

Application requirements

Riccardo Murri <riccardo.murri@uzh.ch>

S3IT: Services and Support for Science IT

University of Zurich

Applications need to allocate computing resources. For instance, request 4 processors for 8 hours.

GC3Pie allows requesting:

- the number of processors that a job can use,
- the architecture (32-bit or 64-bit) of these processors,
- the guaranteed duration of a job,
- the amount of memory that a job can use (per processor).

More fine-grained matching is possible, but outside the scope of this introductory training. Resources are requested using additional constructor parameters for Application objects.

The allowed parameters are: requested_cores, requested_architecture, requested_walltime, requested memory.

Running parallel jobs

You request allocation of a certain number of processors using the requested_cores parameter: set it to the number of CPU cores that you want.

For example, the following runs the command mpixexec simulator on 4 processors:

```
class ZodsApplication(Application):
    # ...
    Application.__init__(self,
        arguments=['mpiexec', '-n', '4', 'simulator'],
        # ...
    requested_cores=4)
```

Note that GC3Pie only guarantees the availability of a certain number of processors; it is your application's responsibility to use them, e.g., by starting a command using MPI or any other parallel processing mechanism.

Requesting processor architecture

If you send the compiled executable along with your application, you need to select only resources that can run that binary file.

The requested_architecture parameter provides the choice between gc3libs.Run.Arch.X86_64 (for 64-bit Intel/AMD computers) and gc3libs.Run.Arch.X86_32 (for 32-bit ones):

```
from gc3libs import Run
class CodemlApplication(Application):
    # ...
Application.__init__(self,
    arguments=['./codeml.bin'],
    inputs = ['/usr/local/bin/codeml', ...]
    # ...
    requested_architecture=Run.Arch.X86_64)
```

Requesting running time

In order to ensure that your job is allotted enough time to run on the remote computing system, use the requested_walltime parameter.

```
from gc3libs.quantity
import minutes, hours, days

class CodemlApplication(Application):
    # ...
Application.__init__(self,
    # ...
    requested_walltime=8*hours)
```

You **must** use a gc3libs.quantity multiple for the requested_walltime parameter; any other value will be rejected with an error.

Units of time

The Python module gc3libs.quantity provides units for expressing time requirements in days, hours, minutes, seconds.

Just multiply the unit by the amount you need:

```
>>> an_hour = 1*hours
```

Or sum the amounts:

```
>>> two_days = 1*days + 24*hours
```

GC3Pie will automatically perform the conversions:

```
>>> two_hours = 2*hours
>>> another_two_hours = 7200*seconds
>>> two_hours == another_two_hours
True
```

Requesting memory

In order to secure a certain amount of memory for a job, use the requested_memory parameter.

Example:

```
from gc3libs.quantity import GB, MB, kB
class CodemlApplication(Application):
    # ...
Application.__init__(self,
    # ...
    requested_memory=8*GB)
```

Note that requested_memory expresses the total memory used by the job!

Units of memory

The Python module gc3libs.quantity provides units for expressing memory requirements in kilo-, Megaand Giga-bytes.

Just multiply the unit by the amount you need:

```
>>> a_gigabyte = 1*GB
>>> two_megabytes = 2*MB
```

GC3Pie will automatically perform the conversions:

```
>>> two_gigabytes = 2*GB
>>> another_two_gbs = 2000*MB
>>> two_gigabytes == another_two_gbs
True
```

All together now

When several resource requirements are specified, GC3Pie tries to satisfy *all* of them. If this is not possible, task submission fails and the task stays in state *NEW*.