Execo

Process based API to run large scale Experiments

Laurent Pouilloux[1] Matthieu Imbert[2]

[1]Héméra engineer
INRIA

[2]AVALON - LIP
ENS Lyon - INRIA

12-03-2013 / Héméra XP Tools workshop

Running large-scale experiments on distributed systems

Working on cluster, grid and cloud systems is based on

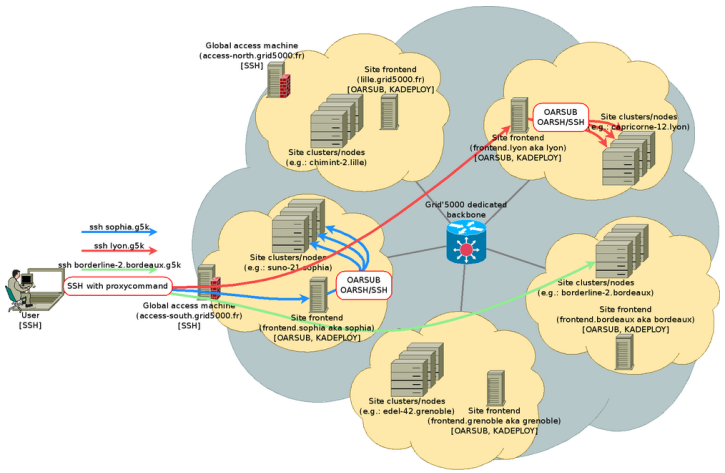
- the execution of parallel and distributed operations
- the ability of managing complex workflow
- the necessity of exploring a wide range of parameters

Require an automated system for rapid experimental development

Many processes on many hosts

We need to:

- execute remote command through SSH
- control the sequential and parallel execution



How to control many processes executed on many hosts ?

Execo

An API to manage local/remote and standalone/parallel processes

- A core API for local or remote, standalone or parallel, processes execution. Well suited for scripting workflows of parallel/distributed operations on local or remote hosts:
  - automate a scientific workflow
  - conducting computer science experiments
  - performing automated python tests
  - automate admin tasks
- A set of Grid5000 dedicated tools
- An extensible experiment engine

http://execo.gforge.inria.fr

Example: perform bandwith measurements with netcat

```
1  from execo import *
2  hosts = [ "lille", "reims", "lyon", "nancy" ]
3  targets = list(reversed(hosts))
4  servers = Remote("nc -l -p 6543 > /dev/null", hosts,
5      connexion_params = {'user': 'lpouilloux'})
6  clients = Remote("dd if=/dev/zero bs=50000 count=125 | nc -q 0 {{targets}} 6543
7      hosts, connexion_params = {'user': 'lpouilloux'})
8  servers.start()
9  sleep(1)
10 clients.run()
11 servers.wait()
12 print Report([ servers, clients ]).to_string()
13 for s in clients.processes():
14         print s.host().address
15         print s.remote_cmd()
16         print "stdout:\n%s" % ( s.stdout())
```

Outline

1. `execo`: powerful services for dealing asynchronously with local / remote, single / parallel operating system processes. Also: logging, time handling and misc services
2. `execo_g5k`: services for dealing with oar, oargrid, kadeploy, the g5k api
3. `execo_engine`: an extensible experiment engine and tools such as services for parameters sweeping.

Outline

module execo

- execo kernel:
  - a thread handles the lifecycle and outputs of all operating system processes
  - asynchronous I/O: poll on linux, select on osx ➜ very efficient
- allows client code to control process asynchronously: start a process, wait for it, kill it
- lifecycle and I/O of processes handled with callbacks. Default callbacks:
  - collect stdout / stderr
  - handle process death: exit code, duration
- You may supply custom callbacks if needed

Defining and controlling a process

execo.Process() and execo.SshProcess()

Hi level class hierachy for local processes `Process` or remote processes `SshProcess`

- control: `start`, `wait`, `kill`
- state: `error`, `exit_code`
- I/O: `stdout`, `stderr`

`SshProcess` inherits from `Process` and targets a `Host`:

- hostname or address
- user *(optional)*
- port *(optional)*
- keyfile *(optional)*

All remote connexions also use default `ConnexionParams` or user-supplied one.

Examples

Performing a ls locally

```
1  from execo import *
2  process = Process("ls /")
3  process.run()
4  print "process:\n%s" + str(process)
5  print "process stdout:\n" + process.stdout()
6  print "process stderr:\n" + process.stderr()
```

Measuring latency to gw-reims on lille frontend

```
1  from execo import *
2  process = SshProcess('ping gw-reims | cut -f4 -d "=" ', 'lille', \
3      connexion_params = {'user': 'lpouilloux'})
4  process.start()
5  sleep(5)
6  process.kill()
7  print process.stdout()
```

Actions

- set of parallel processes to a list of hosts
- same as process:
  - control: `start`, `wait`, `kill`
  - state: `ok`, `error`, `exit_code`, `start_date`, `end_date`
- access to individual processes (`stdout`, `stderr`)
- as for remote processes: connect to list of `Host`, using default or specific `ConnexionParams`

Remote

Launch a command remotely on several hosts, with ssh or a similar remote connexion tool.

```
1  my_remote = EX.Remote(cmd, hosts_list)
2  my_remote.start()
3  my_remote.wait()
```

Possibility to use TaktutRemote() on g5k frontends.

Example: perform bandwith measurements with netcat

```
1  from execo import *
2  hosts = [ "lille", "reims", "lyon", "nancy" ]
3  targets = list(reversed(hosts))
4  servers = Remote("nc -l -p 6543 > /dev/null", hosts,
5      connexion_params = {'user': 'lpouilloux'})
6  clients = Remote("dd if=/dev/zero bs=50000 count=125 | nc -q 0 {{targets}} 6543
7      hosts, connexion_params = {'user': 'lpouilloux'})
8  servers.start()
9  sleep(1)
10 clients.run()
11 servers.wait()
12 print Report([ servers, clients ]).to_string()
13 for s in clients.processes():
14         print s.host().address
15         print s.remote_cmd()
16         print "stdout:\n%s" % ( s.stdout())
```

File transfer

All transfers are performed in parallel.

execo.Put()

To put files on several remote hosts.

execo.Get()

To get files on several remote hosts.

Possibility to use TaktutPut() and TaktukGet() on g5k frontends

Substitutions

A very convenient syntax to express variations in the command line executed in parallel on the set of remote hosts of an Action.

- all occurences of the literal string {{{host}}} are substituted by the address of the connected Host.
- all occurences of {{<expression>}} are substituted in the following way: <expression> must be a python expression, which will be evaluated in the context (globals and locals) of the Action declaration, and which must return a sequence. {{<expression>}} is replaced for all individual remote host by <expression>[index % len(<expression>)].

Example: perform bandwith measurements with netcat

```
1  from execo import *
2  hosts = [ "lille", "reims", "lyon", "nancy" ]
3  targets = list(reversed(hosts))
4  servers = Remote("nc -l -p 6543 > /dev/null", hosts,
5      connexion_params = {'user': 'lpouilloux'})
6  clients = Remote("dd if=/dev/zero bs=50000 count=125 | nc -q 0 {{targets}} 6543
7      hosts, connexion_params = {'user': 'lpouilloux'})
8  servers.start()
9  sleep(1)
10 clients.run()
11 servers.wait()
12 print Report([ servers, clients ]).to_string()
13 for s in clients.processes():
14         print s.host().address
15         print s.remote_cmd()
16         print "stdout:\n%s" % ( s.stdout())
```

Logging
execo.logger()

- standard python logger
- default setup ideal for most situations:
  - quiet
  - except: warning (with detailed infos) for all processes with an error (start error, non zero exit code)
  - can explicitely tell some processes to not log warnings on such conditions
- easy to analyse problems in long experience logs post-mortem

Flow control

Using the Action state methods and some other functions

- `ok`, `wait`, `start`, `started`, ...
- `wait_any_actions`, `wait_multiple_actions`

Sequential and Parallel actions

Defining some actions

```
1 action1 = Remote(cmd1, hosts)
2 action2 = Remote(cmd2, hosts1)
3 action3 = Remote(cmd3, hosts1)
```

SequentialActions

```
1 seq = SequentialActions([action1, action2, action3])
2 seq.run()
```

ParallelActions

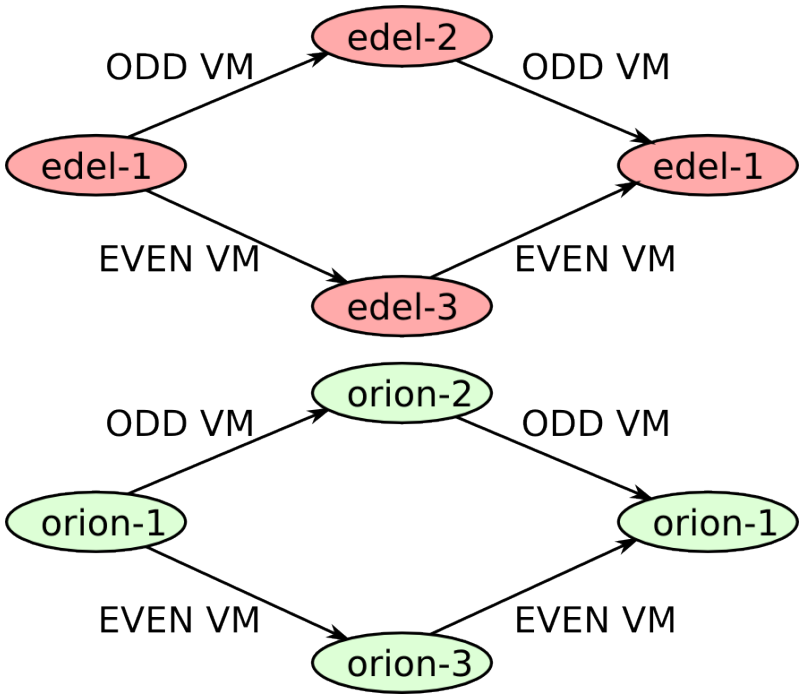
```
1 para = ParallelActions([action1, action2, action3])
2 para.run()
```

Example of combined usage

SPLIT-MERGE VM migration on many clusters

On several clusters, we have VMs running on host 0.

- Migration SPLIT:
  - even VMs go to host 2
  - odd VMs go to host 1
- Migration MERGE:
  - even VMs go back to host 0
  - odd VMs go back to host 0



Example of combined usage

```
1   cluster_actions = []
2   for cluster, params in vms_params.iteritems():
3       split_actions = []; migraction_01 = []; migraction_02 = []
4       for vm in vms_0[cluster]['vms']:
5           cmd = MigrationMeasure( vm['vm_id'], params['hosts'][0], params['hosts'][1], 'SPLIT')
6           migraction_01.append( EX.Remote(cmd, [params['hosts'][0]]))
7       migsplit_01 = EX.SequentialActions(migraction_01)
8       for vm in vms_1[cluster]['vms']:
9           cmd = MigrationMeasure( vm['vm_id'], params['hosts'][0], params['hosts'][2], 'SPLIT')
10          migraction_02.append( EX.Remote(cmd, [params['hosts'][0]]))
11      migsplit_02 = EX.SequentialActions(migraction_02)
12      split_actions = EX.ParallelActions([migsplit_01, migsplit_02] )
13
14      merge_actions = []; migraction_10 = []; migraction_20 = []
15      for vm in vms_0[cluster]['vms']:
16          cmd = MigrationMeasure( vm['vm_id'], params['hosts'][1], params['hosts'][0], 'MERGE')
17          migraction_10.append( EX.Remote(cmd, [params['hosts'][1]]) )
18      migmerge_10 = EX.SequentialActions(migraction_10)
19      for vm in vms_1[cluster]['vms']:
20          cmd = MigrationMeasure( vm['vm_id'], params['hosts'][2], params['hosts'][0], 'MERGE')
21          migraction_20.append( EX.Remote(cmd, [params['hosts'][2]]) )
22      migmerge_20 = EX.SequentialActions( migraction_20 )
23      merge_actions = EX.ParallelActions( [migmerge_10, migmerge_20] )
24
25      cluster_actions.append(EX.SequentialActions([split_actions, merge_actions]))
26  EX.ParallelActions(cluster_actions).run()
```

Free workflow

Example :

- reserve some nodes on any g5k site
- deploy some of those nodes
- wait the deployment end
- start network capture process on a server
- start some processes on the deployed nodes
- make another reservation on another site
- stop some processes on some servers
- and so on...

Outline

Configuring your computer

Changing .ssh/config:

```
Host *.g5k
  User lpouilloux
  ProxyCommand ssh lpouilloux@access.grid5000.fr "nc -q 0 `basename %h .g5k` %p"
  StrictHostKeyChecking no
```

Creating .execo.conf.py:

```
1  import re
2  def host_rewrite_func(host):
3      return re.sub("\.grid5000\.fr$", ".g5k", host)
4  def frontend_rewrite_func(host):
5      return host + ".g5k"
6  g5k_configuration = {
7      'oar_job_key_file': 'path/to/ssh/key/inside/grid5000',
8      'default_frontend' : 'lyon',
9      'api_username' : 'g5k_username'
10     }
11 default_connexion_params = {'host_rewrite_func': host_rewrite_func}
12 default_frontend_connexion_params = {'host_rewrite_func': frontend_rewrite_func
```

OAR functions

Submission and deletion

- OarSubmission(resources, walltime, job_type, ...)
- oarsub([OarSubmission(...), frontend)])
- oardel([(job_id1, frontend1), (job_id2, frontend2)])

Job information

- get_oar_job_info, get_oar_job_nodes
- wait_oar_job_start, get_current_oar_jobs

Network information

- get_oar_job_subnets
- get_oar_job_kavlan

Example

```
1  from execo import *
2  from execo_g5k import *
3  jobs = oarsub([
4    ( OarSubmission(resources = "/cluster=2/nodes=4"), "nancy")
5  ])
6  wait_oar_job_start(jobs[0][0], jobs[0][1])
7  nodes = get_oar_job_nodes(jobs[0][0], jobs[0][1])
8  sources, targets = group_nodes_by_cluster(nodes)
9  servers = Remote("nc -l -p 6543 > /dev/null", hosts,
10     connexion_params = {'user': 'lpouilloux'})
11 clients = Remote("dd if=/dev/zero bs=50000 count=125 | nc -q 0 {{targets}} 6543
12     hosts, connexion_params = {'user': 'lpouilloux'})
13 servers.start()
14 sleep(1)
15 clients.run()
16 servers.kill().wait()
17 print Report([ servers, clients ]).to_string()
18 oardel([(jobs[0][0], jobs[0][1])])
```

OARGRID functions

Submission and deletion

oargridsub, oargriddel

Job information

get_current_oargrid_jobs, get_oargrid_job_info,
get_oargrid_job_oar_jobs, get_oargrid_job_nodes

Flow control

wait_oargrid_job_start

Performing a deployment on Grid5000

First define your deployment:

```
1 deployment = execo_g5k.Deployment( hosts = hosts, env_name = env_name)
2 OR
3 deployment = execo_g5k.Deployment( hosts = hosts, env_file = env_file)
```

Then launch it:

```
1 Hosts = execo_g5k.deploy(deployment, num_tries = 3)
```

Deploy function checked that the nodes are successfully deployed and retry if some nodes failed to deploy

```
Deploying environment squeeze-x64-prod ...
check which hosts are already deployed among: set([Host('taurus-9.lyon.grid5000.fr'), Host('helios-9.sophi
KO Host('taurus-6.lyon.grid5000.fr')
KO Host('edel-8.grenoble.grid5000.fr')
...
try 1, deploying on: set([Host('taurus-9.lyon.grid5000.fr'), Host('helios-9.sophia.grid5000.fr'), Host('ed
check which hosts are already deployed among: set([Host('taurus-9.lyon.grid5000.fr'), Host('helios-9.sophi
OK Host('taurus-6.lyon.grid5000.fr')
OK Host('edel-8.grenoble.grid5000.fr')
OK Host('helios-8.sophia.grid5000.fr')
...
kadeploy reported newly deployed hosts:
[Host('taurus-9.lyon.grid5000.fr'), Host('helios-9.sophia.grid5000.fr'), Host('edel-9.grenoble.grid5000.fr
all deployed hosts:     set([Host('taurus-9.lyon.grid5000.fr'), Host('helios-9.sophia.grid5000.fr'), Host
still undeployed hosts: set([])
deploy finished in 1 tries, 11m47s
2013-02-27 13:12:07,759 execo INFO deploy  duration  attempted  deployed     deployed     total     total
2013-02-27 13:12:07,760 execo INFO                    deploys    as reported  as reported  already   still
2013-02-27 13:12:07,760 execo INFO                               by kadeploy  by check     deployed  undepl
2013-02-27 13:12:07,760 execo INFO ------------------------------------------------------------------------
2013-02-27 13:12:07,760 execo INFO #0      1s        None       None         0            0         8
2013-02-27 13:12:07,760 execo INFO #1      11m46s    0          8            8            8         0
2013-02-27 13:12:07,760 execo INFO deployed hosts: set([Host('taurus-9.lyon.grid5000.fr'), Host('helios-9
2013-02-27 13:12:07,761 execo INFO undeployed hosts: set([])
2013-02-27 13:12:07,761 execo INFO taurus-9.lyon.grid5000.fr helios-9.sophia.grid5000.fr edel-9.grenoble.g
```

API utils

Specific functions

get_g5k_sites, get_g5k_clusters, get_g5k_hosts, get_site_clusters, get_cluster_hosts, get_cluster_site, get_host_cluster, get_host_site, group_hosts

Gathering resources information

get_host_attributes, get_cluster_attributes, get_site_attributes, get_resource_attributes

Example

```
1  from execo import *
2  from execo_g5k import *
3  hosts = get_g5k_sites()
4  targets = list(reversed(hosts))
5  servers = Remote("nc -l -p 6543 > /dev/null", hosts,
6      connexion_params = {'user': 'lpouilloux'})
7  clients = Remote("dd if=/dev/zero bs=50000 count=125 | nc -q 0 {{targets}} 6543
8      hosts, connexion_params = {'user': 'lpouilloux'})
9  servers.start()
10 sleep(1)
11 clients.run()
12 servers.wait()
13 print Report([ servers, clients ]).to_string()
14 for s in clients.processes():
15         print s.host().address
16         print get_site_attributes(s.hosts().address)['description']
17         print s.remote_cmd()
18         print "stdout:\n%s" % ( s.stdout())
```

Some simple but useful tools

Locate in contrib/tools on the git repository

g5k-show-structure

Display the Grid5000 structure site - cluster

g5k-show-all-jobs

Show all your running jobs on grid5000

g5k-clean-all-jobs

Kill all your running jobs on grid5000

g5k-clean-oar-stdout-stderr-files

Remove OAR.stdout and OAR.stderr on all Grid5000 frontends

Funk: (F)ind (U)r (N)odes on G5(K)

A inverse disco software

Ask for resources on different sites or clusters and find slot for your experiment

- analyzing resources (grid5000, sites and cluster, vlan and kavlan)
- gathering planning informaiton from the g5k API
- compute slots and filter to match your resources
- propose a slot (can be automatic)
- generate the oargridsub command
- perform the reservation

.

Can be used in CLI or as a python API

Funk: (F)ind (U)r (N)odes on G5(K)

Examples

Installed in /home/lpouilloux/bin/ on Lyon frontend with a bash script that set up the python path.

Finding a slot for a multicluster/multisite/grid experiment

```
funk -m find_slots -r lille:10,lyon:10,sophia:10 -w 2:00:00
```

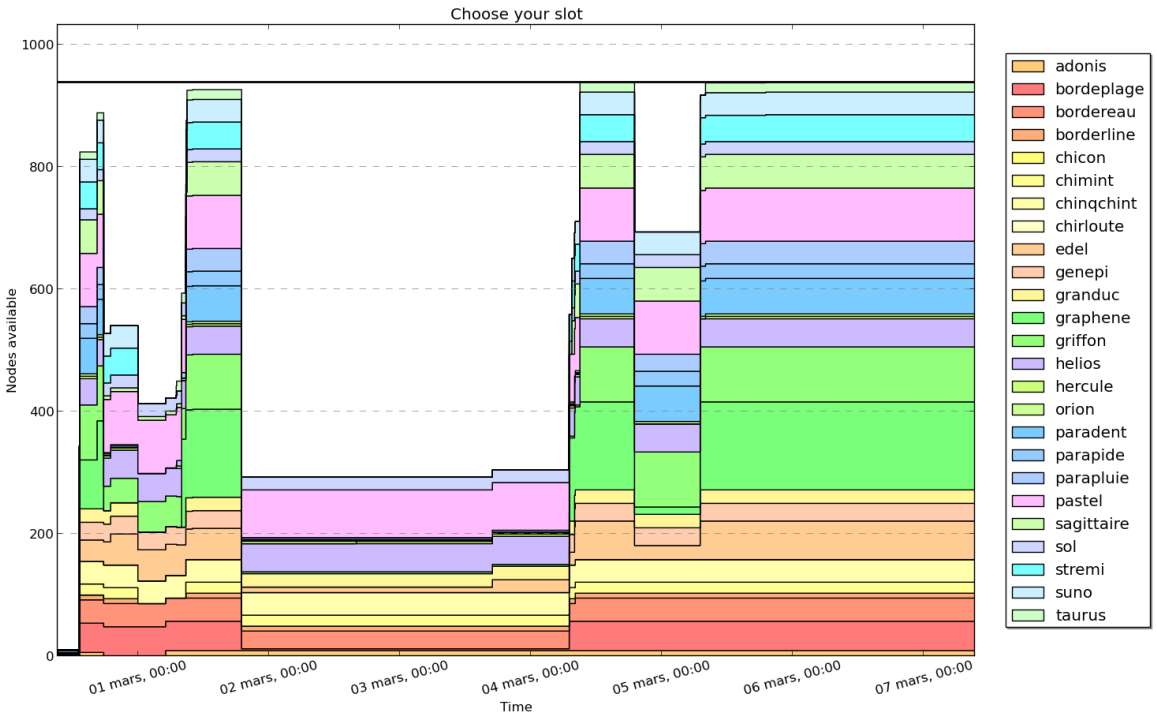
Determining the maximum number of nodes for the next halfday

```
funk -m max_nodes -r lyon:0,sophia:0,grenoble:0 -w 10:00:00 -t 0.07
```

Funk: (F)ind (U)r (N)odes on G5(K)

In progress

- add a user defined start date for the resource discovery
- add the g5k chart option
- draw graphs (Gantt + number of nodes)



Outline

Basics

Define an engine

Create a file myengine.py

```
1 class myengine(Engine):
2     def __init__(self):
3         super(myengine, self).__init__()
4     def run(self):
5         ''' do whatever you want'''
```

Start an experiment

```
execo-run myengine -ML
```

It create the result_dir, parse the options, setup the log and run the experiment

Continue an experiment

You can add some parameters and relaunch the engine

```
execo-run myengine -c result_dir
```

Using a ParamSweeper

Create a python dict with all the parameters you want to explore

```
parameters = {
    "param1": [2, 3],
    "param2": ["kvm", "xen"]
    }
```

Use the sweep to create all the combinations:

```
sweeps = sweep(parameters)
```

ParamSweeper to create an iterable object with some interesting properties like `get_next`, `skip`, `done`, `get_remaining`

```
sweeper = ParamSweeper(os.path.join(self.result_dir, "sweeps"), sweeps)
```

Persistent, thread-safe, process-safe.

Testing your MPI application on one node for all clusters

The content of the run() method:

```
1   frontends = get_g5k_sites()
2   EX.Put(frontends, 'LU_MPI.tar.bz2', connexion_params = {'user': 'lpouilloux'}).run()
3   EX.Remote('tar -xjf LU_MPI.tar.bz2', frontends, connexion_params = {'user': 'lpouilloux'}).run()
4
5   parameters = {'cluster': get_g5k_clusters(), 'n_core': [1, 2, 4, 8, 16]}
6   sweeps = sweep(parameters)
7   sweeper = ParamSweeper(os.path.join(self.result_dir, "sweeps"), sweeps)
8   while len(sweeper.get_remaining()) > 0:
9       comb = sweeper.get_next()
10      if comb['n_core'] > get_host_attributes(comb['cluster']+'-1')['architecture']['smt_size']:
11          sweeper.skip(comb); continue
12      site = get_cluster_site(comb['cluster'])
13      job = EX5.oarsub( [(EX5.OarSubmission(resources = "{cluster=\\'"+comb['cluster']+"\\'}/nodes=1",
14              job_type = 'allow_classic_ssh', walltime ='0:05:00'), site)])
15      EX5.wait_oar_job_start(job[0][0], job[0][1])
16      nodes = EX5.get_oar_job_nodes( job[0][0], job[0][1])
17      EX.Remote('cd ~/NPB3.3-MPI && make clean && make suite', nodes,
18          connexion_params = {'user': 'lpouilloux'}).run()
19      lu_bench = EX.Remote('mpirun -n '+str(comb['n_core'])+\
20          ' -mca btl sm,self ~/NPB3.3-MPI/bin/lu.A.'+str(comb['n_core']), nodes,
21          connexion_params = {'user': 'lpouilloux'}).run() EX5.oardel(job)
22      for p in lu_bench.processes():
23          print p.stdout()
24      sweeper.done(comb)
```

Statistics about the running engine

stats() method allow to follow the evolution of the engine.

```
1  ipython
2  In [1]: from execo_engine import ParamSweeper
3  In [2]: ps = ParamSweeper('sweeps')
4  In [3]: ps.update()
5  In [4]: ps.stats()['done_ratio']
```

Testing a java application for big data management

Anthony Simonet

- 1 "Active Data" server that store and manage on-demand big data structure
- n clients that:
  - ask to create structures
  - update the structure (transitions)

Systematic exploration of

- the number of clients
- the number of client per nodes
- the number of transition requests per client

VM_Migration_Measurements

with Jonathan Rouzaud-Cornabas

A wide range of parameters and experimental conditions

```
parameters = {              "n_nodes": [2, 3],
                            "n_vm": [1, 2, 3, 4, 6, 12],
                            "n_cpu":[1, 2],
                            "hdd_size" : [2, 4, 8],
                            "mem_size" : [512, 1024, 2048],
                            "technology": ["kvm"]
                            }
clusters = [ "sol", "helios", "taurus", "edel"]
stress_type = [ "None", "cpu", "ram", "hdd" ]
```

Using Parallel and Sequential actions for workflow definition:

- 2 nodes: sequential, parallel and crossed migration
- 3 nodes: split-merge, circseq, circpara, circcross

g5k_bench_flops

Matthieu Imbert

Perform XHPL benchmark on all g5k clusters

A complex workflow

- node preparation
- flop execution
- complex scheduler (besteffort during the day, passive during the night and weekend)

Managing error using native python methods

https://github.com/mimbert/g5k_bench_flops

Execo strong points

- manage processes on remote hosts
  - start, wait, kill asynchronously individual or groups of local or remote operating system processes
  - remote file copying
- powerful workflow managements
  - program your workflow in a very intuitive way
  - can accomodate even the most complex workflows
- extensible experiment engine and associated software services
  - Grid5000: multi-cluster, multi-site friendly
  - powerfull logging and reporting
  - simplification of the parameters exploration

If you want to test it:
http://execo.gforge.inria.fr/doc/tutorial.html

Thank you for your attention. Any questions ?