The Mobyle Execution System

Mobyle 1.0

Contents

1 concepts

There are 3 main actors in the Mobyle execution system:

- The ExecutionSystem is the interface between Mobyle and the low level execution system as your local system or your favorite Distributed Resources Management System (DRMS),
- The ExecutionConfig is an object which provides the basic information needed by the ExecutionSystem to be used in your environment,
- The Dispatcher chooses which execution system to use for a job.

Each actor is modelized as a class and several implementations are provided in Mobyle distribution.

1.1 The ExecutionSystem

All execution System classes inherit from the abstract class ExecutionSystem and are located in the MOBYLE-HOME/Src/Mobyle/Execution package. 5 ExecutionSystem classes are available:

- SYS which is the interface between Mobyle and your local system.
- SGE which is the interface between Mobyle and the Sun Grid Engine DRMS.
- SgeDRMAA which is the interface between Mobyle and the Sun Grid Engine DRMS using the drmaa library.
- PbsDRMAA which is the interface between Mobyle and the PBS/torque DRMS using the drmaa library.
- Lsf DRMAA which is the interface between Mobyle and the LSF DRMS using the drmaa library.

For SGE and PBS, two execution systems are proposed: one which wraps the shell commands and another which deals with Distributed Resource Management Application Api (DRMAA) library. The benefits to use the DRMAA library is that there is no need to use intermediate shell to run, get the status or kill a job. For those who cannot or do not want to install libdrmaa, the legacy SGE execution systems is kept.

1.2 The ExecutionConfig

Mobyle is highly flexible towards the execution of jobs, you can use several Execution System in parallel. For instance you can use SGE for most of jobs and SYS for some very small jobs or one cluster managed by SGE for a set of jobs and an other cluster managed by PBS for the other jobs on so on. For each execution system you want to use you must define an ExecutionConfig. this instanciation must be done in EXECUTION_SYSTEM_ALIAS. To each class of ExecutionSystem a class of ExecutionConfig is associated. So the choice of an ExecutionConfig determines which ExecutionSystem will be used.

1.2.1 SgeDRMAAConfig

is associated to SgeDRMAA ExecutionSystem, which is the interface with SGE using libdrmaa. This class takes 3 mandatory parameters:

1. the path of the drmaa library (eg. /usr/local/sge/lib/lx26-amd64/libdrmaa.so)
2. root the content of the SGE_ROOT variable.
3. cell the content of the SGE_CELL variable.

1.2.2 PbsDRMAAConfig

is associated to PbsDRMAA ExecutionSystem, which is the interface with Pbs/torque using libdrmaa. This class takes 2 mandatory parameters:

1. the path of the drmaa library (eg. /usr/local/lib64/libdrmaa.so)
2. le fully qualified name of the host where is located the PBS/torque daemon server

1.2.3 LsfDRMAAConfig

is associated to LsfDRMAA ExecutionSystem, which is the interface with LSF using libdrmaa. This class takes 3 mandatory parameters:

1. the path of the drmaa library (eg. /usr/local/lib64/libdrmaa.so)
2. lsf_envdir the content of the variable ENVDIR of LSF
3. lsf_serverdir the content of the variable SERVERDIR of LSF

1.2.4 SGEConfig

is associated to SGE ExecutionSystem, which is the interface with SGE using the shell commands wrapping. This class takes 2 mandatory parameters:

1. root the content of the SGE_ROOT variable
2. cell the content of the SGE_CELL variable

1.2.5 SYSConfig

is associated to SYS ExecutionSystem. It is used to launch job without any DRMS. There is no argument to run a job in "local".

1.3 The Dispatcher

After having defined all your execution system, you have to specify what system must be used for a given program. This is the role of the dispatcher.
A DefaultDispatcher is provided.

1.3.1 The DefaultDispatcher

associates statically one program to an ExecutionSystem and a queue. The DefaultDispatcher takes as argument a dictionary where the name of the programs are the keys and the values are tuple with 2 arguments. the first one is an EXECUTION_SYSTEM_ALIAS entry and the second a queue name. There is a joker program name : "DEFAULT" to define a system for all programs which are not listed in the keys.

2 how to configure

2.1 First step: define the execution system you want to use.

The key is a symbolic name you give to an execution system.
The value is an instance of ExecutionConfig. following an example of EXECUTION_SYSTEM_ALIAS using all ExecutionConfig we provide.

```
from Execution import  *
EXECUTION_SYSTEM_ALIAS = {
          'DRMAA_sge'  : SgeDRMAAConfig( '/usr/local/sge/lib/lx26-amd64/libdrmaa.so' ,
                                         root ='/usr/local/sge',
                                         cell = 'default' ) ,
          'DRMAA_torque': PbsDRMAAConfig('/usr/local/lib64/libdrmaa.so' ,
                                         'marygay.sis.pasteur.fr'),

          'SGE'         : SGEConfig( root = '/usr/local/sge',
                                     cell= 'default' ) ,
          'SYS'        : SYSConfig()  ,
          'LSF'         : LsfDRMAAConfig( '/home/bneron/Sys/lib/libdrmaa.so' ,
                                                  lsf_envdir = '/home/bneron/Sys/share/ls
                                                  lsf_serverdir = '/home/bneron/Sys/share
                        }
          }
```

here a second example to illustrate that you can use the same Execution System with different Config.

```
from Execution import  *
EXECUTION_SYSTEM_ALIAS = {
          'cluster1'  : SgeDRMAAConfig( '/usr/local/sge/lib/lx26-amd64/libdrmaa.so' ,
                                         root ='/usr/local/sge',
                                         cell = 'cluster1' ) ,
```

```
        'cluster2' : SgeDRMAAConfig( '/usr/local/sge/lib/lx26-amd64/libdrmaa.so' ,
                                     root ='/usr/local/sge',
                                     cell = 'cluster2' ) ,
                    }
```

in this example, the cluster1 and cluster2 have differents features, number of nodes, memory ... and some jobs must run on cluster1 and the other on cluster2.

2.2 Second step: define which SystemExecution will be used for given program.

After having defined your ExecutionSystem you must specify in what conditions you will use it. We illustrate here the configuration of the DefaultDispatcher which links a job name to an ExecutionConfig and a queue.

```
from Mobyle.Dispatcher import DefaultDispatcher

DISPATCHER = DefaultDispatcher( {
       'blast2'    : ( EXECUTION_SYSTEM_ALIAS[ 'DRMAA_sge' ] , 'mobyle' ),
       'fastdnaml' : ( EXECUTION_SYSTEM_ALIAS[ 'DRMAA_torque' ] , 'mobyle' ),
       'toppred'   : ( EXECUTION_SYSTEM_ALIAS[ 'DRMAA_torque' ] , 'short' ),
       'dnapars'   : ( EXECUTION_SYSTEM_ALIAS[ 'SGE' ]    , 'long' ),
       'golden'    : ( EXECUTION_SYSTEM_ALIAS[ 'SYS' ]   ,   ''    ),
       'kitch'     : ( EXECUTION_SYSTEM_ALIAS[ 'LSF' ] , 'mobyle' ),
       'DEFAULT'   : ( EXECUTION_SYSTEM_ALIAS[ 'DRMAA_sge' ]    , 'mobyle' )
                          } )
```

or

```
from Mobyle.Dispatcher import DefaultDispatcher

DISPATCHER = DefaultDispatcher( {
       'job1' : ( EXECUTION_SYSTEM_ALIAS[ 'cluster1' ] , 'short' ),
       'job2' : ( EXECUTION_SYSTEM_ALIAS[ 'cluster1' ] , 'long' ),
       'DEFAULT'   : ( EXECUTION_SYSTEM_ALIAS[ 'cluster2' ] , 'mobyle' )
                          } )
```

Don't be afraid by this configuration once it is done you do not have to change it very often, and for most of you, you will have only one ExecutionConfig. For instance if you use SGE and one queue 'mobyle' for all jobs, the configuration will be:

```
EXECUTION_SYSTEM_ALIAS = { 'SGE'  : SGEConfig( root = '/usr/local/sge', cell='default' ) }
DISPATCHER = DefaultDispatcher( { 'DEFAULT' : ( EXECUTION_SYSTEM_ALIAS[ 'SGE'] , 'mobyle' )
```

3 add new execution system

If you have an other execution system not supported by Mobyle, you can develop our own ExecutionSystem. This Class must inherits from the abstract ExecutionSystem Class. The module must contain a class named as the module. Only this class can be used by Mobyle. Your module must be located in MOBYLEHOME/Src/Execution package. The new Class must implement 4 methods __init__, _run , getStatus and kill.

3.1 __init__

The init method has one argument which is the ExecutionConfig and is used to do all requirements to comunicate with the DRM. For instance set some variables in the environment ... usually these informations are contained in the ExecutionConfig. The __init__ method is not necessary if you don't need any configuration like SYS class.

3.2 _run

This method is the most complex you have to write to implement, and it has several responsabilities.

- This method is responsible for submitting the job to the DRMS.
- This method must be synchron with the job execution.
- As we do not use a server which can keep a record of all jobs, we must store some informations to retrieve a given job from an other process (cgi) to get the status of a job or to kill it. These informations are stored in a '.admin' file located in each job directory. The _run method is responsible for setting the value of the execution Sytem used and the key to retrieve this job on this Execution system to the an Admin object. The name is always accesible through self.execution_config_alias attribute and the key is the pid of the job for SYS or the job identifier in SGE....
- The ADMINDIR directory is a kind of table of all jobs currently in execution in Mobyle. It contains a symbolic link toward each job currently running. The _run method must make this link when it submits the job to the DRMS and remove it when the job is finished.
- Finally, when the job is finished the _run must map the status job to a Mobyle.Status and return it.

There is a Dummy class in the Execution package to help the developer to write his own Execution class.

3.3 getStatus

Has one argument, the identifier of the job for this DRMs (one that you store in .admin file in the _run method). This method queries the DMRS about the status of this job and maps this DRMS status to a Mobyle Status. If the job cannot be found in the DRMS the Status "unknown" must be returned.

3.4 kill

Has one argument, the identifier of the job for this DRMs (one that you store in .admin file in the _run method). This method asks the drms to kill the job and return None.

3.5 ExecutionConfig

You must implement also the ExecutionConfig which will be associated to this Class. The ExecutionConfig Class must be named as the new ExecutionSystem you develop with "Config" at the end. The module containing this ExecutionConfig class must be called also as the Class and located in MOBYLEHOME/Local/Config/Execution package.

Your ExecutionConfig class must inherit from ExecutionConfig and implement all requirements needed by your ExecutionSystem you have coded. At this point you can use your new Execution suitable to your need from the general Config as any other provided classes.

4 add new dispatcher

The Default dispatcher allows to associate one program to one ExecutionSystem and one queue. This queue is determined statically in the config. If you need something more dynamic, like compute the quqe based on the email of the user for instance, you must develop a new Dispatcher. This new dispatcher will inherit from Dispatcher Class and must implement 2 methods:

- getQueue
- getExecutionConfig

4.1 getQueue

Has one argument which is a JobState instance. You can easily access to all job characteristics (the name of the job, the email of the user...) to compute the queue name and return it.

4.2 getExecutionConfig

returns the ExecutionConfig which will used to execute a job.

5 DRMS requirement if DRMAA is used

5.1 python-drmaa

5.2 torque

To work with DRMAA and Mobyle (to run a synchronous job) torque must be able to report the status of a completed jobs. This feature is enabled by setting the keep_completed attribute on the job execution queue or server configuration.

5.3 LSF

We need lsf-drmaa from the FedStage DRMAA for LSF project http://sourceforge.net/projects/lsf-drmaa/