Table of Contents

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, an installation script is provided. Type the command below on your shell terminal:

```
$ wget --no-check-certificate -O- https://meteo.unican.es/work/DRM4G/install.sh | bash

==========================
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:

   . /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.
```

By default, it will install DRM4G on your current directory. But, you can download the installation script:

```
$ wget --no-check-certificate https://meteo.unican.es/work/DRM4G/install.sh
```

And run it manually:

```
$ bash ./install.sh [options]
```

The options available are:

- -d, --dir DIRECTORY: Install DRM4G into a directory.
- -V, --version: Version to install.
- -h, --help: Print help text.

My first job

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

1. Enable DRM4G:

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

2. Start up DRM4G :

```
[user@mycomputer~]$ drm4g start
Creating a DRM4G local configuration in '/home/user/.drm4g'
Creating '/home/user/.drm4g/var/acct' directory
Coping from '/home/user/drm4g/etc' to '/home/user/.drm4g/etc'
Starting DRM4G ....
OK
Starting ssh-agent ...
OK
```

3. Show information about all available resources and their host :

```
[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
```

4. Create a job template :

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

5. Submit the job :

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

6. 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:

```
0   pend ---- 19:39:09 --:--:-- 0:00:00 0:00:00 --   date.job        --
0   prol ---- 19:39:09 --:--:-- 0:00:00 0:00:00 --   date.job        --
0   wrap pend 19:39:09 --:--:-- 0:00:00 0:00:00 --   date.job localhost/fork
0   wrap actv 19:39:09 --:--:-- 0:00:05 0:00:00 --   date.job localhost/fork
0   epil ---- 19:39:09 --:--:-- 0:00:10 0:00:00 --   date.job localhost/fork
0   done ---- 19:39:09 19:39:27 0:00:10 0:00:01 0    date.job localhost/fork
```

- `pend`: The job is waiting for a resource to run on.
- `prol` :The remote system is being prepared for execution.
- `wrap pend` :The job has been successfully submitted to the computing resource and it is waiting.
- `wrap actv`:The job is being executed by the computing resource.
- `epil`:The job is finalizing.

- `done`:The job has finished.

7. Results are standard output (stdout) and standard error (stderr), both files will be in the same directory of job template:

```
[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

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
[meteo]
enable            = true
communicator      = ssh
username          = user
frontend          = ui.macc.unican.es
private_key       = ~/.ssh/meteo_rsa
lrms              = pbs
queue             = qrid
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
meteo               enabled

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

That's it! Now, you can summit jobs to `meteo`.

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 Jobs

- C code :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main (int argc, char** args)
{
 int task_id;
 int total_tasks;
 long long int n;
 long long int i;

 double l_sum, x, h;

 task_id = atoi(args[1]);
 total_tasks = atoi(args[2]);
 n = atoll(args[3]);

 fprintf(stderr, "task_id=%d total_tasks=%d n=%lld\n", task_id, total_tasks, n);

 h = 1.0/n;

 l_sum = 0.0;

 for (i = task_id; i < n; i += total_tasks)
 {
   x = (i + 0.5)*h;
   l_sum += 4.0/(1.0 + x*x);
 }

 l_sum *= h;

 printf("%0.12g\n", l_sum);

 return 0;
}
```

Array Jobs

- C code :

```
  #include <stdio.h>
 #include <string.h>

 int main (int argc, char** args)
 {
   int task_id;
   int total_tasks;
   long long int n;
   long long int i;

   double l_sum, x, h;

   task_id = atoi(args[1]);
   total_tasks = atoi(args[2]);
   n = atoll(args[3]);

   fprintf(stderr, "task_id=%d total_tasks=%d n=%lld\n", task_id, total_tasks, n);

   h = 1.0/n;
```

```
    l_sum = 0.0;

    for (i = task_id; i < n; i += total_tasks)
    {
      x = (i + 0.5)*h;
      l_sum += 4.0/(1.0 + x*x);
    }

    l_sum *= h;

    printf("%0.12g\n", l_sum);

    return 0;
  }
```

- Job template :

```
EXECUTABLE   = pi
ARGUMENTS    = ${TASK_ID} ${TOTAL_TASKS} 100000
STDOUT_FILE = stdout_file.${TASK_ID}
STDERR_FILE = stderr_file.${TASK_ID}
```

MPI Jobs

- C code :

```
 #include "mpi.h"
#include <stdio.h>
#include <math.h>

int main( int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    printf("Process %d on %s\n", myid, processor_name);

    n = 100000000;

    startwtime = MPI_Wtime();

    h   = 1.0 / (double) n;
    sum = 0.0;
    for (i = myid + 1; i <= n; i += numprocs)
    {
        x = h * ((double)i - 0.5);
        sum += 4.0 / (1.0 + x*x);
    }
    mypi = h * sum;
```

```
    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (myid == 0)
    {
        printf("pi is approximately %.16f, Error is %.16f\n",
               pi, fabs(pi - PI25DT));
        endwtime = MPI_Wtime();
        printf("wall clock time = %f\n", endwtime-startwtime);
    }

    MPI_Finalize();

    return 0;
}
```

Job template :

```
EXECUTABLE     = mpi
STDOUT_FILE    = stdout.${JOB_ID}
STDERR_FILE    = stderr.${JOB_ID}
NP             = 2
```