GC3Pie: orchestrating large-scale execution of scientific applications

Riccardo Murri, Sergio Maffioletti
S3IT: Service and Support for Science IT,
University of Zurich

HPC-CH, June 11, 2015

Let's start with an example

Soil Mites Population Dynamics

Test which experimental designs give the most powerful data, by:

- *simulating soil mite population dynamics and observations of those dynamics in different experimental designs,*
- *fitting a Bayesian statistical model to the observed data to estimate the rates that govern the population dynamics.*

Mollie Brooks, http://www.popecol.org/

Scaling out from local experiment

– Code is an R function that performs the statistical computations.
– Typical run on one parameter set takes between 1 and 8 hours on a 4-core processor.

Scaling out from local experiment

- Code is an R function that performs the statistical computations.
- Typical run on one parameter set takes between 1 and 8 hours on a 4-core processor.
- Three sets of input parameters:
  - sampling parameters (22 rows),
  - isolation parameters (4 rows),
  - detection parameters (11 rows).

Scaling out from local experiment

- Code is an R function that performs the statistical computations.
- Typical run on one parameter set takes between 1 and 8 hours on a 4-core processor.
- Three sets of input parameters:
  - sampling parameters (22 rows),
  - isolation parameters (4 rows),
  - detection parameters (11 rows).
- Need to run the same function on each combination of parameters. . .
- . . . and repeat 100 times for statistical significance.

Scaling out from local experiment

- Code is an R function that performs the statistical computations.
- Typical run on one parameter set takes between 1 and 8 hours on a 4-core processor.
- Three sets of input parameters:
  - sampling parameters (22 rows),
  - isolation parameters (4 rows),
  - detection parameters (11 rows).
- Need to run the same function on each combination of parameters...
- ...and repeat 100 times for statistical significance.

So it's 96800 runs,
totalling circa 1'600'000 core hours.

How does GC3Pie help? *(1)*

Write a Python script to drive execution!

```
class GmbsimScript(SessionBasedScript):
  """
  Read the specified INPUT ``.csv`` files and submit jobs according
  to the content of those files.
  """
  # ...setup command-line options etc ...
  def new_tasks(self, extra):
    # read list of parameter sets from input files
    dates_and_sampling_exps = read_param_file(self.params.sampling)
    isolation_exps = read_param_file(self.params.isolation)
    detection_exps = read_param_file(self.params.detection)
    # loop over data to create jobs
    for date_and_sampling, isolation, detection in \
            product(dates_and_sampling_exps, isolation_exps, detection_exps):
      # ...prepare data and job description ...
      for n in range(1, self.params.replicates):
        yield GmbsimApplication(
          scriptfile=self.params.scriptfile,
          datafiles=self.params.datafiles,
          days_of_the_week=dates,
          sampling_exp=sampling,
          isolation_exp=isolation,
          detection_exp=detection,
          ...)
```

How does GC3Pie help? *(2)*

```
./gmbsim.py 100 sampling.csv isolation2.csv detection.csv \
  sim_run_dclone_design_test3.R weird_dates_sdur.R \
  -s take4 -C 600 -J 500 -w '72 hours' -c 4
# ...
Status of jobs in the 'take4' session: (at 01:25:10, 05/12/15)
        NEW  4764/9600   (49.62%)
    RUNNING   216/9600   (2.25%)
    STOPPED     0/9600   (0.00%)
  SUBMITTED   284/9600   (2.96%)
 TERMINATED  4336/9600   (45.17%)
TERMINATING     0/9600   (0.00%)
    UNKNOWN     0/9600   (0.00%)
      total  9600/9600  (100.00%)
```

What is *GC3Pie* and why do we need something like that ?

What is GC3Pie then?

GC3Pie is . . .

1. An *opinionated* Python framework for defining and running computational workflows;

What is GC3Pie then?

GC3Pie is . . .

1. An *opinionated* Python framework for defining and running computational workflows;
2. A *rapid development toolkit* for enabling user applications run unmodified on clusters and IaaS cloud resources;

What is GC3Pie then?

GC3Pie is . . .

1. An *opinionated* Python framework for defining and running computational workflows;
2. A *rapid development toolkit* for enabling user applications run unmodified on clusters and IaaS cloud resources;
3. The worst name ever given to a middleware piece. . .

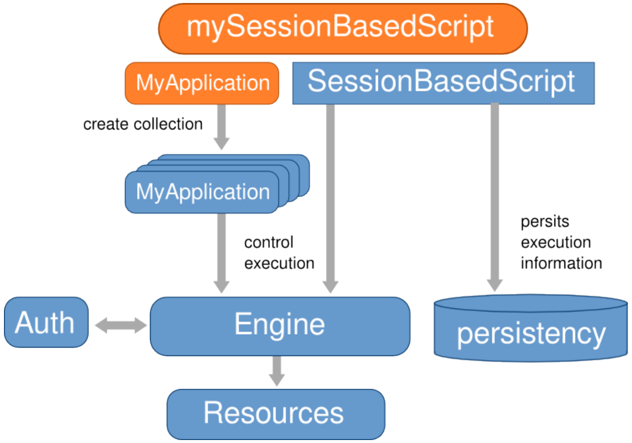
The issues GC3Pie wants to solve

1. **Portability:** Run on a different computing infrastructure without rewriting all the scripts.
2. **Code reuse:** Scripts are often very tied to a certain purpose, so they are difficult to reuse.
3. **Heavy maintenance:** the more a script does its job well, the more you'll find yourself adding *generic* features and maintaining requests from other users.

High-level architecture overview

Again, let's do this through an example.

High-level architecture



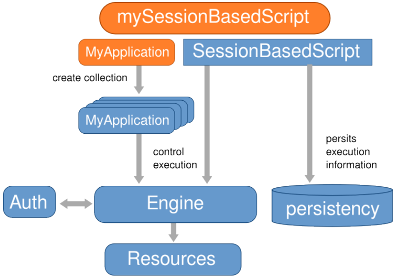
An application is a subclass of the `gc3libs.Application` class.

```
class GmbsimScript(SessionBasedScript):
  # ...
  def new_tasks(self, extra):
    ...
    # loop over data to create jobs
    for date_and_sampling, isolation, detection in \
            product(dates_and_sampling_exps, isolation_exps, detection_exps):
      # ...prepare data and job description ...
      for n in range(1, self.params.replicates):
        yield GmbsimApplication (
          scriptfile=self.params.scriptfile,
          datafiles=self.params.datafiles,
          days_of_the_week=dates,
          sampling_exp=sampling,
          isolation_exp=isolation,
          detection_exp=detection,
          ...)
```

Applications can be grouped into *collections*.

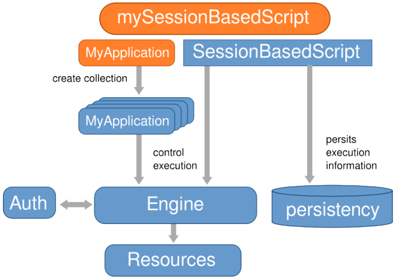
```
class GmbsimScript(SessionBasedScript):
  # ...
  def new_tasks(self, extra):
    ...
    # loop over data to create jobs
    for date_and_sampling, isolation, detection in \
        product(dates\_and\_sampling\_exps, isolation\_exps, detection\_exps):
      # ...prepare data and job description ...
      for n in range(1, self.params.replicates):
        yield GmbsimApplication(
          scriptfile=self.params.scriptfile,
          datafiles=self.params.datafiles,
          days_of_the_week=dates,
          sampling_exp=sampling,
          isolation_exp=isolation,
          detection_exp=detection,
          ...)
```

High-level architecture: `Engine`



Execution of applications and collections is delegated to an `Engine`.

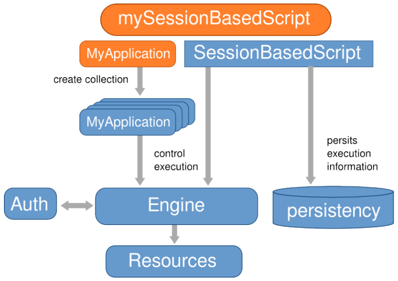
High-level architecture: resources



GC3Pie can execute applications on a variety of resources: *cloud-based VMs*, *batch-queueing clusters*, and any host which you can `ssh` to.

The `Engine` handles the access to computational *resources* transparently.

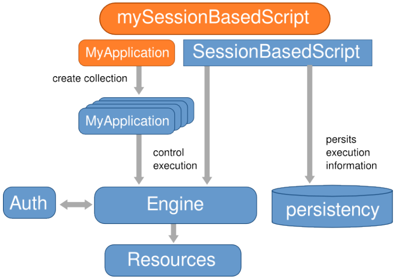
High-level architecture: resources



GC3Pie can execute applications on a variety of resources: *cloud-based VMs, batch-queueing clusters*, and any host which you can `ssh` to.

The `Engine` handles the access to computational *resources* transparently.

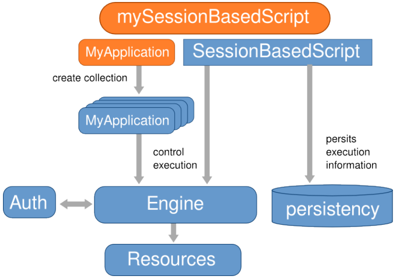
High-level architecture: resources



GC3Pie can execute applications on a variety of resources: *cloud-based VMs*, *batch-queueing clusters*, and any host which you can `ssh` to.

The `Engine` handles the access to computational *resources* transparently.

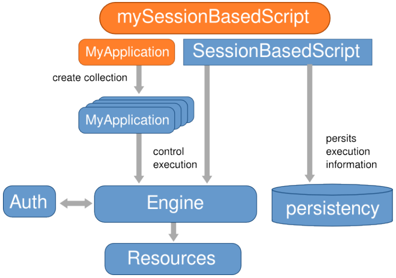
High-level architecture: resources



GC3Pie can execute applications on a variety of resources: *cloud-based VMs*, *batch-queueing clusters*, and any host which you can `ssh` to.

The `Engine` handles the access to computational *resources* transparently.

High-level architecture: `SessionBasedScript`



A convenient *SessionBasedScript* class contains already most of the control logic for instructing the execution engine.

The `SessionBasedScript` takes also care of *persisting* execution information.

To create a script just subclass `SessionBasedScript`.

```
class GmbsimScript(SessionBasedScript):
  # ...
  def new_tasks(self, extra):
    ...
    # loop over data to create jobs
    for date\_and\_sampling, isolation, detection in \
        product(dates\_and\_sampling\_exps, isolation\_exps, detection\_exps):
      # ...prepare data and job description ...
      for n in range(1, self.params.replicates):
        yield GmbsimApplication(
          scriptfile=self.params.scriptfile,
          datafiles=self.params.datafiles,
          days_of_the_week=dates,
          sampling_exp=sampling,
          isolation_exp=isolation,
          detection_exp=detection,
          ...)
```

Customization is done by overriding specific methods.

```
class GmbsimScript(SessionBasedScript):
  # ...
  def setup_options(self):
    self.add_param("-b", "-nb", "--burn-in", metavar="NUM",
             dest="nb", default=1,
             help="Execute NUM iterations for JAGS burn-in.")
    self.add_param("-i", "-ni", "--iter", metavar="NUM",
             dest="ni", default=3,
             help="Execute NUM JAGS main iterations.")
    ...
    # change default for the core/memory/walltime options
    self.actions['memory_per_core'].default = 1*Memory.GB
    self.actions['wctime'].default = '2 hours'
    self.actions['ncores'].default = 4
```

SessionBasedScript example

```
$ ./gmbsim.py --help
usage: gmbsim [-h] [-V] [-v] [--config-files CONFIG_FILES] [-c NUM]
              [-m GIGABYTES] [-r NAME] [-w DURATION] [-s PATH] [-u URL] [-N]
              [-C NUM] [-J NUM] [-o DIRECTORY] [-l [STATES]] [-b NUM] [-i NUM]
              [-t NUM]
              replicates sampling isolation detection scriptfile
              [datafiles [datafiles ...]]
```

From single tasks to workflows

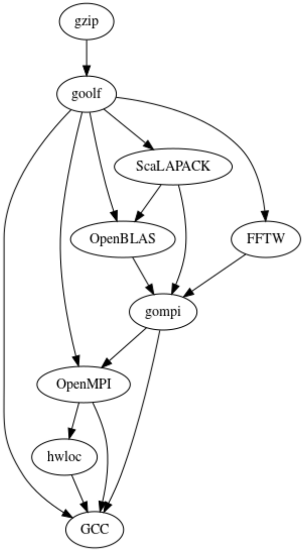
Task dependency management

An `Engine` manages all jobs concurrently.
What if there are inter-application dependencies?

GC3Pie provides *Task composition* support (workflow), created programmatically from Python code.

Which means, no graphical editor. But also means you can create workflows *on-the-fly* as your computation proceeds.

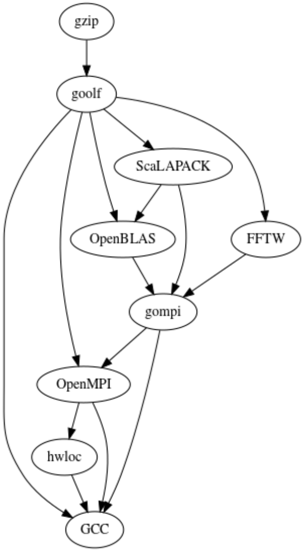
Example: EasyBuild



EasyBuild has built-in dependency resolution. To compile `gzip` you first have to build another 8 software packages.

Starting with version 2.2, EasyBuild can launch all these compilation jobs through GC3Pie.

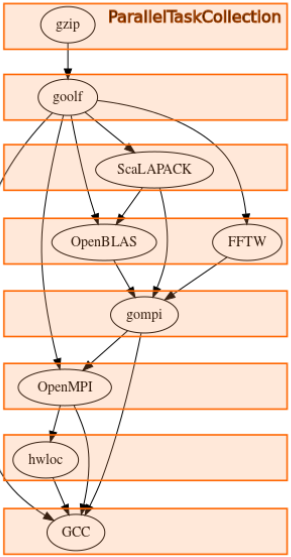
Dependency-based task management



Code-wise this is easy. Create a `DependentTaskCollection` and tell it what tasks it has to run and the dependencies of each.

```
coll = DependentTaskCollection()
coll.add(gzip_task, [goolf_task])
coll.add(goolf_task, [
  gcc_task,
  fftw_task,
  scalapack_task
])
...
```

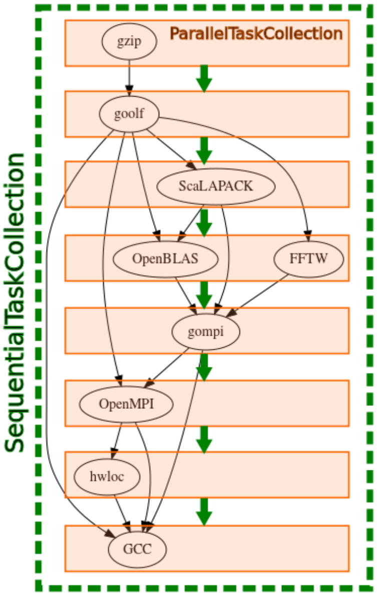
Parallel task management



Under the hood, GC3Pie groups independent tasks into a `ParallelTaskCollection`.

This means they can run independently.

Sequencing tasks



Several tasks and task collections can be forced to run in a sequence using a `SequentialTaskCollection`.

Sequencing tasks, II

The interesting thing about `SequentialTaskCollection` is that it can be built “lazily” while it runs.

In other words, not all tasks need to be known when the workflow starts running.

For example, GC3Pie sports a *differential evolution* numerical optimizer built using this mechanism.

References

Read more: http://gc3pie.readthedocs.org/

Thank you for your attention!

We're renaming!

Help choose a better name for GC3Pie!

Send suggestions and cast your vote at:
http://tinyurl.com/gc3pie-rename

Additional material

GC3Pie (SW) users

iBRAIN – High-throughput screening framework, Pelkmans Lab UZH.

Huygens Remote Manager – Web-based interface to Huygens Core for multi-user batch-scheduled deconvolution.

TRAL – the *Tandem Repeat Annotation Library*, Elke Schaper (ISB-SIB) et al.