimenter starts to execute the experiment on the testbed nodes, monitors the execution and when the experiment is finished, he inspects the final state of experiment and the collects the experiment data to evaluate the experiment. If the results are not satisfying he has to refine his experiment and start over from the beginning of the experimentation cycle.

With GPLMT the experimenter starts with defining his experiment and creates a so called *task list*, described in detail in Section 1.3, which contains the experimentation tasks to be executed during his experiment. This task list contains the steps to perform in a sequential order. Next the experimenter has to decide on which of the supported testbeds he wants to execute his experiment. In GPLMT these testbeds are called *targets* and, if the notion is supported by the testbed, the separate execution instances of the testbed of are called testbed *nodes*. GPLMT supports at the moment several different targets: local execution, remote execution using ssh or execution on the PlanetLab networking testbed. By using a generic approach additional targets can easily be added. If a target has to be prepared in a specific way to run the experiment the user can be specify these steps in a separate task list. This may for example involve software deployment, configuration tasks or other preparation tasks.

When the experiment and additional preparational tasks are defined the experimenter can start the execution of his experiment. He instructs GPLMT to execute the experiment on the specified target. Depending on the target GPLMT will than interact the testbed to obtain information about the current state of the testbed, for example the nodes available, and will then execute the tasks included in the task list(s) on the testbed or the respective node(s). The experimenter can monitor the state of execution and can get feedback

about the progress of the execution. If required the tool can collect results from the participating nodes.

When the execution is complete the experimenter can inspect the result of the single steps and collected results. If the results are not satisfying or the experiment has to be rerun he can easily trigger the execution of a new run of the experiment. By specifying a different target he can easily re-run the experiment on a different testbed under the same conditions.

1.3 Defining Experiments

The steps to be executed during an experiment or while preparing a target to run an experiment are defined in a so called *task list*. `GPLMT` provides an experiment description language which can be used to create such a task list or the graphical user interface can be used to create a task list.

A task list represents the sequence of operations, called *tasks*, to be executed during an experiment. When a task list is executed, `GPLMT` executes each single task contained in the task list on all participating nodes.

The possible types of task operations are:

- distributing data to nodes (*put*)
- execute a command on the host (*run*)
- execute a command depending on the current node (*run_per_host)*)
- retrieve data from the node (*get*)

Each task or sequence has assigned a descriptive name, a task ID and a flag if it is currently enabled. With each task the expected result of the task can be specified and how `GPLMT` should continue depending on the result. This can include a timeout the task is expected to finish within, the expected return code of the operation or expected output and if the experiment should continue if this task fails. Tasks correlated to each other can be grouped in task sequences to represent their relation.

A *put* operation is used to distribute files required for the experiment to the target nodes. The operation takes a source file to store on the testbed node as an input and a destination where to store this file on the node.

A *get* operation is used to obtain files from the target nodes and store them on the local machine. The operation takes a source file to retrieve from the testbed node as an input and a destination where to store this file on the local machine.

A *run* operation is used to execute commands on all nodes of this target. The operation takes a command and additional arguments to execute on all nodes participating in the experiment.

A *run_per_host* operation is used to execute specific commands on each node of the target. The operation takes a command and in addition a

command file containing the node specific commands. The command file consists of the node name and the command to execute on this specific host. So to execute command x on node a and command y on node b the file would like like:

```
a;x
b;y
```

An example task file first checking if the address gnunet.org can be pinged and then performing a traceroute looks like:

```
Example:
<?xml version="1.0" encoding="UTF-8"?>
<tasklist name="ping gnunet.org " xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
  <task>
        <id>0</id>
        <name>ping gnunet.org</name>
        <type>run</type>
        <command>ping</command>
        <arguments>gnunet.org</arguments>
        <timeout>10</timeout>
        <expected_return_code>0</expected_return_code>
        <expected_output></expected_output>
        <stop_on_fail>1</stop_on_fail>
    </task>
    <task>
        <id>1</id>
        <name>tracerout gnunet.org</name>
        <type>run</type>
        <command>tracepath</command>
        <arguments>gnunet.org</arguments>
        <timeout>30</timeout>
        <expected_return_code>0</expected_return_code>
        <expected_output></expected_output>
        <stop_on_fail>1</stop_on_fail>
  </task>
</tasklist>
```

More examples are included in the `contrib` folder of `GPLMT` . A convenient way for user to create these files is the use of the graphical user interface described in the remainder.

1.4 Supported Targets

To run experiments on different testbeds GPLMT supports the notion of a *target* creating an abstraction from how the underlying testbed to use is implemented and functioning. For GPLMT a target is an abstraction of testbed resources and how to access them. GPLMT assumes that each testbed consists of several nodes to run the experiment on. GPLMT can use a programmatic API to interact with the testbed if provided by the testbed to obtain information about the resources available. In addition to these set of resources each target has a way to provide access to resources and to perform operations on these resources.

By using this abstraction the experimenter can easily change the testbed to run the experiment on by specifying a different target for GPLMT but leave the experimentation description untouched.

Current targets for GPLMT are concurrent execution on a local machine (*local*), distributed execution on remote machines with ssh (*remote_ssh*)and execution on the PlanetLab testbed (*planetlab*).

With the *local* target the experiment is executed on the local machine. This can be useful to test your application before deploying it to a remote testbed infrastructure. Each instance is separated by being executed in a distinct node directory containing the node specific information to be used. The put and get operation copy experiment data to these directories and the run operations execute the commands in the specific directory.

The *remote_ssh* target allows to execute an experiment on a set of remote machines which can be accesses using ssh. This target is useful for an initial test of the distributed experiment. The *remote_ssh* target supports put and get operations using SCP and SFTP and both password based and public key based authentication. Nodes can be specified in detail by host name, user name and port number.

The *planetlab* target allows to deploy and run a networking experiment on the planetary scale PlanetLab testbed. GPLMT supports both PlanetLab Europe and PlanetLab Central and can interact with PlanetLab using the PlanetLab API to obtain the nodes currently assigned to the slice used and in the graphical user interface to assign new nodes to the slice. Data to the PlanetLab node are transfered using GPLMT 's SCP mode for the get and put operations since SFTP is not available with PlanetLab.

GPLMT by default uses all nodes available with a target but the experimenter can also instruct GPLMT to only use a subset of the nodes available or even only single node. Detailed instructions can be found in the sections on how to use the command line tool (Section 3) and the section describing the graphical user interface (Section 4)

Chapter 2

Installing GPLMT

This chapter covers the installation of `GPLMT` on a UNIX-style system. The steps described here should work for most recent distributions. Installation on Microsoft Windows Systems is not covered here.

2.1 Requirements

`GPLMT` is written in Python, so you need a Python interpreter installed. Please check your distribution manual how to install python if it is not already installed. We recommend the use of Python 2.7, but 2.5 should also work fine. More information about Python can be found on the Python website `http://www.python.org/`.

2.2 Installing Additional Requirements

Some few additional python modules are required for `GPLMT` . You can install them using the `pip` installer or following the installation instructions on the website. We recommend the use of `pip` for unexperienced users since it is a powerful but easy to use tool to manage python packages. To install the dependencies, `root` permission are required on most systems. The following python modules are additionally required:

ElementTree

- XML tasklist parsing
- Version: >= 1.3
- Available from: `http://effbot.org/zone/element-index.htm`

minixsv

- XML tasklist validation

- Version: >= 1.7
- Available from: `http://www.leuthe-net.de/DownloadMiniXsv.html`

Paramiko

- `SSH` communication
- Version: >= 0.9
- Available from: `http://www.lag.net/paramiko/`

To install the `pip` installer run the following commands:

```
$ curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py
$ sudo python get-pip.py
```

To install the requirements with the `pip` installer run the following commands:

```
$ sudo pip install ElementTree minixsv paramiko
```

Since now all requirements are satifisied, we can continue with installing `GPLMT` itself.

2.2.1 Common Errors

If you see a message like

```
$ pip install ElementTree minixsv paramiko
bash: /usr/bin/pip: No such file or directory
```

you have to run the `pip` installer with root super user permissions using the `sudo` command. Please run

```
$ sudo pip install ElementTree minixsv paramiko
bash: /usr/bin/pip: No such file or directory
```

2.3 Installation

The `GPLMT` tool can be obtained from our Subversion repository. To checkout the latest version you need a subversion[1] client installed, then run

```
$ svn co https://gnunet.org/svn/gnunet-planetlab/gplmt/
```

After a successful checkout a new directory `GPLMT` is created containing the tool, documentation and examples. To check if your checkout was successful and see how to use the `GPLMT` command line tool, run

```
$ cd gplmt
$ ./gplmt.py --help
```

[1] `http://subversion.tigris.org/`

If your installation was successful, you should a list of command line arguments you can pass to GPLMT .

2.3.1 Common Errors

If one of the required dependencies, listed in Section 2.2 is not installed, you should see a message like

```
paramiko SSH module is required for execution, please check README
```

2.4 Configuration

GPLMT can be configured using configuration files or by passing the required information on the command line. The configuration files are text files containing information as the nodes to execute the experiment on, the default SSH user name, PlanetLab user information, GPLMT logging configuration etc.

If the configuration information change frequently, the configuration information can be stored in a experiment specific configuration file which can be passed to the GPLMT tool using the -c command line switch. If the configuration information do not change frequently they can be stored in a default configuration file gplmt.conf in the .gplmt in the user's home directory. This default configuration files is loaded by default if no configuration file is explicitly specified.

The configuration file is structured in sections with each section containing information for a specific component of the tool:

- [gplmt]:
  tool specific configuration settings like task file, logging mechanism
- [ssh]:
  SSH specific configuration settings like user name, private key file etc.
- [planetlab]:
  PlanetLab specific configuration settings like API user name, slice name etc.

An example configuration file containing all valid configuration settings can be found in the appendix A.
To run GPLMT with a specific configuration file example.conf, run:

```
$ ./gplmt.py -c example.conf
```

To run GPLMT with the default configuration file run:

```
$ ./gplmt.py
```

2.4.1 The `gplmt` section

This section contains general `GPLMT` specific settings. The configuration values you can specific in this section are:

- `nodes = <file>`
  A file containing a list of nodes to execute this experiment on, one host per line
- `tasks = <file>`
  A file containing the task list to execute
- `notification = [simple|result]`
- `max_parallelism = 0`
  Maximum number of operations
- `target = [local|planetlab|remote_ssh]`
  The target to run the experiment on
- `userdir = <dir>`
  User directory used as a home base directory for get and put operations

2.4.2 The `planetlab` section

This section contains general PlanetLab specific settings. The configuration values you can specific in this section are:

- `slice = <file>`
  The PlanetLab slice to use
- `api_url = <url>`
  The PlanetLab API url depending if you have a PLC or PLE account
- `username = <username>`
  PlanetLab API username
- `password = <pasword>`
  PlanetLab API password, can be kept empy and will asked when running the tool

2.4.3 The `ssh` section

This section contains general `SSH` specific settings. The configuration values you can specific in this section are:

- `ssh_username = <username>`
  The `SSH` username

- `ssh_password = <password>`
  A `SSH` password
- `ssh_keyfile = <file>`
  A `SSH` private key file to use
- `ssh_transfer = [scp|sftp]`
  The transfer mode to use for `SSH`
- `ssh_use_known_hosts = [yes|no]`
  Load the system's `SSH` known hosts file
- `add_known_hostkey = [yes|no]`
  Add a unknown hostkey automatically to the system's known hosts file

2.4.4 Common Errors

If you cannot find or edit the default configuration in the user's directory, you have to create it first since the directory and the file is not created automatically. Run:

```
$ mkdir ~/.gplmt
$ touch ~/.gplmt/gplmt.conf
```

2.5 Defining Experiments

After installing and configuring the tool the next step is to define the tasks to perform in a experiment. For a detailed explanation please refer to Section 1.3. A experiment usually consists of many different individual steps to perform, here called *tasks*. The whole set of tasks is called a task list and represents a experiment. A task specifies a single step of an experiment. GPLMT's experimentation is XML based, so you can use any text editor to create these files. A more convenient way is to use the graphical user interface described later. In the `contrib` subdirectory the XML schema file `tasklist_schema.xsd` is included, giving a full description about the possibilities of the language. This file can be used to validate correctness of task lists. Task lists are also automatically validated before execution. More examples for task lists can be found in the `contrib/tasklists` folder.

Chapter 3

Using the Command Line

The GPLMT is invoked on the command line using:

```
$ ./gplmt.py
```

and help on how to use the tool and the possible arguments can be obtained using

```
$ ./gplmt.py --help
```

The most important arguments are:

- `-c, -config=FILENAME`
  use configuration file FILENAME
- `-n, -nodes=FILENAME`
  use nodes in file FILENAME
- `-l, -tasks=FILENAME`
  use tasks file FILENAME
- `-t, -target=TARGET`
  run experiment on target $TARGET = local|remote_sh|planetlab$
- `-a, -all`
  use all nodes assigned to PlanetLab slice instead of nodes file
- `-p, -password`
  password to access PlanetLab API
- `-H, -host`
  run task list only on given host host
- `-s, -startid`
  start with this task id

- `-h, -help`
  print help
- `-V, -verbose`
  be verbose

All arguments not given on the command line are loaded from the default configuration file if existing.

3.1 Running Tasks

In this section we try to execute an very simple experiment pinging the host gnunet.org once to see if we can reach the host from all nodes in the testbed. We expect a return code of 0 if succeeded:

```
<?xml version="1.0" encoding="UTF-8"?>
<tasklist name="ping␣gnunet.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
        <run id="0" name="check␣peer">
        <command>ping</command>
        <arguments>-c 1 gnunet.org</arguments>
        <timeout>10</timeout>
        <expected_return_code>0</expected_return_code>
        <expected_output></expected_output>
        <stop_on_fail>true</stop_on_fail>
        </run>
</tasklist>
```

The most important argument is the task list to execute and of course on which target to execute it. So if you want to execute task list *ping.tasks* on the default target configured with all available nodes you can use:

```
$ ./gplmt.py -l ping.tasks
```

You can also specify a specific target to run the experiment on:

```
$ ./gplmt.py -l ping.tasks -t local
$ ./gplmt.py -l ping.tasks -t remote_ssh
$ ./gplmt.py -l ping.tasks -t planetlab
```

These three commands execute the experiment *ping.tasks* on the local machine, all nodes configure with $remote_ssh$ and finally on all nodes available on the PlanetLab slice configure in the configuration file. By specifying the target, the specific methods to connect, copy data and execute tasks for this testbed are used.

Besides running the task on all configure hosts, you can also reduce involved nodes. To run the *ping.tasks* experiment remote host *test.local* using ssh or on PlanetLab node *test.planetlab* you can use:

```
$ ./gplmt.py -l ping.tasks -t remote_ssh -H test.local
$ ./gplmt.py -l ping.tasks -t planetlab -H test.planetlab
```

You can also give a file containing node names to GPLMT. If you have a file *mynodes.txt* containing 3 nodes:

```
a.node.local
b.node.local
c.node.local
```

You run the experiment on these nodes with for example ssh or PlanetLab using:

```
$ ./gplmt.py -l ping.tasks -t remote_ssh -n mynodes.txt
$ ./gplmt.py -l ping.tasks -t planetlab -n mynodes.txt
```

3.2 Monitoring Experiments with SSH

First we present the result logger configured, displaying information about the result of the experiment:

Successfully running the *ping.tasks* experiment with the result logger on a single host with ssh should look like:

```
 ./gplmt.py -l ping.task -t remote_ssh -H test.local
Loading task file : ping.task
Duplicate target to run on defined, command line wins!
Using target remote_ssh
test.local finished
test.local | ping gnunet.org | success
```

If it fails to connect to a node, GPLMT gives you detailed information about the error. For example if the user name "unknown_user" used is invalid:

```
 ./gplmt.py -l ping.task -t remote_ssh -H unknown_user@test.local
Loading task file : ping.task
Duplicate target to run on defined, command line wins!
Using target remote_ssh
test.local finished
fulcrum | failed in: connecting: Failed to connect: Authentication failed.
```

More detailled output can be obtained using the "-v" switch.

With the more detailed "simple" logger you can get more information about the progress of the experiment:

```
./gplmt.py -l ping.task -t remote_ssh -H test.local
Default configuration /home/user/.gplmt/gplmt.conf found
Loading configuration file '/home/user/.gplmt/gplmt.conf'
```

```
Loading task file : ping.task
Loading tasks file 'ping.task'
Task 0: check peer
Loaded 1 tasks
Duplicate target to run on defined, command line wins!
Using target remote_ssh
Using single node 'test.local'
Loading node_results 'test.local'
Loaded node 'test.local'
Starting execution on target 'remote_ssh'
Starting execution for node test.local
fulcrum : Loading known hosts
0 of 1 nodes finished
fulcrum : Trying to connect to 'test.local'
fulcrum : Using no information
0 of 1 nodes finished
test.local : Connected!
test.local : connected successfully
test.local : Starting tasklist 'ping gnunet.org'
test.local : Tasklist 'ping gnunet.org' started
test.local : Running task id 0 'check peer'
test.local : Task 'check peer' started
test.local : 'PING gnunet.org (131.159.74.67) 56(84) bytes of data.'
test.local : '64 bytes from gnunet.org (131.159.74.67): icmp_req=1 ttl=62 time=0.556

--- gnunet.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.556/0.556/0.556/0.000 ms'
test.local  : Task 'check peer' completed after 0.0236370563507 sec, exit code 0 was
test.local  : Task 'check peer' successful
test.local  : Task 'check peer' completed successfully
test.local  : Tasklist 'ping gnunet.org' completed successfully
test.local  : disconnected
test.local  : All tasks done for test.local
0 of 1 nodes finished
test.local  finished
1 of 1 nodes finished
```

3.3 Monitoring Experiments with PlanetLab

We now want to execute the same experiment on our PlanetLab slice *test_slice* with currently 3 nodes assigned:

```
node1.planetlab
```

```
node2.planetlab
node3.planetlab
```

To execute the same experiment on your PlanetLab slice, just modify the target to *planetlab*.

```
./gplmt.py -l ping.task -t planetlab
```

And we can see for the result logger:

```
Loading task file : ping.task
Duplicate target to run on defined, command line wins!
Using target planetlab
node2.planetlab finished
node3.planetlab finished
node1.planetlab finished
node2.planetlab | ping gnunet.org | success
node3.planetlab | ping gnunet.org | success
node1.planetlab | ping gnunet.org | success
```

And for the simple logger:

```
./gplmt.py -l ping.task -t planetlab
Loading task file : ping.task
Using target planetlab
node2.planetlab : connected successfully
node2.planetlab : Tasklist 'ping gnunet.org ' started
node2.planetlab : Task 'check peer' started
node2.planetlab : Task 'check peer' completed successfully
node2.planetlab : Tasklist 'ping gnunet.org ' completed successfully
node2.planetlab : disconnected
node2.planetlab finished
node3.planetlab : connected successfully
node3.planetlab : Tasklist 'ping gnunet.org ' started
node3.planetlab : Task 'check peer' started
node1.planetlab : connected successfully
node1.planetlab : Tasklist 'ping gnunet.org ' started
node1.planetlab : Task 'check peer' started
node3.planetlab : Task 'check peer' completed successfully
node3.planetlab : Tasklist 'ping gnunet.org ' completed successfully
node3.planetlab : disconnected
node3.planetlab finished
node1.planetlab : Task 'check peer' completed successfully
node1.planetlab : Tasklist 'ping gnunet.org ' completed successfully
node1.planetlab : disconnected
node1.planetlab finished
```

Chapter 4

Using the Graphical User Interface

To use the graphical user interface, you need to log into the eclectic experimentation platform. In the menu you can see the right-most menu entry "Experiment", select this item to get to the `GPLMT` user interface.

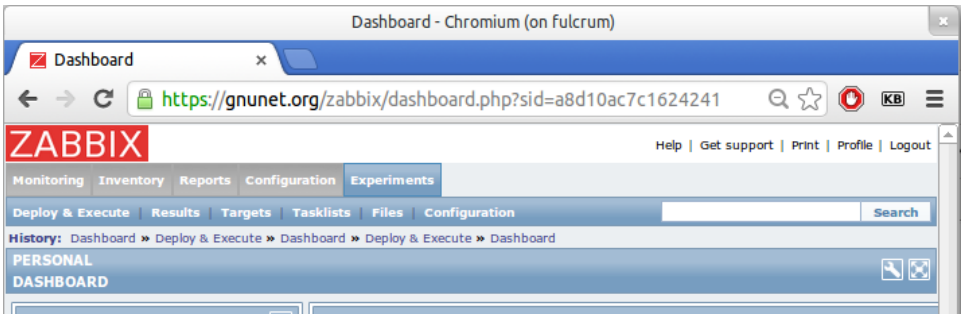

Figure 4.1: GPLMT GUI in the Zabbix menu

As you can see in image 4 provides the GPLMT GUI multiple submenu points:

- Deploy & Execute:
  Starts a new deployment or experimentation operation
- Results:
  Show results of previous executions
- Targets:
  Configure testbed targets
- Tasklists:
  Create or manage tasklists

- Files:
  Manage files to deploy and keyfiles
- Configuration:
  Manage your GPLMT configuration

As a first time user, you should select "Configuration" to configure your user data how to access your testbeds.

4.1 Configure GPLMT

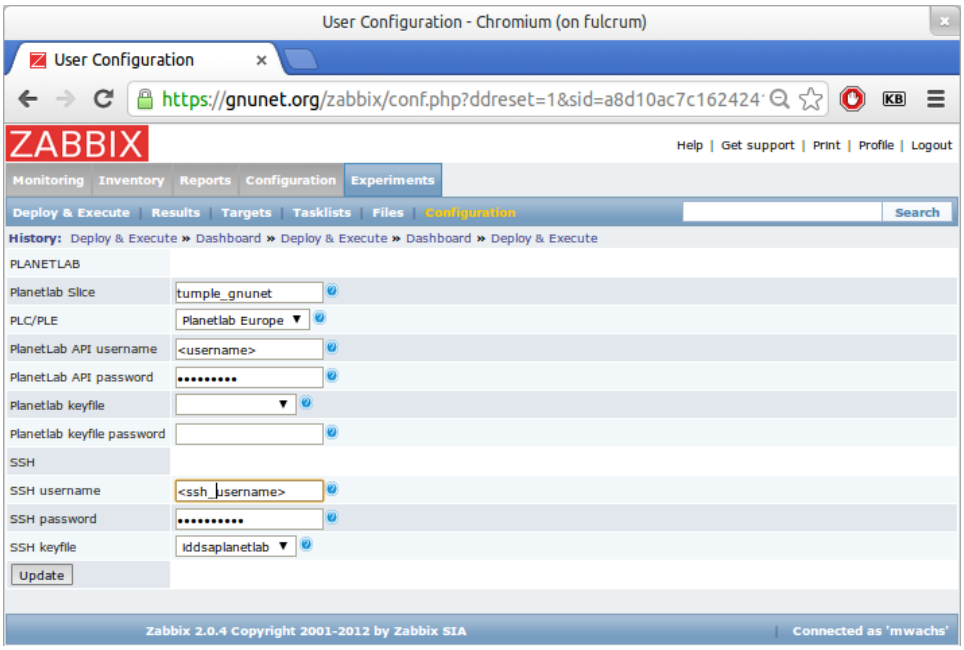

Figure 4.2: GPLMT GUI Configuration Menu

In the "Configuration" you can configure how you access the various testbeds supported by `GPLMT` .

The first section is PlanetLab related. In the first box you specify the PlanetLab Slice to use, followed by the option if your slice is a PlanetLab Central or PlanetLab Europe slice. This affects the API used to obtain and modify the nodes assigned to your slice. In addition you have to specify your PlanetLab credentials you use to access the PlanetLab web site. These credentials are used to log in with the PlanetLab API. With the PlanetLab key file box you select a SSH key file to log in to the nodes and the pass phrase for the key file. The key file can be uploaded in the "Files" menu.

Attention: Please this SSH key file only PlanetLab!

The next section configures GPLMT's SSH target. Here you can specify the default SSH user name and password to log into the SSH nodes. You can also specify a SSH private key file to log into the nodes. This key file can be uploaded in the "Files" Menu. This key file can be different to the PlanetLab key file.

After inserting your credential click "Update" to apply the changes.

4.2 Managing Files

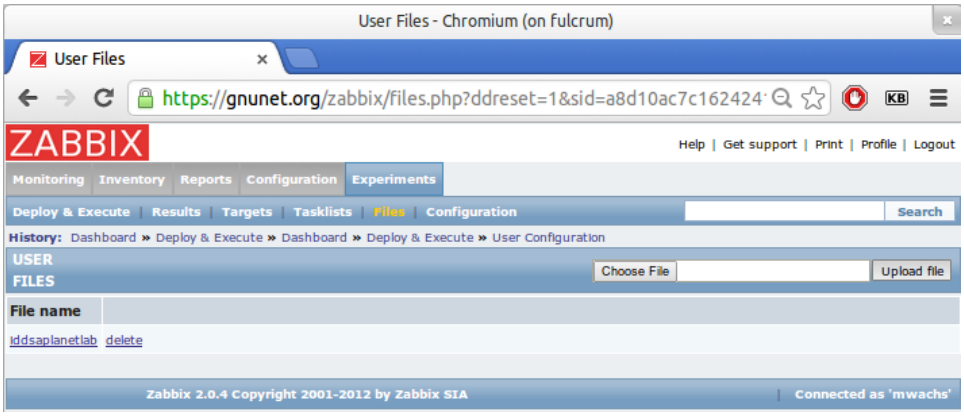

Figure 4.3: GPLMT GUI File Storage Menu

The "Files" menu provides you a place to store your files on the server. Here you can upload the files to deploy during an experiment or your SSH key files. You can upload new file using the "Choose File" button and in the next step select the local file to upload to the server. A list of the files currently available on the server is shown below, here including the "iddsaplanetlab" file to use as SSH private key file to access PlanetLab. Files can be deleted using the delete link next to the file.

4.3 Configuring Targets

In the "Targets" menu, you can configure the testbed targets and the default nodes to use with this target. First of all, you select the Target using the "Target" drop down box, here you can select "SSH" or PlanetLab.

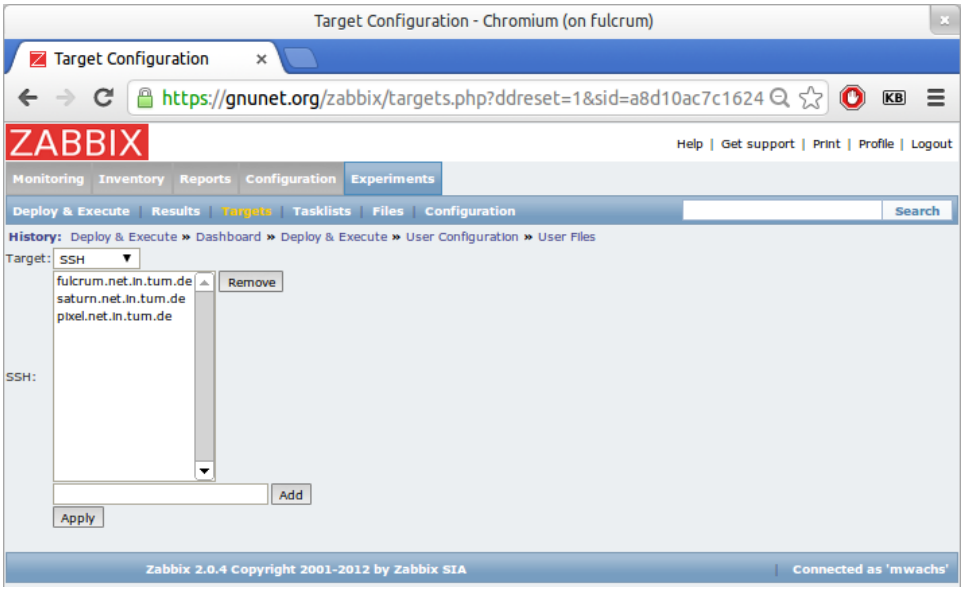

Figure 4.4: SSH Target Configuration

4.3.1 Configure the SSH Target

As shown in figure 4.3.1, when SSH is selected, you see a list of SSH nodes currently assigned to this target. To add a new node, insert the

- hostname, e.g. `local.net`
- hostname and username, e.g. `user@local.net`
- hostname, username and port, e.g. `user@local.net:2222`

and click "Add" to add the new host. To remove a host, select the host in the list and click "Remove". When finished, click "Apply "

4.3.2 Configure the PlanetLab Target

When "PlanetLab" is selected in the Target drop down box, you can manage the nodes currently assigned to the PlanetLab slice configured. As shown in Figure 4.3.2, you now have basically two select boxes: in the left box you have the list of nodes currently assigned and on the right a select box with nodes not assigned, but available. This menu interacts with the PlanetLab API and retrieves the list of nodes assigned to your slice. If you add or remove nodes, `GPLMT` will update your slice configuration.

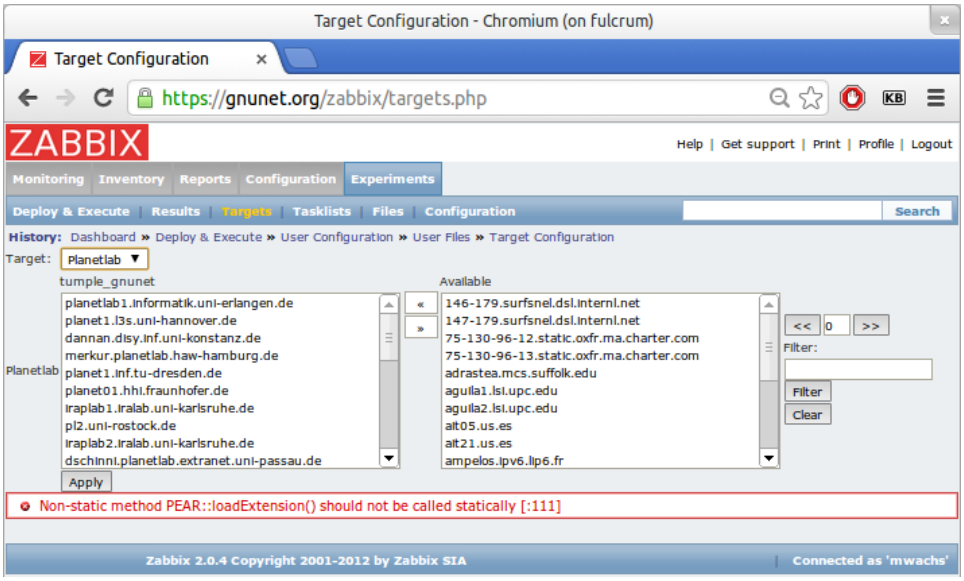

Figure 4.5: PlanetLab Target Configuration

Attention: Adding and removing nodes modifies the nodes assigned to your slice!

To remove nodes from your slice select the nodes in the left box, the click "»" between the boxes to move these nodes to the available box. When done click "Apply". To assign new nodes to your slice, select nodes in the right box and click the "«" to assign these available nodes to your slices.

You can apply a filter on the list of nodes available, so to for example search for nodes in China, enter ".cn" in the "Filter" box and click "Filter". To remove the Filter click on "Clear".

4.4 Managing Task Lists

In the "Tasklists" you can manage existing or create new task lists. When the menu is selected, you get a list of your current tasklists as shown in figure 4.4. You can modify a task list by clicking on the name or delete it by clicking "Delete". To create a new task list, click on "Create tasklist", to modify an existing click on the name.

A new task list created as shown in figure 4.4. First you specify a name and give a description for the task list. If you select "Shared", the task list will be available to other users. In the next section you create the tasks, the task list has to execute. In figure 4.4 a short example to display the system information on a testbed nodes is shown. We have a single

Figure 4.6: Manage Task Lists

sequence containing one *run* task. Tasks can be moved up and down using the respective buttons and new tasks added by selecting the respective type of task in the select box and then clicking "Add". A task can be removed using the "delete" button.

A task has a "Name" field and an "Enabled" flag. For *run* tasks a command to execute is given and additional arguments. The "timeout" field specifies the maximum time the command is allowed to run. The expected return code specifies the expected return code and the expected output can contain a substring the command output has to contain. If the "Stop on fail" flag is checked, the task list execution is aborted if the expected values are not correct.

More sequences and tasks can be nested and removed, as the user likes. When the task list is created, click "Update" and the task list will be stored on the server.

4.5 Deploy and Execute

Now we can continue with executing our new task list on the testbeds we configured before. To do so we select the "Deploy & Execute" menu. This will present us a drop down box to select the target: "SSH" or "PlanetLab"

4.5.1 Execute with SSH

When we execute our new task list with SSH, we will see two select boxes: the left box contains all nodes previously configured in the "Targets" section. We can now refine the nodes we want to execute the command on. In Figure 4.5.1 we see two nodes excluded from execution, so the task list will

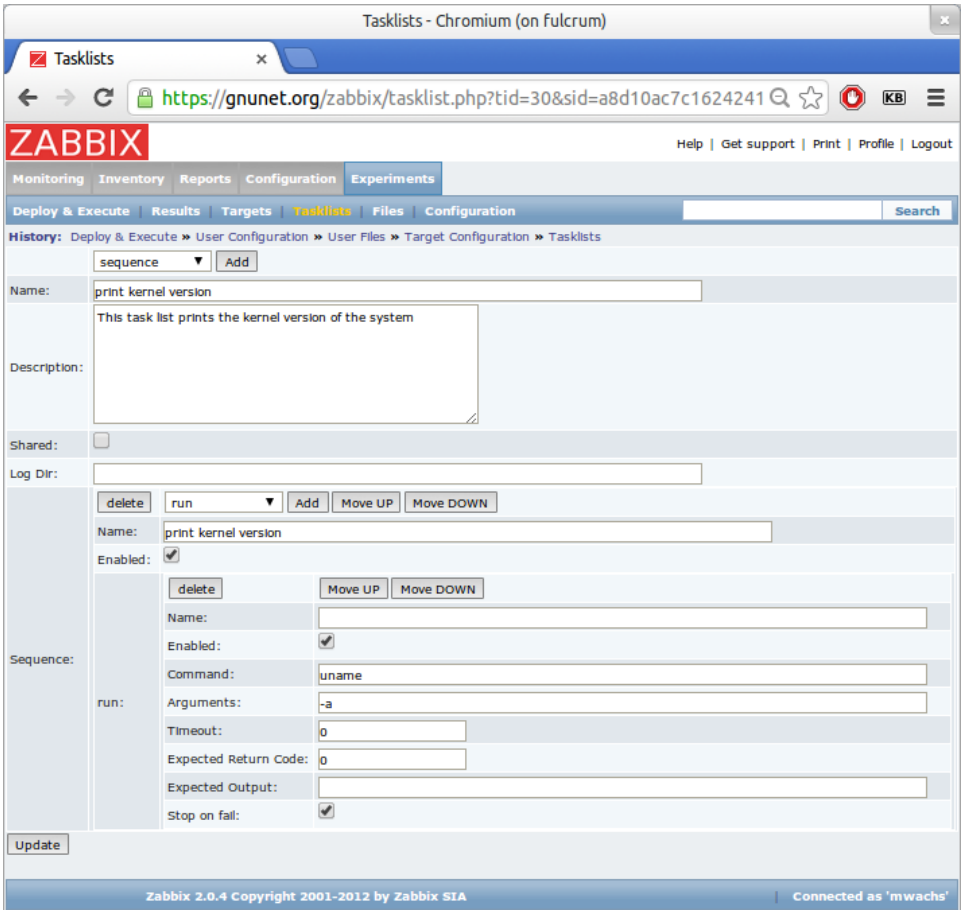

Figure 4.7: Create a new Task List

only be executed on host `fulcrum.net.in.tum.de` with SSH.

Below the node selection boxes, we can select the task list to execute, so here we selected the task list to print the kernel version created in the previous section. We can also execute instead if a task list just a single command by specifying this command in the "Command" box. In our case we could specify the "`uname -a`" command here directly. When the nodes and the command or task list to run with SSH are specified, you can click "Run". This will issue the execution to `GPLMT` and the result will be listed in "Results".

4.5.2 Execute with PlanetLab

When we want to execute our task list on PlanetLab, we select PlanetLab in the drop down box. We will see a menu as shown in Figure 4.5.2. The

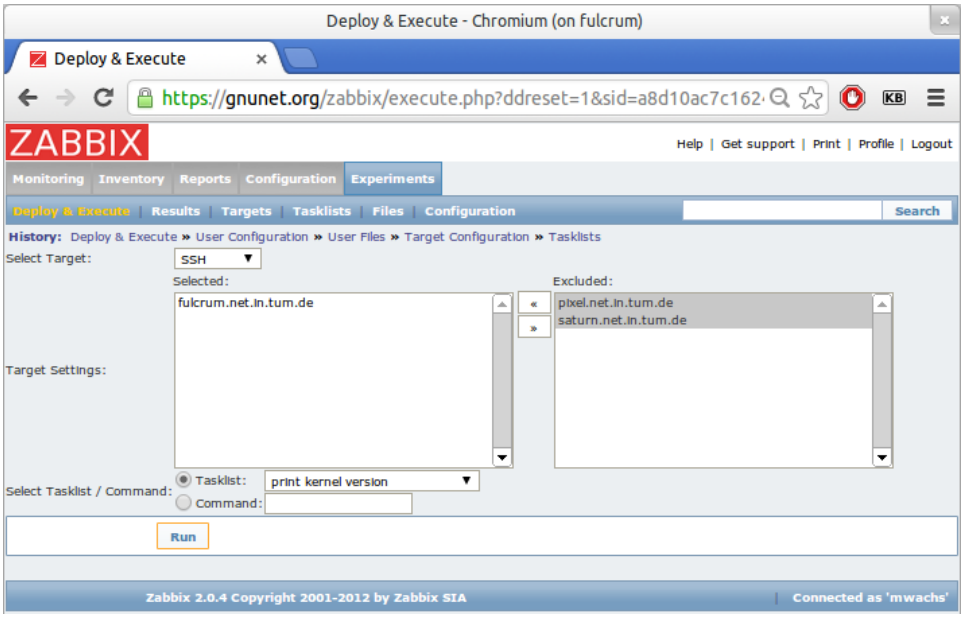

Figure 4.8: Execute a Task List with SSH

left box contains all nodes assigned in the slice, the right box contains nodes excluded from the execution (Note: these nodes are only excluded, not removed from the slice). Below the node selection boxes, we can select the task list or a single command to execute. When we are ready, we press "Run" and the execution will be issued and the results will be listed in the "Results" section.

4.6 Results

The results page shows the results for the different experiments executed. By using the "+/-" buttons, you can expand the details for every experiment executed. By clicking on "log" the resulting logs will be displayed and you can use the "re-run" button to execute an experiment one more time. If you do not need results, you can delete them using the delete button.

When the results for an experiment are expanded the specific results for every involved node can be inspected. In Figure 4.6 we see the results for the execution of our previously created task list on three PlanetLab nodes. On two of the nodes, the task list was successfully executed, but one node failed. To examine issue with this node, we can click on "log" and see the log messages for this node. As we can see in the Figure, we were not able to login to the node and therefore the execution failed.

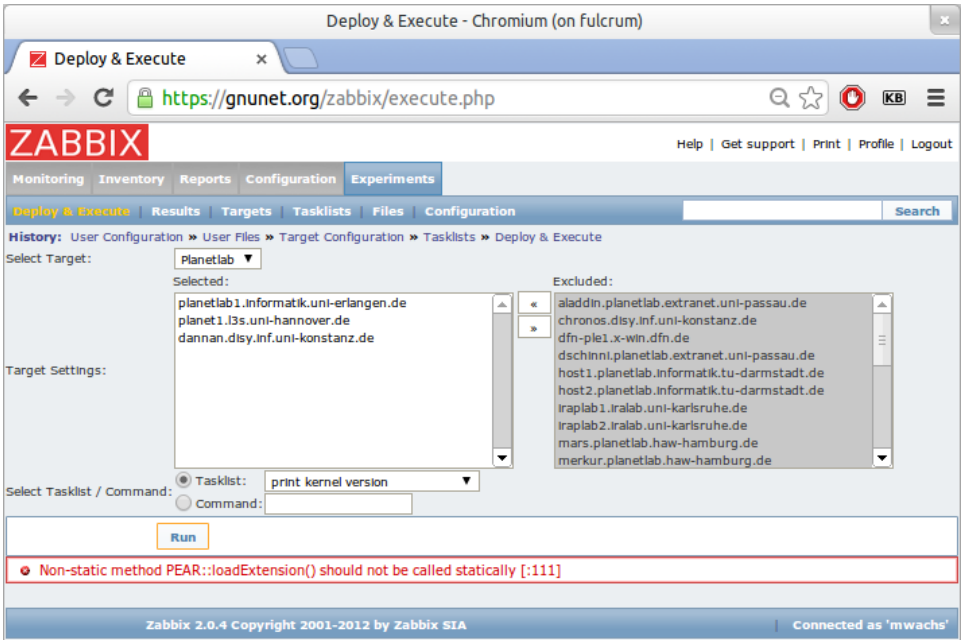

Figure 4.9: Execute a Task List on PlanetLab

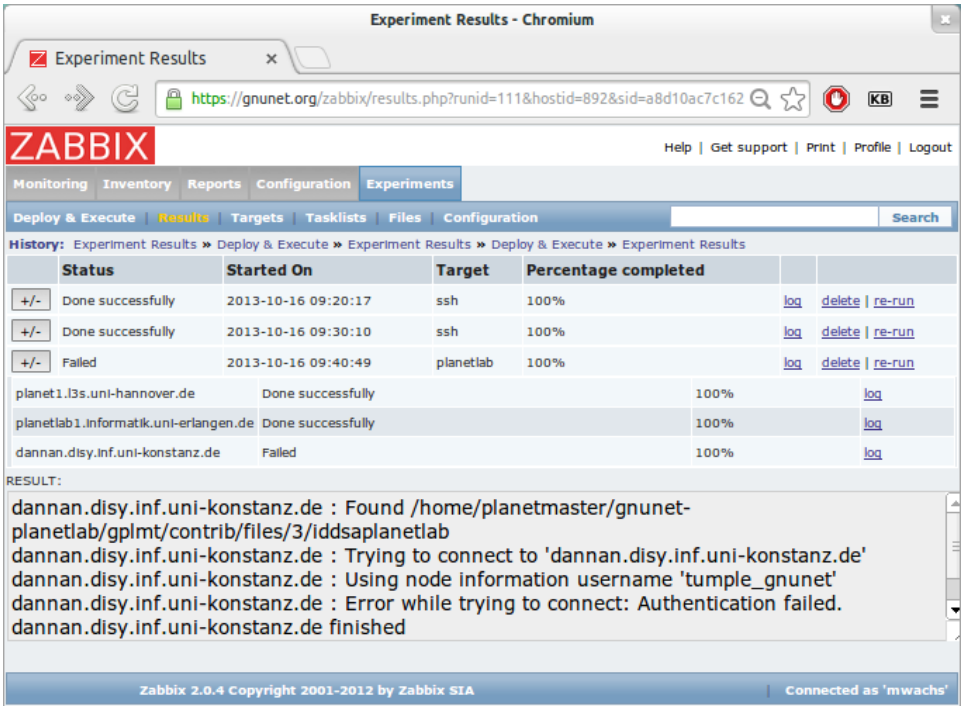

Figure 4.10: results

Appendices

Appendix A

GPLMT Configuration File

```
[gplmt]
# File containing the nodes to use
nodes = contrib/test_node.nodes
# File containing the tasks to execute
tasks = contrib/tasklists/check_node.xml
# Default target to execute experiment on [local|planetlab|remote_ssh]
target = planetlab
# Which notification mechanism to use [simple|result]:
notification = result
# Maximum number of parallel operations [0 == unlimited]
max_parallelism = 100
# If set, the userdir directory will be used for get/put operations
# userdir = /home/gplmt/
# Default target to execute experiment on [local|planetlab|remote_ssh]
# target = planetlab

[planetlab]
# Planetlab API username
username = <foo@bar>
# Planetlab API password
password = <password>
# Name of slice to use
slice = tumple_gnunet
# If you have a PlanetLab Europe account
api_url = https://www.planet-lab.eu/PLCAPI/
# If you have a PlanetLab Central account
#api_url = https://www.planet-lab.org/PLCAPI/

[ssh]
# SSH username
```

```
ssh_username = <username>
# SSH keyfile
ssh_keyfile = ~/.ssh/id_dsa_planetlab
# SSH password
ssh_password = <password>
# Protocol for put get operations [sct|sftp]
ssh_transfer = scp
#ssh_transfer = sftp
# Use systems SSH "known hosts" file
ssh_use_known_hosts = yes
# Add node hostkeys automatically
add_unkown_hostkeys = yes
```