

Table of Contents

About DRM4G	2
Start Guide	2
Start to run DRM4G	2
My first job	3
How to configure a TORQUE/PBS resource	4
User Scenarios	5
Single Job	5
Array Job	5
MPI Job	6

About DRM4G

DRM4G is an open platform, based on [?GridWay](#) , to define, submit, and manage computational jobs. DRM4G is a [?Python \(2.6+, 3.3+\)](#) implementation that provides a single point of control for computing resources without installing any intermediate middlewares. As a result, a user is able to run the same job on laptops, desktops, workstations, clusters, supercomputers, and any grid.

Start Guide

In order to install DRM4G, download the installation script :

```
[user@mycomputer~]$ wget --no-check-certificate https://meteo.unican.es/work/DRM4G/drm4g_bootstrap.sh
```

And then run it :

```
[user@mycomputer~]$ bash ./drm4g_bootstrap.sh --dir $HOME

=====
DRM4G installation script
=====

--> Checking the last version of DRM4G ...

--> DRM4G version selected: 2.4.1

--> Downloading drm4g-2.4.1.tar.gz ...

--> Unpacking drm4g-2.4.1.tar.gz in directory /home/user ...

--> Installing DRM4G python requirements locally ...

=====
Installation of DRM4G 2.4.1 is done!
=====

In order to work with DRM4G you have to enable its
environment with the command:

    source /home/user/drm4g/bin/drm4g_init.sh

You need to run the above command on every new shell you
open before using DRM4G, but just once per session.
```

Start to run DRM4G

If the the directory `~/ .drm4g` does not exist, `drm4g` will create one with a local configuration

1. Enable DRM4G:

```
[user@mycomputer~]$ source $HOME/drm4g/bin/drm4g_init.sh
```

2. Start up DRM4G :

```
[user@mycomputer~]$ drm4g start
Checking DRM4G local configuration ...
Creating a DRM4G local configuration in '/home/user/.drm4g'
Copying from '/home/user/drm4g/etc' to '/home/user/.drm4g/etc'
Starting DRM4G ....
OK
```

```
Starting ssh-agent ...
OK
```

3. Show information about all available resources, their hosts and their queues :

```
[user@mycomputer~]$ drm4g resource list
RESOURCE      STATE
localmachine  enabled
```

```
[user@mycomputer~]$ drm4g host list
HID ARCH      JOBS(R/T) LRMS      HOST
0  x86_64      0/0 fork   localmachine
```

```
[user@mycomputer~]$ drm4g host list 0
HID ARCH      JOBS(R/T) LRMS      HOST
0  x86_64      0/0 fork   localmachine

QUEUENAME     JOBS(R/T) WALLT CPUT  MAXR  MAXQ
default       0/0 0    0    1    1
```

My first job

1. Create a job template :

```
[user@mycomputer~]$ echo "EXECUTABLE=/bin/date" > date.job
```

2. Submit the job :

```
[user@mycomputer~]$ drm4g job submit date.job
ID: 0
```

3. Check the evolution of the job :

```
[user@mycomputer~]$ drm4g job list 0
JID DM  EM  START  END      EXEC  XFER  EXIT NAME  HOST
0  pend ---- 19:39:09 --:--:-- 0:00:00 0:00:00 --  date.job  --
```

If you execute successive `drm4g job list 0`, you will see the different states of this job:

`pend`: The job is pending for a host to run on.

```
JID DM  EM  START  END      EXEC  XFER  EXIT NAME  HOST
0  pend ---- 19:39:09 --:--:-- 0:00:00 0:00:00 --  date.job  --
```

`prol`: The frontend is being prepared for execution.

```
JID DM  EM  START  END      EXEC  XFER  EXIT NAME  HOST
0  prol ---- 19:39:09 --:--:-- 0:00:00 0:00:00 --  date.job  --
```

`wrap pend`: The job has been successfully submitted to the frontend and it is pending in the queue

```
JID DM  EM  START  END      EXEC  XFER  EXIT NAME  HOST
0  wrap pend 19:39:09 --:--:-- 0:00:00 0:00:00 --  date.job localhost/fork
```

`wrap actv`: The job is running in the remote queue.

```
JID DM  EM  START  END      EXEC  XFER  EXIT NAME      HOST
0  wrap  actv  19:39:09  --:--:--  0:00:05  0:00:00  --  date.job localhost/fork
```

epil:The job is done/complete in queue and it's fetching the results.

```
JID DM  EM  START  END      EXEC  XFER  EXIT NAME      HOST
0  epil  ----  19:39:09  --:--:--  0:00:10  0:00:00  --  date.job localhost/fork
```

done:The job is done.

```
JID DM  EM  START  END      EXEC  XFER  EXIT NAME      HOST
0  done  ----  19:39:09  19:39:27  0:00:10  0:00:01  0  date.job localhost/fork
```

4. In this job template, the results from the job are standard output (stdout) and standard error (stderr), both files will be in the same directory of the job submission:

```
[user@mycomputer~]$ cat stdout.0
Mon Jul 28 12:29:43 CEST 2014

[user@mycomputer~]$ cat stderr.0
```

How to configure a TORQUE/PBS resource

DRM4G uses the following environmental variable `EDITOR` to select which editor is going to be used for configuring resources. By default the editor is `vi`

In order to configure a TORQUE/PBS cluster accessed through ssh protocol, you can follow the next steps:

1. Generate a public/private key pair without password :

```
[user@mycomputer~]$ ssh-keygen -t rsa -b 2048 -f $HOME/.ssh/meteo_rsa -N ""
```

2. Copy the new public key to the TORQUE/PBS resource :

```
[user@mycomputer~]$ ssh-copy-id -i $HOME/.ssh/meteo_rsa.pub user@ui.macc.unican.es
```

Configure the meteo resource :

```
[user@mycomputer~]$ drm4g resource edit
[DEFAULT]
enable           = true
communicator     = local
frontend        = localhost
lrms             = fork

[localmachine]
max_jobs_running = 1

[meteo]
enable           = true
communicator     = ssh
username        = user
frontend        = ui.macc.unican.es
private_key     = ~/.ssh/meteo_rsa
lrms            = pbs
queue           = grid
max_jobs_running = 1
max_jobs_in_queue = 2
```

4. List and check if resource has been created successfully :

```
[user@mycomputer~]$ drm4g resource list
RESOURCE          STATE
localmachine      enabled
meteo              enabled

[user@mycomputer~]$ drm4g host list
HID ARCH          JOBS(R/T) LRMS      HOST
0  x86_64          0/0 fork   localmachine
1  x86_64          0/0 pbs    meteo
```

That's it! Now, you can submit jobs to both resources.

User Scenarios

In this section it will be described how to take advantage of DRM4G to calculate the number Pi. To do that, three types of jobs **single**, **array** and **mpi** will be used.

Single Job

- C binary [pi_serial?](#)
- DRM4G job template :

```
EXECUTABLE = pi.sh
ARGUMENTS  = 0 1 100000000
STDOUT_FILE = stdout_file.${JOB_ID}
STDERR_FILE = stderr_file.${JOB_ID}
INPUT_FILES = pi_serial, pi.sh
```

- pi.sh script :

```
#!/bin/bash
chmod +x ./pi_serial
./pi_serial $@
```

Array Job

- C binary : [pi_serial?](#)
- DRM4G job template :

```
EXECUTABLE = pi.sh
ARGUMENTS  = ${TASK_ID} ${TOTAL_TASKS} 100000000
STDOUT_FILE = stdout_file.${TASK_ID}
STDERR_FILE = stderr_file.${TASK_ID}
INPUT_FILES = pi_serial.c, pi.sh
```

- pi.sh script :

```
#!/bin/bash
chmod +x ./pi_serial
./pi_serial $@
```

- Sum the results inside each file :

```
$ awk 'BEGIN {sum=0} {sum+=$1} END {printf "Pi is %0.12g\n", sum}' stdout_file.*
```

MPI Job

- C code :
- DRM4G job template :

```
EXECUTABLE    = mpi.sh
STDOUT_FILE   = stdout.${JOB_ID}
STDERR_FILE   = stderr.${JOB_ID}
INPUT_FILES   = mpi.sh, pi_parallel.c
NP            = 2
```

- mpi.sh script :

```
source /software/meteo/use/load_use
use openmpi4intel
chmod +x pi_parallel
mpirun -np 2 ./pi_parallel
```