The Grid[Way] Job Template Manager, a tool for parameter sweeping

Alejandro Lorca[*], Eduardo Huedo[†], Ignacio M. Llorente[‡]

Facultad de Informática, Universidad Complutense de Madrid,
C/ Prof. José García Santesmases s/n, E-28040, Madrid, Spain

October 22, 2018

Abstract

Parameter sweeping is a widely used algorithmic technique in computational science. It is specially suited for high-throughput computing since the jobs evaluating the parameter space are loosely coupled or independent.

A tool that integrates the modeling of a parameter study with the control of jobs in a distributed architecture is presented. The main task is to facilitate the creation and deletion of job templates, which are the elements describing the jobs to be run. Extra functionality relies upon the GridWay Metascheduler, acting as the middleware layer for job submission and control. It supports interesting features like multi-dimensional sweeping space, wildcarding of parameters, functional evaluation of ranges, value-skipping and job template automatic indexation.

The use of this tool increases the reliability of the parameter sweep study thanks to the systematic bookkeping of job templates and respective job statuses. Furthermore, it simplifies the porting of the target application to the grid reducing the required amount of time and effort.

[*]Corresponding author, e-Mail:alejandro.lorca@fdi.ucm.es
[†]e-Mail:ehuedo@fdi.ucm.es
[‡]e-Mail:llorente@dacya.ucm.es

1

1 Program summary

- *Title of the program*: Grid[Way] Job Template Manager (version 1.0, released on February 26th, 2010).

- *Catalogue identifier*:

- *Program obtainable from*:
  http://dev.gridway.org/projects/gwjobtemplatemanager.

- *Computer*: any (tested on PC x86 and x86_64).

- *Operating system*: Unix, GNU/Linux (tested on Ubuntu 9.04, Scientific Linux 4.7, centOS 5.4), Mac OS X (tested on Snow Leopard 10.6).

- *Programming language used*: perl 5.8.5 and above.

- *Additional programs/libraries used*: The GridWay Metascheduler [1].

- *Memory required to execute with typical data*: 10MB.

- *No. of processors used*: 1.

- *No. of bytes in distributed program, including test data, etc.*: 20799.

- *Distribution format*: Gzipped tar file.

- *High-speed storage required*: No.

- *Keywords*: e-science, parameter sweep, grid computing, middleware, high-throughput computing.

- *Nature of the physical problem*: To parameterize and manage an application running on a grid or cluster.

- *Method of solution*: Generation of job templates as a cross product of the input parameter sets. Also management of the job template files including the job submission to the grid, control and information retrieval.

- *Restriction of the complexity of the problem*: The parameter sweep is limited by disk space during generation of the job templates. The wildcarding of parameters cannot be done in decreasing order. Job submission, control and information is delegated to the GridWay Metascheduler.

- *Typical running time*: From half a second in the simplest operation to a few minutes for thousands of exponential sampling parameters.

2 Introduction

Parameter sweeping is a very common strategy in order to solve complex scientific problems. It consists on scanning different points on a relevant parameter space region. There are many possible interests on doing such, for instance, the optimization of a functional by finding its minima over the scanned region, to parameterize an analytical function with the help of interpolated values or to understand different regimes of simulated physical models. The variety of the scientific disciplines profiting from this strategy is overwhelming; experimental high-energy physics, astrophysics and astroparticle physics, life sciences, computational chemistry, earth sciences, computational linguistics, game theory, etc, are just prominent examples.

The evaluation of parameter space points is a simple problem within computational theory, but requires, in turn, some pre-processing of the problem according to a given search criteria, the systematic exploration of the parameter space points and a final data post-processing.

According to the application profile three different categories can be established for better understanding the scope of the tool:

- Parallelization. The second stage mentioned above, in which the parameter space is explored, requires the evaluation of some functional for each point. Making this stage *parallel* is in some cases unfeasible and the applications are run in *sequential* mode. Those parallelizable applications require some porting (i.e. either to cluster, grid or supercomputers) to improve the execution time, surmount memory limitations, etc.

- Interactivity. User intervention might be automated to a certain degree depending on the application profile. If it is not required at any stage of the computation we will consider the applications as *non-interactive*. Otherwise, for the *interactive* ones, the user is obliged to take some decision during execution.

- Recursion. On one hand, *single-pass* applications do not profit from the knowledge of the parameter space exploration or the results at a post-processing stage. On the other hand, if the application is aware of such feedback, an improved range of the parameter space can be chosen for optimizing results or further testing. This workflow is common in *master-worker* execution profiles.

With such classifications in mind, the *parallel, non-interactive, single-pass* applications are the target group for the Grid[Way] Job Template Manager, specially those with a large amount of independent tasks to be accomplished.

Large-scale problems well suited to high-throughput computing can be tackled in different manners, from last generation supercomputers to distributed solutions such as workstations or PCs. Indeed, grid computing has emerged during the last two decades as a new and affordable computational paradigm. The main distinguishing feature of this distributed architecture is the integration of different computing resources, typically involving high performance clusters (computing elements) and massive database storages (storage elements) hosted on universities, research centres, private companies, etc. The access to the grid is mediated through a software layer, usually referred to as the *middleware*, which takes care of the user's certificate checking, data transfer, host monitoring, file registering, and many other additional services.

There are other tools on the market dealing with parameter studies on the grid, like Nimrod/G [2] or APST[3] from AppLeS [4]. Nimrod/G possess a "parametric engine" which is the agent centralizing the parameterization of the experiment, job creation, submission and control. Nimrod/G requires a static declarative language, allowing multi-dimensional analysis to set of strings and basic real ranges. Alternatively, the AppLeS Parameter Sweep Template uses a simple, XML-based[1] interface which can be used from the command-line or called from scripts, but it is left to the user the enumeration of individual tasks. In both cases, the scheduling and other grid-related tasks are part of the projects. This is not the case of the Grid[Way] Job Template Manager, because the GridWay Metascheduler handles those operations directly. As will be shown later, the reason for naming our tool with a bracket in the word Grid[Way] tries to make clear the relation to the grid technology, being optional the underlying use of GridWay.

GridWay itself contains a mechanism to proceed with a collection of jobs (i.e. job array)[5], but it is certainly limiting: only single parameter variation through ingeters, both in step and values, can be achieved. This is somewhat similar to the possibilities integrated on the gLite WMS job description language [6]. A remarkable effort to stardardize the parameter sweeps has been carried out [7], but the approach is more a formal extension of the job standard description language than a useful grammar easy to by remembered by the end user.

The structure of the paper is as follows: The Sect. 3 describes the method and framework used for the composition of the parameter sweep region directly from the spanning of the one-dimensional parameter sets. It also describes the syntax of the parameter file. Sect. 4 moves on to the technical steps to have the tool up and running while Sect. 5 shows the general use

---

[1]Extensible Markup Language.

4

of the tool. Several examples have been worked out in detail in Sect. 6 with increasing dificulty. Finally we summarize the work in Sect. 7.

3   Algorithm

The basic working environment of the program is depicted in Fig. 1. The user specifies what is going to be performed through the command line interface and, depending on which tasks has been asked for, some input files will be required. The parameter sweep specification comes from the parameter file and job template appendices from the template file. During the processing, some calls to the GridWay Metascheduler are required for submitting, purging, killing and getting information about jobs. The main outcome of the tool are the job template files, which contain the information about the parameter sweep study and are ready for submission. The information about the job creation and deletion, as well as from the jobs coming from these job template files is available through the standard output and error. Additionally, if GridWay is used, the corresponding jobs' output will be available as extra files.
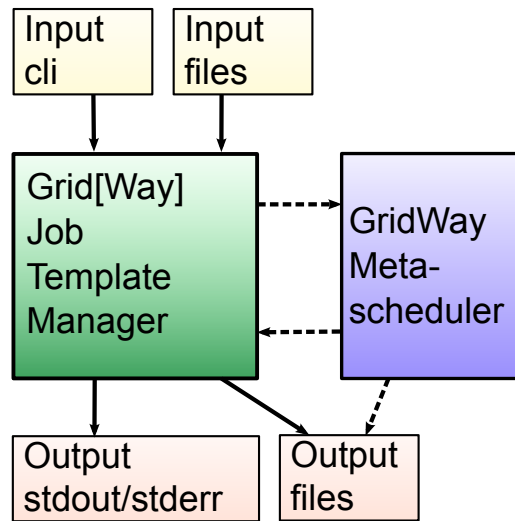

Figure 1: Grid[Way] Job Template Manager framework. Continuous lines represent input/output while dashed lines indicates interaction with GridWay through optional instructions and information feedback.

3.1 Mathematical framework

The starting point for carrying out a parameter study is to define which parameter points are considered for evaluation. Let denote by $\mathbf{r}$ a parameter point belonging to the direct product space of suitable dimensions $\mathbf{R}$:

$$\mathbf{r} \in \mathbf{R}, \quad \mathbf{R} = R_1 \times \cdots \times R_n, \tag{1}$$

being R_i the respective one-dimensional space of the i-th argument.

A given set of points P of arbitrary size m forms the complete parameter sweep set:

$$P = \{\mathbf{r}_j : \mathbf{r}_j \in \mathbf{R}; 0 \le j \le m - 1\}, \tag{2}$$

but for simplification we will consider only the case where any element $\mathbf{r}_j$ comes from a Cartesian product of subsets $P_i \subset R_i$ of some chosen elements, each subset with size m_i. Thus,

$$P = P_1 \times P_2 \times \cdots \times P_n, \tag{3}$$

$$m = m_1 m_2 \cdots m_n = \prod_{i=1}^{n} m_i. \tag{4}$$

Indexing the n-tuple elements of P is trivial when following the natural ordering of each set P_i until it gets exhausted, then incrementing the previous set.

Note the difference in labeling convention: the elements in each set P_i are indexed from 0 to $m_i - 1$, while the composition of the n-tuples correspond to $1, 2, \ldots, n$. This is because we expect the user to feed the application with n command line arguments, being each job uniquely indexed:

$$
\begin{aligned}
\mathbf{r}_0 &= (r_{1,0}, r_{2,0}, \ldots, r_{n,0}), \\
\mathbf{r}_1 &= (r_{1,0}, r_{2,0}, \ldots, r_{n,1}), \\
\vdots &= \vdots \\
\mathbf{r}_{m-1} &= (r_{1,m_1-1}, r_{2,m_2-1}, \ldots, r_{n,m_n-1}).
\end{aligned}
\tag{5}
$$

We have not yet described which is the allowed nature of the spaces R_i where the subsets P_i are included. Let us discuss this issue altogether with the mechanism to model the study.

3.2 Parameter file parsing

Next, the way to declare the sets P_i for each argument is explained. The declaration grammar has been chosen to be fairly simple for user's convenience:

- The parametric study, denoted by P is declared as n valid sentences in the parameter file.

- Each set P_i is declared at the valid i-th sentence of the file, one per line. It contains instructions to define the structure of the set and value assignments.

- The words of the language are KEY=VALUE pairs, separated by blank spaces.

The structure of each set is declared in the first word. The following possibilities are currently available:

1. `LOOPTYPE=LIST`. The set consists of a list of given values. These are assigned through an arbitrary number of `VALUE=`$r_{i,j}$ pairs. Admitted values are taken as strings.

2. `LOOPTYPE=RANGE`. The set is built according to a linear range of values. The keys `START` and `END` define the initial and final elements of the set, while the tags `STEP` or `POINTS` fix either the iteration increment between each element or the total number of points of the set respectively. Admitted values can be entered in the following formats:

   a) integer (i.e. `-4` or `54764563`),

   b) floating point up to 16 digits (i.e. `567884.2234`),

   c) scientific up to $10^{\pm 308}$ (i.e. `1.4E-12`),

   d) characters (i.e. `j` or `z`),

   e) arbitrary-size[2] (i.e. `100000000000000000000000000001`).

3. `LOOPTYPE=EXPRANGE`. It is similar to `RANGE` but interpolates exponentially between the initial and final elements. Therefore a `STEP=1` means exponential increases by one order of magnitude. Admitted values take the same form than for `RANGE`.

A summary of the correct sentence composition is given in Table 1, where the square brackets ([ ]) enclose optional words, and the pipe symbol (|) refers to mutually exclusive pairs.

Additionally, there are two other keys which may be included for modifying the evaluation of the sets, specially suited to the `RANGE` and `EXPRANGE`:

- `SKIP`. It permits to remove the specific point from the set.

---

[2]Should the user require this feature, then the the following option through the command line interface has to be added: `--config use_bignum=1`.

Structure	Assignment
`LOOPTYPE=LIST`	`VALUE=`$r_{i,0}$ $[$`VALUE=`$r_{i,1}$ ... `VALUE=`$r_{i,m_i-1}]$
`LOOPTYPE=RANGE`	`START=`$r_{i,0}$ `END=`r_{i,m_i-1} $[$`STEP=`$s_i\|$`POINTS=`$m_i]$
`LOOPTYPE=EXPRANGE`	`START=`$r_{i,0}$ `END=`r_{i,m_i-1} $[$`STEP=`$e_i\|$`POINTS=`$m_i]$

Table 1: Basic syntactic grammar for the i-th line of the parameter file. $r_{i,j}$ is the j-th value of the set P_i, s_i is a linear step increment while e_i defines order of magnitude factor to separate points in the `EXPRANGE`.

- `FUNCTION`. Any valid perl function or concatenation of functions which accept an argument are candidates to transform the j-th element in the set P_i according to $r_{i,j} \rightarrow f_i(r_{i,j})$.

A summary of possible option values for the `FUNCTION` key as well as the syntax is given in Table 2.

3.3 Wildcarding

Once the parsing of the file is finished, the tool has obtained sufficient information to create the Cartesian product given by Eq. (3). A nice feature which allows parameter substitution and job template index substitution is wildcarding. The pattern for substitution has the following syntax:

- `${1}`, `${2}`,..., `${n}`. These tags are replaced during evaluation by the corresponding values $r_1, r_2, \ldots, r_n$. Note that due to the strict increasing order of replacement, no replacement for `${i}` is performed at lower set declarations from P_1 to P_{i-1}.

- `${JT_ID}`. It is replaced by the index ordering the whole set P. If $m > 10$, so many leading zeroes are included as necessary making file matching and ordering easier.

4 Setup

The Grid[Way] Job Template Manager is a stand-alone perl script, which includes all necessary subroutines for execution.

4.1 Installation

A simple download of the program into the current working directory is enough condition to have it installed.

Key=Value	Description
SKIP=$r_{i,j}$	
FUNCTION=$f_i =$	
abs	absolute value function
atan2	arctangent in the range -π to π
cos	cosine function
exp	raise to a power
hex	convert a string to a hexadecimal number
int	get the integer portion of a number
log	retrieve the natural logarithm for a number
oct	convert a string to an octal number
rand	retrieve the next pseudo-random number
sin	return the sine of a number
sqrt	square root function
srand	seed the random number generator
chomp	remove a trailing record separator from a string
chop	remove the last character from a string
chr	get character this number represents
crypt	one-way passwd-style encryption
hex	convert a string to a hexadecimal number
lc	return lower-case version of a string
lcfirst	return a string with just the next letter in lower case
length	return the number of bytes in a string
oct	convert a string to an octal number
ord	find a character's numeric representation
reverse	flip a string or a list
uc	return upper-case version of a string
ucfirst	return a string with just the next letter in upper case

Table 2: Optional tags available for modifying the standard composition of elements in each set.

If the GridWay Metascheduler is missing, only the create and delete operations will be available for use. Instructions to install the Metascheduler are available elsewhere[8] and lay outside the scope of the paper.

4.2   Configuration

There are two mechanisms to configure the behaviour of the program.

- In-line config option. This way allows the user a prompt modification of

Parameter	Default Value
job_template_wildcard	'${JT_ID}'
job_template_prefix	''
job_template_suffix	'.jt'
std_output_dir	'.'
std_error_dir	'.'
input_file_default_suffix	'.in'
comment_char	'#'
keyassignment_char	'='
separation_char	','
separation_char_cli	' '
separation_char_filename	'_'
jt_id_to_arg_separation	'_'
unix_operators	'&|<>;()`'
gridway_submit	'gwsubmit'
gridway_ps	'gwps'
gridway_kill	'gwkill'
gridway_wait	'gwwait'
gridway_dir_var	''
use_bignum	0
huge_number_points	10000
inode_size_kB	4

Table 3: Get_config_keywords parameter available. Strings are enclosed by single ticks.

the configuration parameter through the `--config PARAMETER=VALUE` command line option.

- Editing the program file. Alternatively, the program file contains all the configuration options ordered in two subroutines Get_config_parameters and Get_config_template_keywords at the end of the program.

The whole set of possible parameters to be assigned are given in Tables 3−4

5 Usage

The tool has been designed to be used through the command line interface of the users' terminal. It suffices to invoke the main command together with a subcommand which specifies the action to be performed and arguments to set the scope of the operation. Typed at the prompt, it reads:

Parameter	Default Value
Template_executable	'EXECUTABLE'
Template_arguments	'ARGUMENTS'
Template_stdout_file	'STDOUT_FILE'
Template_stderr_file	'STDERR_FILE'
Template_job_name	'NAME'
Template_encloser_char	''
Template_end_of_line	''

Table 4: Get_config_template_keywords. Strings are enclosed by single ticks.

```
gw_job_template_manager [OPTION] SUBCOMMAND ARG(S)
```

where some options for debug and configure also exists. All the elements needed for a correct use are summarized in Table 5.

Firstly, the user would edit a parameter file containing the instruction to generate the arguments wishing to pass as arguments to the executable. These job templates include five definitions:

- EXECUTABLE = executable filename (i.e. WORKER_FILE),

- ARGUMENTS = argument list, separated by whitespaces,

- STDOUT_FILE = standard output destination file,

- STDERR_FILE = standard error destination file,

- NAME = label for naming the job.

These definitions correspond to the GridWay Job Template Language and can be extended by adding the TEMPLATE_FILE with extra ones or modified through the config option.

After a successful run of the program, a zero exit status is delivered. Otherwise, it indicates an error, probably associated with either internal code errors or wrong syntax. Because there is no systematic way to represent program exit codes, an indication of the encountered problem is given in Table 6.

6 Examples

A simple collection of examples will illustrate the usage of the tool and some tips enhancing the possibilities of writing job templates.

6.1 Hello world!

Sticking to tradition, the first example must create a job template whose job is submitted and prints "Hello world!". Doing this involves at least two executions of the Grid[Way] Job Template Manager. Step by step:

1. A file (say `parameter.in`) describes which is the string argument to be printed. One valid line suffices in this case:

   ```
   $ cat parameter.in
   LOOPTYPE=LIST, VALUE="Hello world!"
   ```

2. Let us run the command[3] with the creation of job templates (`--create`) and an executable of the system (`--worker`):

   ```
   $ gw_job_template_manager --create parameter.in --worker /bin/echo
   Composed 1 job templates
   ```

   Once finished, the job template appears in the current directory:

   ```
   $ ls
   0_echo_Hello_world!.jt parameter.in
   ```

   with a name indicating the job template index (`0`), the job executable (`echo`) without trailing path, the sampled argument (`Hello World!`) where the whitespaces have been substituted by underscores, and a suffix extension (`.jt`). It contains five fields with instructions for Grid-Way:

   ```
   $ cat 0_echo_Hello_world!.jt
   NAME = 0_echo
   EXECUTABLE = /bin/echo
   ARGUMENTS = "Hello world!"
   STDOUT_FILE = 0_echo_Hello_world!.out
   STDERR_FILE = 0_echo_Hello_world!.err
   ```

3. The submission step is quite straightforward, since the tag `all` covers the submission of all generated job templates:

   ```
   $ gw_job_template_manager --submit all
   Submitted 1 jobs from templates
   ```

   depending on the scheduling, and resource availability, it will take more or less time for the remote job to finish. When it is done, the standard output file will be there containing the expected greeting.

   ```
   $ cat  0_echo_Hello_world!.out
   Hello world!
   ```

---

[3]Note that the command has been preceeded by a `$` symbol resembling the shell prompt.

6.2 Interplanetary Hello world!

As a second non-trivial example, we present an extension of the previous case where the ability to perform a two-dimensional product of sets is shown.

1. The file `parameter.in` has two lines:

   ```
   LOOPTYPE=LIST, VALUE=hello, VALUE=goodbye, FUNCTION=ucfirst
   LOOPTYPE=LIST, VALUE=world!, VALUE=mars!
   ```

   where two new aspects appear. First, we have changed our strategy and instead of echoing a single argument passed as a quoted string, we have given two arguments to the `/bin/echo` worker application. The first of them is composed by a combination of the set $P_1 =$ {hello, goodbye} and the second belongs to the set $P_2 =$ {world!, mars!}. Secondly, there is a transformation according to the perl function `ucfirst` which capitalizes the first letter of the word in P_1.

2. Creating the job templates generates the following files:

   ```
   $ ls -1
   0_echo_Hello_world!.jt
   1_echo_Hello_mars!.jt
   2_echo_Goodbye_world!.jt
   3_echo_Goodbye_mars!.jt
   parameter.in
   ```

   with equivalent content as shown in the simple "Hello world!" example.

3. After the submission of all the templates, it is very useful to monitor the status of the jobs. This action can be performed with the `--info` subcommand

   ```
   $ gw_job_template_manager --info now
   JOB_NAME,LOCALTIME,TIME,MANAGER,STATUS,QUEUE_NAME,HOST_NAME,EXIT_STATUS
   0_echo,Tue Feb 23 19:24:45 2010,1266949485,DISPATCH,WRAPPER,prod,egee.srce.hr,
   1_echo,Tue Feb 23 19:25:05 2010,1266949505,DISPATCH,WRAPPER,gilda,grid.acad.bg,
   2_echo,Tue Feb 23 19:24:54 2010,1266949494,EXECUTION,ACTIVE,prod,egee.srce.hr,
   3_echo,Tue Feb 23 19:25:04 2010,1266949504,DISPATCH,DONE,prod,egee.srce.hr,0
   ```

   which outputs a CSV-formatted[4] table with information about each job. This info subcommand is intended to provide knowledge about the set of jobs of the parameter study, leaving aside those jobs which do not match a parent job template in the current directory.

4. Once the jobs are successfully finished, it is possible to purge them from the list

---

[4]Comma separated values.

```
$ gw_job_template_manager --purge successful
Purged 4 jobs from templates
```

and also to get rid of all the job templates

```
$ gw_job_template_manager --delete all
Deleted 4 job templates
```

being in the current directory. Nevertheless, there is some limitation here; the [un]submitted, [un]finished, and [un]successful tags only make sense while matching their respectives jobs from templates listed through the gwps GridWay command. This means that if the user purges the successful jobs, only the unsuccessful remain in the list being therefore impossible to pursue more actions (like delete) on a successful set.

5. The output of the jobs now resembles an interplanetary dialog.

```
$ cat *.out
Hello world!
Hello mars!
Goodbye world!
Goodbye mars!
```

6.3   Squaring and skipping

The next example tries to compute the square of the set $P = \{1,2,4,5\}$.

1. Instead of typing them by hand, a squaring executable which receives a single argument has been prepared (named square). The parameter file parameter.in has one line:

```
LOOPTYPE=RANGE, START=1, END=5, STEP=1, SKIP=3
```

saying it should fill the integer interval [1,5], and from them skip the value 3

```
$ gw_job_template_manager -c parameter.in -w square
Composed 4 job templates
$ ls -1
0_square_1.jt
1_square_2.jt
2_square_4.jt
3_square_5.jt
parameter.in
square
```

Note that this time, the shortcuts -c and -w have been use instead of --create and --worker.

2. Job templates do not need to be submitted all at a time. For instance, the user could perform a first operation for jobs 0 to 1 and, later on, a second operation with the rest.

```
$ gw_job_template_manager --submit 0-1
Submitted 2 jobs from templates
$ gw_job_template_manager --submit unsubmitted
Submitted 2 jobs from templates
```

3. Another way to get to know what happened to each job is to use the history tag with the information command:

```
$ gw_job_template_manager --i history
JOB_NAME,LOCALTIME,TIME,MANAGER,STATUS,QUEUE_NAME,HOST_NAME,EXIT_STATUS
0_square,Wed Feb 24 12:33:35 2010,1267011215,DISPATCH,PENDING,,,
0_square,Wed Feb 24 12:33:38 2010,1267011218,DISPATCH,PROLOG,,,
0_square,Wed Feb 24 12:33:39 2010,1267011219,DISPATCH,WRAPPER,prod,egee.srce.hr,
0_square,Wed Feb 24 12:33:39 2010,1267011219,EXECUTION,FAILED,prod,egee.srce.hr,
0_square,Wed Feb 24 12:33:39 2010,1267011219,DISPATCH,EPILOG_FAIL,prod,egee.srce.hr,
0_square,Wed Feb 24 12:33:39 2010,1267011219,DISPATCH,PENDING,,,
0_square,Wed Feb 24 12:33:48 2010,1267011228,DISPATCH,PROLOG,,,
0_square,Wed Feb 24 12:33:48 2010,1267011228,DISPATCH,WRAPPER,gilda,grid.acad.bg,
0_square,Wed Feb 24 12:33:53 2010,1267011233,EXECUTION,FAILED,gilda,grid.acad.bg,
0_square,Wed Feb 24 12:33:53 2010,1267011233,DISPATCH,EPILOG_FAIL,gilda,grid.acad.bg,
0_square,Wed Feb 24 12:33:53 2010,1267011233,DISPATCH,PENDING,,,
0_square,Wed Feb 24 12:33:58 2010,1267011238,DISPATCH,PROLOG,,,
0_square,Wed Feb 24 12:34:33 2010,1267011273,DISPATCH,WRAPPER,default,gridway.org,
0_square,Wed Feb 24 12:34:33 2010,1267011273,EXECUTION,PENDING,default,gridway.org,
0_square,Wed Feb 24 12:34:38 2010,1267011278,EXECUTION,ACTIVE,default,gridway.org,
0_square,Wed Feb 24 12:34:39 2010,1267011279,EXECUTION,DONE,default,gridway.org,
0_square,Wed Feb 24 12:34:39 2010,1267011279,DISPATCH,EPILOG_STD,default,gridway.org,
0_square,Wed Feb 24 12:34:48 2010,1267011288,DISPATCH,EPILOG,default,gridway.org,
0_square,Wed Feb 24 12:34:57 2010,1267011297,DISPATCH,DONE,default,gridway.org,0
1_square,Wed Feb 24 12:33:35 2010,1267011215,DISPATCH,PENDING,,,
1_square,Wed Feb 24 12:33:39 2010,1267011219,DISPATCH,PROLOG,,,
1_square,Wed Feb 24 12:33:39 2010,1267011219,DISPATCH,WRAPPER,prod,egee.srce.hr,
1_square,Wed Feb 24 12:33:39 2010,1267011219,EXECUTION,FAILED,prod,egee.srce.hr,
1_square,Wed Feb 24 12:33:39 2010,1267011219,DISPATCH,EPILOG_FAIL,prod,egee.srce.hr,
1_square,Wed Feb 24 12:33:39 2010,1267011219,DISPATCH,PENDING,,,
1_square,Wed Feb 24 12:33:48 2010,1267011228,DISPATCH,PROLOG,,,
1_square,Wed Feb 24 12:33:48 2010,1267011228,DISPATCH,WRAPPER,gilda,grid.acad.bg,
1_square,Wed Feb 24 12:33:53 2010,1267011233,EXECUTION,FAILED,gilda,grid.acad.bg,
1_square,Wed Feb 24 12:33:53 2010,1267011233,DISPATCH,EPILOG_FAIL,gilda,grid.acad.bg,
1_square,Wed Feb 24 12:33:53 2010,1267011233,DISPATCH,PENDING,,,
1_square,Wed Feb 24 12:33:58 2010,1267011238,DISPATCH,PROLOG,,,
1_square,Wed Feb 24 12:34:33 2010,1267011273,DISPATCH,WRAPPER,default,gridway.org,
1_square,Wed Feb 24 12:34:33 2010,1267011273,EXECUTION,PENDING,default,gridway.org,
1_square,Wed Feb 24 12:34:38 2010,1267011278,EXECUTION,ACTIVE,default,gridway.org,
1_square,Wed Feb 24 12:34:39 2010,1267011279,EXECUTION,DONE,default,gridway.org,
1_square,Wed Feb 24 12:34:39 2010,1267011279,DISPATCH,EPILOG_STD,default,gridway.org,
1_square,Wed Feb 24 12:34:49 2010,1267011289,DISPATCH,EPILOG,default,gridway.org,
1_square,Wed Feb 24 12:34:58 2010,1267011298,DISPATCH,DONE,default,gridway.org,0
2_square,Wed Feb 24 12:34:01 2010,1267011241,DISPATCH,PENDING,,,
2_square,Wed Feb 24 12:34:08 2010,1267011248,DISPATCH,PROLOG,,,
2_square,Wed Feb 24 12:34:24 2010,1267011264,DISPATCH,WRAPPER,default,gridway.org,
2_square,Wed Feb 24 12:34:24 2010,1267011264,EXECUTION,PENDING,default,gridway.org,
2_square,Wed Feb 24 12:34:33 2010,1267011273,EXECUTION,ACTIVE,default,gridway.org,
2_square,Wed Feb 24 12:34:34 2010,1267011274,EXECUTION,DONE,default,gridway.org,
```

```
2_square,Wed Feb 24 12:34:34 2010,1267011274,DISPATCH,EPILOG_STD,default,gridway.org,
2_square,Wed Feb 24 12:34:41 2010,1267011281,DISPATCH,EPILOG,default,gridway.org,
2_square,Wed Feb 24 12:34:56 2010,1267011296,DISPATCH,DONE,default,gridway.org,0
3_square,Wed Feb 24 12:34:01 2010,1267011241,DISPATCH,PENDING,,,
3_square,Wed Feb 24 12:34:08 2010,1267011248,DISPATCH,PROLOG,,,
3_square,Wed Feb 24 12:34:22 2010,1267011262,DISPATCH,WRAPPER,default,gridway.org,
3_square,Wed Feb 24 12:34:24 2010,1267011264,EXECUTION,PENDING,default,gridway.org,
3_square,Wed Feb 24 12:34:34 2010,1267011274,EXECUTION,ACTIVE,default,gridway.org,
3_square,Wed Feb 24 12:34:34 2010,1267011274,EXECUTION,DONE,default,gridway.org,
3_square,Wed Feb 24 12:34:34 2010,1267011274,DISPATCH,EPILOG_STD,default,gridway.org,
3_square,Wed Feb 24 12:34:41 2010,1267011281,DISPATCH,EPILOG,default,gridway.org,
3_square,Wed Feb 24 12:34:55 2010,1267011295,DISPATCH,DONE,default,gridway.org,0
```

where the example shows how jobs 0 and 1 failed after trying to be
run without permissions at different resources and ended up in another
resource where jobs were allowed to run. The GridWay Metascheduler
learned from that and managed jobs 2 and 3 more efficiently.

4. The output shows a properly squared set

```
$ cat  *.out
1^2=1
2^2=4
4^2=16
5^2=25
```

6.4   Random Monte-Carlo

Often, programs involving large statistical runs require a proper initiation
of the random number generator. This can be done by means of a seed
and is a key point for ensuring uncorrelated jobs. Here a simple way of
getting different seeds in the integer interval [0,1000) is implemented for
eight independent jobs.

1. The `parameter.in` looks like:

```
$ cat parameter.in
LOOPTYPE=RANGE, START=1000, END=1000, POINTS=8, \
FUNCTION=int rand
```

and note that evaluating eight jobs within 1000 and 1000 makes an
apparently dummy set $\{1000, \ldots, 1000\}$, but the transformation under
the function composition `int rand` generates eight random numbers
and throws away the decimal part.

2. A possible output of the creation

```
$ gw_job_template_manager -w worker -c parameter.in
Composed 8 job templates
```

16

is

```
$ ls -1
0_worker_292.jt
1_worker_741.jt
2_worker_468.jt
3_worker_481.jt
4_worker_159.jt
5_worker_310.jt
6_worker_555.jt
7_worker_645.jt
parameter.in
worker
```

6.5 Advanced wildcarding

Another common way to produce results is writing to an output file instead of to the standard output. In such a case, let us consider a worker executable comparing a Taylor expansion and an exact result for several orders of magnitude. It takes three arguments, being the last one a string containing the file name in which results are to be written.

1. The parameter file could be

   ```
   $ cat parameter.in
   LOOPTYPE=LIST, VALUE=Taylor, VALUE=Exact
   LOOPTYPE=EXPRANGE, START=1, END=1E3, STEP=1, SKIP=100
   LOOPTYPE=LIST, VALUE=${JT_ID}.txt
   ```

   with three features not yet seen: the EXPRANGE type which ranges exponentially, the scientific notation 1E3 and a last line, indicating that part of the third argument is going to be translated into the job template index $JT_ID for each job.

2. The creation of templates is this time slightly different due to the retrieval of extra output files. For this to be taken into account, an extra job template input file is needed template.in:

   ```
   $ cat template.in
   OUTPUT_FILES=${3}
   $ gw_job_template_manager -w worker -c parameter.in -t worker.in
   Composed 6 job templates
   ```

   and the result

   ```
   $ ls -1
   0_worker_Taylor_1_0.txt.jt
   1_worker_Taylor_10_1.txt.jt
   2_worker_Taylor_1000_2.txt.jt
   3_worker_Exact_1_3.txt.jt
   ```

```
4_worker_Exact_10_4.txt.jt
5_worker_Exact_1000_5.txt.jt
parameter.in
worker
template.in
$ cat 5_worker_Exact_1000_5.txt.jt
NAME = 5_worker
EXECUTABLE = worker
ARGUMENTS = Exact 1000 5.txt
STDOUT_FILE = 5_worker_Exact_1000_5.txt.out
STDERR_FILE = 5_worker_Exact_1000_5.txt.err
OUTPUT_FILES=5.txt
```

3. The submission does not require any difference, but this time we want
   to know how much do the jobs take to finish:

```
$ gw_job_template_manager -s all; time gw_job_template_manager -p all
Submitted 6 jobs from templates
Purged 6 jobs from templates

real    1m51.861s
user    0m0.660s
sys     0m0.060s
```

4. In this case, the relevant results are the text files

```
$ ls -1 *.txt
0.txt
1.txt
2.txt
3.txt
4.txt
5.txt
```

6.6 Big numbers

What if the input are very large numbers? Next illustrates how to proceed

1. Let consider the case in which the parameter file is

```
$ cat parameter.in
LOOPTYPE=RANGE,\
START=12345678901234567891123456789212345678931234567894 1,\
  END=12345678901234567891123456789212345678931234567894 3,\
POINTS=3
```

2. The standard tentative will **not** work[5]

---

[5]Depending on the perl version, for perl 5.10 it does not work, but earlier versions (5.8)
did not yet transform automatically into scientific notation.

18

```
$ gw_job_template_manager -w /bin/echo -c parameter.in
Composed 3 job templates
$ ls -1
0_echo_1.23456789012346e+50.jt
1_echo_1.23456789012346e+50.jt
2_echo_1.23456789012346e+50.jt
parameter.in
```

because the transformation into scientific notation dropped out the precision we expect to work with.

3. Instead, the user should instruct the tool to profit from the `bignum` perl package

```
$ gw_job_template_manager --delete all
Deleted 3 job templates
$ gw_job_template_manager --config use_bignum=1\
  -w /bin/echo -c parameter.in
Composed 3 job templates
$ ls -1
0_echo_12345678901234567891123456789212345678931234567894 1.jt
1_echo_12345678901234567891123456789212345678931234567894 2.jt
2_echo_12345678901234567891123456789212345678931234567894 3.jt
parameter.in
```

so obtaining the desired job templates.

6.7   Using gLite Job Description Language

When using the middleware gLite, the user will probably find a user interface without the GridWay Metascheduler installed. In this example a method to translate the job templates into JDL and how to submit them is described.

1. Four simple tests are prepared. They give a hint about the working environment:

```
$ cat parameter.in
LOOPTYPE=LIST, VALUE=ps, VALUE=pwd, VALUE=ls, VALUE=whoami
$ cat template.in
OutputSandbox = {"${JT_ID}_env_${1}.out","${JT_ID}_env_${1}.err"};
```

2. A way to set up the propper grammar in the template file is to change the keys and some formating elements through config options:

```
$ gw_job_template_manager \
-w /usr/bin/env -c parameter.in -t template.in \
--config Template_executable=Executable \
--config Template_arguments=Arguments \
--config Template_stdout_file=StdOutput \
--config Template_stderr_file=StdError \
--config Template_job_name=JobName \
```

```
--config Template_encloser_char=\" \
--config Template_end_of_line=\; \
--config job_template_suffix=.jdl
WARNING: GridWay location not set up.
This means that the usability of this tool is limited to create and delete
job templates. Please identify your $GW_LOCATION directory and set the
parameter to that value with "--config GW_LOCATION=value".
Composed 4 job templates
```

3. The Grid[Way] Job Template Manager does not know how to submit templates for other middleware, but the user can still tweak the settings through the `--config` modifier and get the submission procedure done:

```
$ gw_job_template_manager -s all \
--config gridway_submit=glite-wms-job-submit \
--config gridway_submit_flag=-a \
--config job_template_suffix=.jdl
WARNING: GridWay location not set up.
This means that the usability of this tool is limited to create and delete
job templates. Please identify your $GW_LOCATION directory and set the
parameter to that value with "--config GW_LOCATION=value".

Connecting to the service https://gilda-rb.rediris.es:7443/glite_wms_wmproxy_server


===================== glite-wms-job-submit Success =====================

The job has been successfully submitted to the WMProxy
Your job identifier is:

https://gilda-rb.rediris.es:9000/9xFbq28QPgTcYLxCeVU5Kw

========================================================================


Connecting to the service https://gilda-rb.rediris.es:7443/glite_wms_wmproxy_server


===================== glite-wms-job-submit Success =====================

The job has been successfully submitted to the WMProxy
Your job identifier is:

https://gilda-rb.rediris.es:9000/IP4LKZgkpxcL9NE58AHuPQ

========================================================================


Connecting to the service https://gilda-rb.rediris.es:7443/glite_wms_wmproxy_server


===================== glite-wms-job-submit Success =====================

The job has been successfully submitted to the WMProxy
Your job identifier is:

https://gilda-rb.rediris.es:9000/3fyT-LnV3xkimEPfSsAHag
```

```
==========================================================================


Connecting to the service https://gilda-rb.rediris.es:7443/glite_wms_wmproxy_server


===================== glite-wms-job-submit Success =====================

The job has been successfully submitted to the WMProxy
Your job identifier is:

https://gilda-rb.rediris.es:9000/53HM4HRBCsqwHwpa01zEVw

==========================================================================


Submitted 4 jobs from templates
```

4. Finally, by picking up the job identifiers of the output, a final recovery of data is straightforward but not possible with the help of the tool. For example, to retrieve the output of the last job:

```
$ glite-wms-job-output https://gilda-rb.rediris.es:9000/IP4LKZgkpxcL9NE58AHuPQ
Connecting to the service https://gilda-rb.rediris.es:7443/glite_wms_wmproxy_server


================================================================================

JOB GET OUTPUT OUTCOME

Output sandbox files for the job:
https://gilda-rb.rediris.es:9000/53HM4HRBCsqwHwpa01zEVw
have been successfully retrieved and stored in the directory:
/tmp/jobOutput/user_53HM4HRBCsqwHwpa01zEVw

================================================================================
$ cd /tmp/jobOutput/user_53HM4HRBCsqwHwpa01zEVw
$ ls -1
3_env_whoami.err
3_env_whoami.out
$ cat 3_env_whoami.out
gilda075
```

7   Conclusion

A tool for managing a parameter sweep study on a distributed architecture has been presented, with special emphasis on the mechanism to create job templates from a given parameter file. The syntax of the parameter file is simple though quite powerful, specially because broad scoped features (arbitrary dimensions, exponentiation, value skipping, wildcarding and functions) have been implemented.

The usage has been designed to perform an action and the anti-action, so to the creation of job templates a deletion exists. Moreover, submission of jobs is also complemented by the purge. In any case, all the operations are grouped under useful subcommands which can be accessed through the command line interface.

In order to allow enough flexibility, the user has at disposal configuration options to modify program parameters. Some examples have been worked out for illustrating how a little amount of input give rise to a plethora of possibilities, from the "Hello world!" trivial case up to a parameter study controlled with the GridWay Metascheduler.

Within the aspects which could be desirable but not yet implemented, it can be mentioned the capability to handle job templates in the XML-formatted Job Standard Description Language. The problem to accept this format is the inherent hierarchy of the XML syntax, which is incompatible with appending a template file with extra information. Nevertheless, an smart merge might be implemented under certain scheme. Another kind of limitation is the use of wildcards and functions altogether, or the need of other wildcards. The inclusion of these features could be studied if they turn to handicap parameter sweep studies from the scientific community.

The tool can be viewed an extension of GridWay, which performs the resource discovering, scheduling or workloading for the jobs. Being modular, it is much easier to focus on the aspects regarding parameter sweeps and the programming protocol to interact with the middleware. It has been shown also the possibilities to interact with other submission agents, such as the gLite WMS.

Altogether, this tool provides the user a powerful mechanism to port applications to clusters, grid or cloud. Our experience on this subject indicates that the steep learning curve and lack of success are the main drawbacks for adopting the solutions of distributed computing. Offering an opportunity to save time on the generation of the parameters to be swept and to ensure consistent operations on specific blocks of jobs is clearly an step forward to enhance the overall performance of the application at the user's convenience.

Acknowledgements

References

References

[1] E. Huedo, R. S. Montero, I. M. Llorente.
The GridWay Framework for Adaptive Scheduling and Execution on Grids.
Scalable Computing, Practice and Experience 6 (3) 2005, pp 1-8.

[2] D. Abramson, J. Giddy, L. Kotler.
High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?.
International Parallel and Distributed Processing Symposium (IPDPS), Cancun, Mexico. May 2000. pp. 520-528

[3] H. Casanova, F. Berman.
Parameter Sweeps on the Grid with APST.
Grid Computing: Making the Global Infrastructure a Reality, 2003, pp. 773-787.

[4] F. Berman, R. Wolski, H. Casanova et al.
Adaptive Computing on the Grid Using AppLeS.
IEEE Transactions on Parallel and Distributed Systems, pp. 369-382, April, 2003.

[5] E. Huedo, R. S. Montero, I. M. Llorente.
Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications.
Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'04), 2004. pp. 28-33.

[6] GLite Job Description Language.
Workload Management System User and Reference Guide.

`https://edms.cern.ch/document/572489/1`,
Attributes Specification.
`https://edms.cern.ch/document/555796/1JDL`

[7] M. Drescher, A. Anjomshoaa, G. Williams, D. Meredith.
JSDL Parameter Sweep Job Extension.
`http://www.ogf.org/documents/GFD.149.pdf`

[8] *GridWay Installation Guide.*
`http://gridway.org/doku.php?id=documentation:release_5.6:ig`

[9] Grid INFN Laboratory for Dissemination Activities.
`https://gilda.ct.infn.it/vo.html`

[10] EGEE: Enabling Grids for E-sciencE.
`http://www.eu-egee.org`

Subcommand	Description
`-c, --create`	create templates
`-d, --delete`	delete templates
`-s, --submit`	submit the jobs from templates
`-p, --purge`	purge the existing jobs from templates
`-k, --kill`	kill the existing jobs from templates
`-i, --info`	information about status of the submitted jobs
`-v, --version`	version number of the program
`-l, --license`	credits and license
`-h, --help`	print help

Option	Description
`-w, --worker`	add worker file
`-t, --template`	add template file
`--signal`	add signal for kill
`--debug`	show debugging information
`--config key=value`	assign a key=value pair for configuration settings

Syntax
`-c <PARAMETER_FILE> [-w WORKER_FILE] [-t TEMPLATE_FILE]`
`-d <all|[un]submitted|[un]finished|[un]successful|FROM-TO>`
`-s <all|[un]submitted|[un]finished|[un]successful|FROM-TO>`
`-p <all|[un]finished|[un]successful|FROM-TO>`
`-k <all|[un]finished|[un]successful|FROM-TO> [--signal SIG]`
`-i <history|now|evolution>`

Argument	Description
`PARAMETER_FILE`	file with parameter description
`WORKER_FILE`	executable file to be run remotely
`TEMPLATE_FILE`	append extra variables from template file
`all`	all the templates are subcommanded
`[un]submitted`	only those which were [not] submitted
`[un]finished`	only those whose jobs have [not] finished
`[un]successful`	only those whose jobs finished [un]succesfully
`FROM-TO`	deletes the range [FROM,TO]
`SIG`	kill with signal passed to gwkill
`history`	full historic information about each job
`now`	last status update for each job
`evolution`	timely snapshots of job statuses

Table 5: Different subcommands, syntax, arguments and options for the command line interface. Square brackets indicate optional features whilst angle brackets are compulsory elements.

Exit status	Description
0	success
1	wrong syntax in the command line
2	system execution error
3	file not found or requirement not matched
4	parameter file syntax inconsistent
5	error opening file
6	error closing file
7	no job found coming from template in the list
8	internal computation error
9	internal parsing error

Table 6: Success and error code description from exit status.