Managing Large Scale Experiments In Distributed Systems

Cristian Camilo Ruiz Sanabria

MESCAL team, LIG/INRIA
LAAS, Toulouse

Ecole Doctorale Mathématiques, Sciences et Technologies de L'information, Informatique

March 12, 2013

Outline

Reusing work

If I have seen further it is by standing on the shoulders of giants - Isaac Newton -

Experimentation with distributed systems

It is very difficult to re-do or re-use experiments that involve distributed systems.

Reusing work

If I have seen further it is by standing on the shoulders of giants - Isaac Newton -

Replay and Reproduce

Experimentation with distributed systems

It is very difficult to re-do or re-use experiments that involve distributed systems.

Why do we want to do real experimentation?

Test the system in real conditions

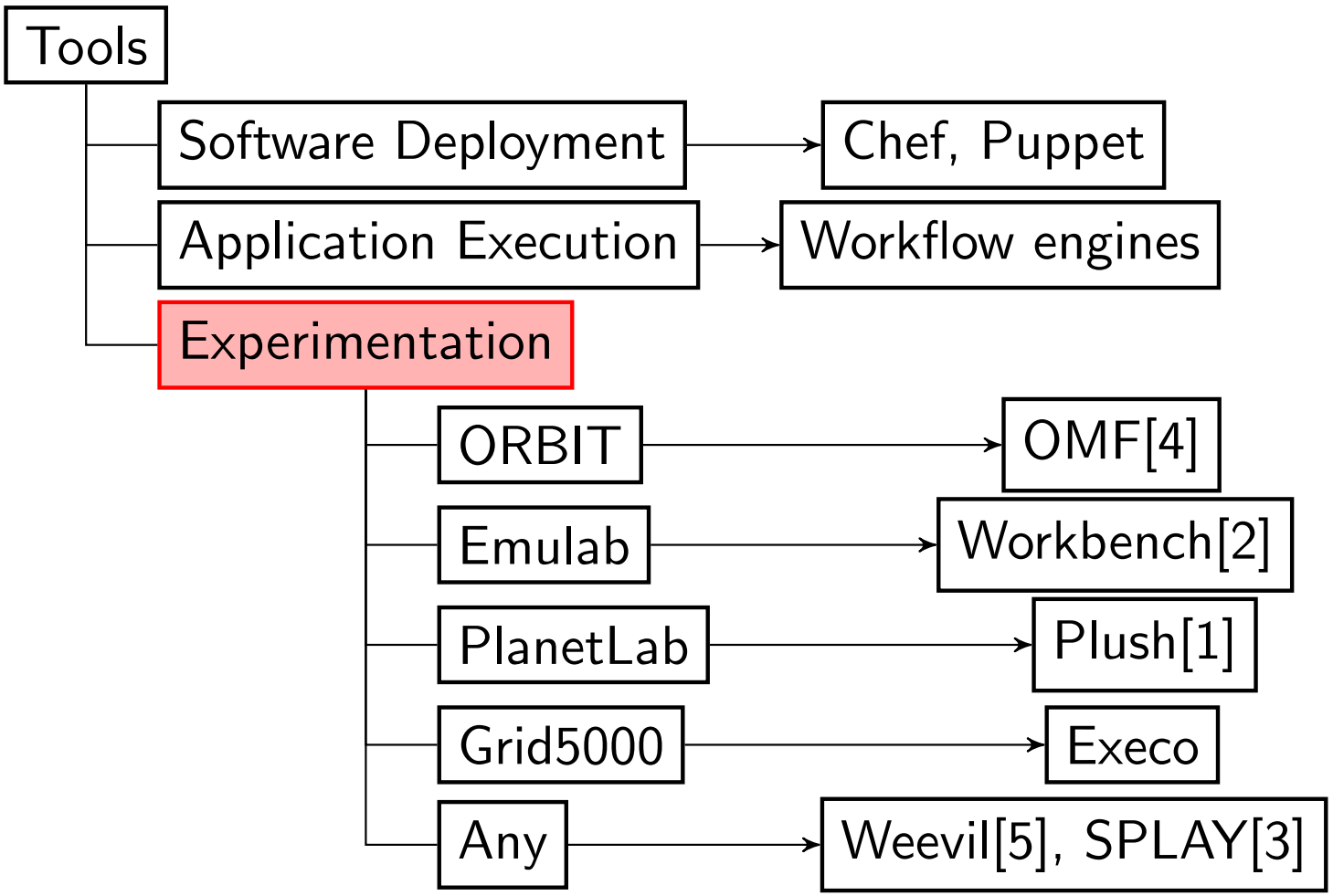
- Simulation is not enough.
- Discrepancies between the expected behavior of an application as modeled or simulated and its actual behavior when deployed in a live environment.

Ad hoc approaches

Normally seen in experimentation with distributed systems using real platforms. Some Issues:

- Dependency on the person who coded.
- Difficult to repeat, some parameters could be forgotten. Thus, it is almost impossible to reproduce.
- Difficult to control large scale experiments (involving many nodes).
- Description and code organization can be difficult to understand.

Some Tools


Tools
Software Deployment
Chef, Puppet
Application Execution
Workflow engines
Experimentation
ORBIT
OMF[4]
Emulab
Workbench[2]
PlanetLab
Plush[1]
Grid5000
Execo
Any
Weevil[5], SPLAY[3]

Some Issues

- Platform dependent.
- Domain dependent (Algorithm Testing, Wireless Networks, Software deployment).
- Scalability.
- Expressiveness (Experiment Description)

Some Issues

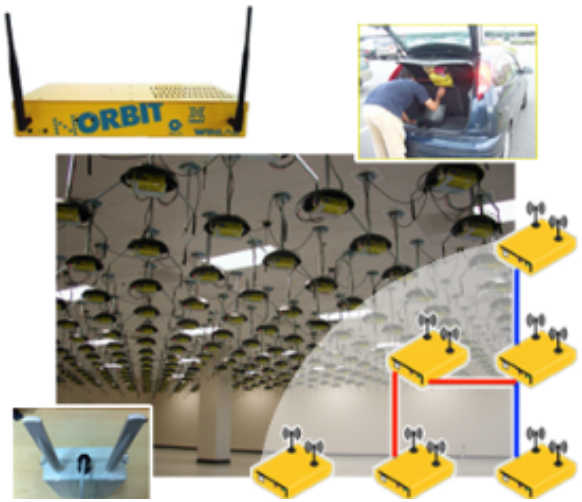
Platform dependent : Workbench It is bound to Emulab

Experimentation Workbench: Management of Experimentation, Data, and Analyses

Flux Research Group, University of Utah School of Computing
Emulab Project

Some Issues

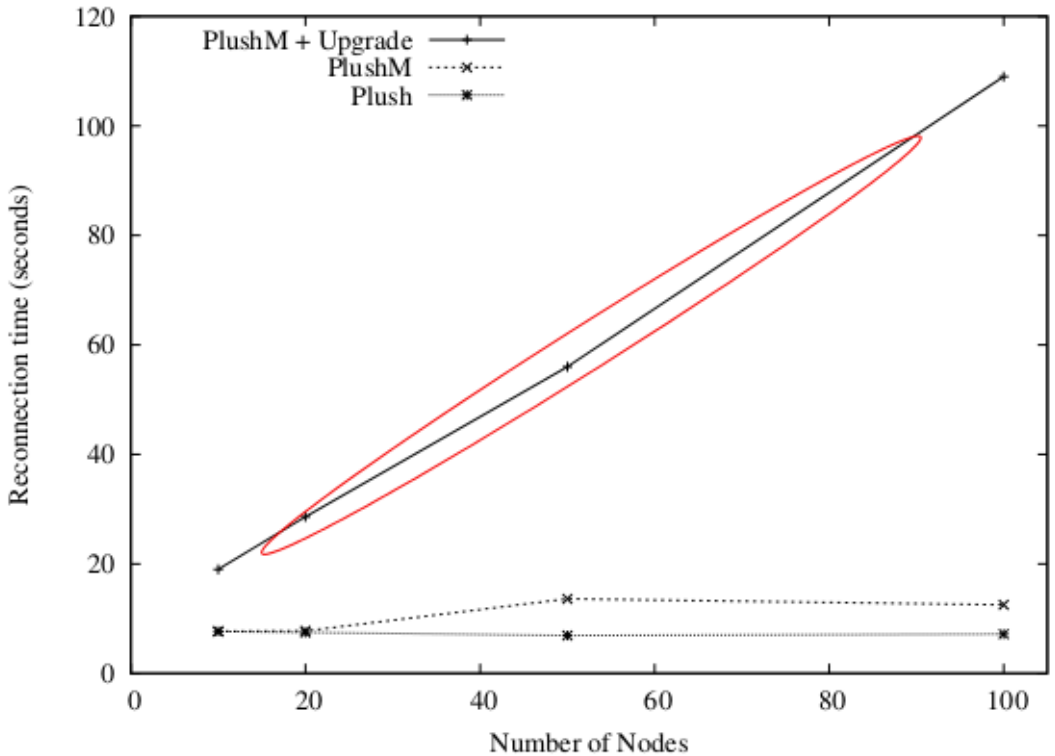
Domain dependent (Algorithm Testing, Wireless Networks, Software deployment).




```
4  function join(n0)                                  -- n0: some node in the ring
5    predecessor = nil
6    finger[1] = call(n0, {'find_successor', n.id})
7    call(finger[1], {'notify', n})
8  end
9  function stabilize()                               -- periodically verify n's successor
10   local x = call(finger[1], 'predecessor')
11   if x and between(x.id, n.id, finger[1].id, false, false) then
12     finger[1] = x                                  -- new successor
13   end
14   call(finger[1], {'notify', n})
15 end
16 function notify(n0)                                -- n0 thinks it might be our predecessor
17   if not predecessor or between(n0.id, predecessor.id, n.id, false, false) then
18     predecessor = n0                               -- new predecessor
19   end
20 end
21 function fix_fingers()                              -- refresh fingers
22   refresh = (refresh % m) + 1                       -- 1 ≤ refresh ≤ m
23   finger[refresh] = find_successor((n.id + 2^(refresh - 1)) % 2^m)
24 end
25 function check_predecessor()                        -- checks if predecessor has failed
26   if predecessor and not ping(predecessor) then
27     predecessor = nil
28   end
29 end
```

Some Issues

Scalability.



Some Issues

Experiment Description

```
<?xml version="1.0" encoding="utf-8"?>
<plush>
  <project name="Bullet">
    <software name="bullet" type="tar">
      <package name="bullet.tar" type="web">
        <path>http://strength.ucsd.edu/~albrecht/bullet.tar</path>
        <dest>bullet.tar</dest>
      </package>
    </software>
    <component name="bullet_hosts">
      <rspec>
        <num_hosts>130</num_hosts>
        <sword>
          <request>
            <group>
              <name>bullet_hosts</name>
              <num_machines>all</num_machines>
              <fiveminload>0, 0, 2, 5, 1</fiveminload>
              <cpuspeed>1, 2, 5, 5, 1</cpuspeed>
            </group>
          </request>
        </sword>
      </rspec>
      <software name="bullet" />
      <resources>
        <resource type="planetlab" group="ucsd_bullet" />
      </resources>
    </component>
    <application_block name="Bullet">
      <execution>
        <component_block name="run_bullet">
          <component name="bullet_hosts" />
          <process_block name="run">
            <process name="appmacedon">
              <path>./appmacedon</path>
              <cwd>bullet</cwd>
              <log_manager type="default" use_api="true">
                <fd fd="1" type="file" path_prefix="bulletlog-" path_postfix=".txt" />
              </log_manager>
            </process>
          </process_block>
          <barrier_block name="bullet_barrier">
            <predecessor name="run" />
            <barrier name="ready to stream" type="1" max="130" knee_det="true"/>
          </barrier_block>
        </component_block>
      </execution>
    </application_block>
  </project>
</plush>
```

Difficult to read and understand

```
set ns [new Simulator]
source tb_compat.tcl

# STATIC PART: nodes, networks, and agents.
set cnode  [$ns node] # Define a node
set snode  [$ns node]
set lan    [$ns make-lan "$cnode $snode" 100Mb 0ms]
set client [$cnode program-agent]
set server [$snode program-agent]

# DYNAMIC PART: events.
set do_client [$ns event-sequence {
    $client run -command "setup.sh"
    $client run -command "client.sh"
}]
set do_server [$ns event-sequence {
    $server run -command "server.sh"
}]
set do_expt [$ns event-sequence {
    $do_server start  # Do not wait for completion
    $do_client run    # Run client, wait till end
}]
$ns at 0.0 "$do_expt run"
$ns run
```

Approach

Expo: Experiment Engine for Distributed Platforms

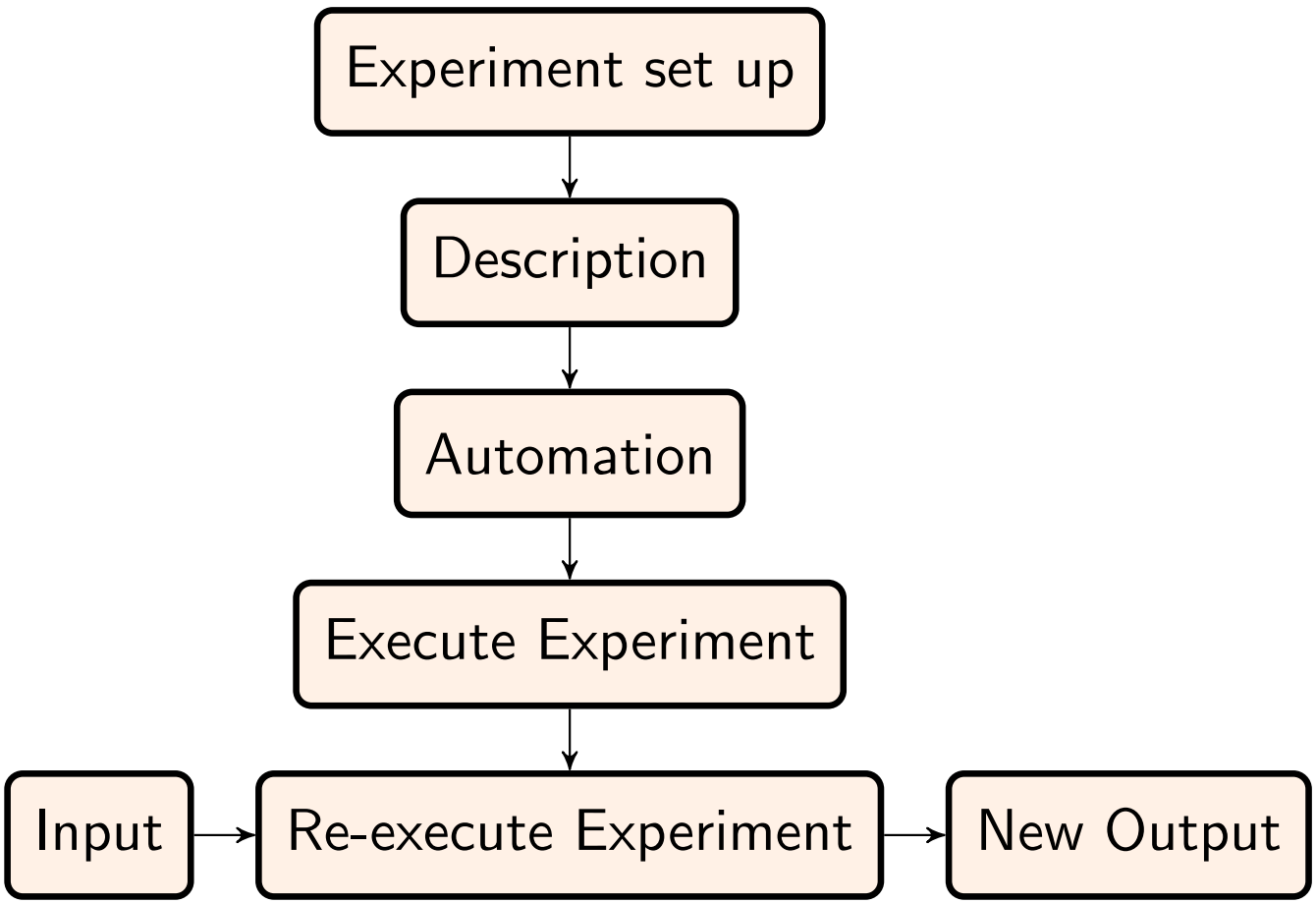
Kameleon: Appliance Generator

Expo

Experimental engine which provides

- Flexible experiment description.
  - Domain Specific Language based on Ruby
  - Borrows syntax from Capistrano and Rake.
- Efficient execution of the experiment on a large set of resources.
- Abstractions for managing resources, tasks and results.
- Interaction with Grid'5000 and Planetlab APIs.
- Interactive mode.

Flow of experimentation



Concepts

We introduce three abstractions to describe an experiment:

- Resource Set: resources are grouped and some characteristics are associated to them.
- Tasks: the building blocks to structure the experiment. Associate software to execute on a group of resources. They can be seen as Rake tasks.
- Results: help the collection of results and give some quick information about the execution of the experiment.

Resource Set

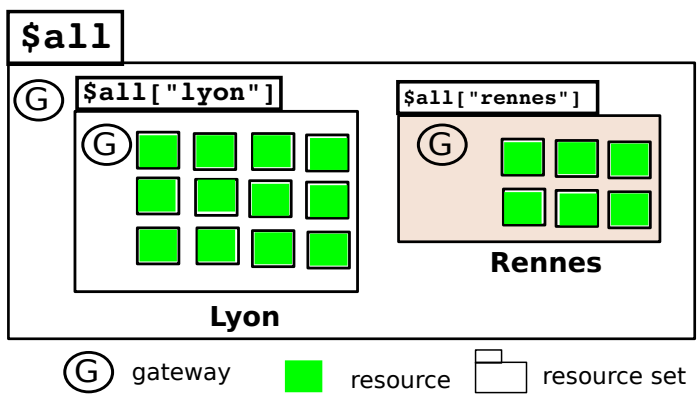
```
require 'g5k_api'
reserv=ExpoEngine::new(@connection)
reserv.site=["lyon","rennes"]
reserv.resources=["nodes=12","nodes=6"]
reserv.walltime=7200

reserv.run!

# All The resources reserved

$all


$all.gateway
# New resource set with only
# resources from lyon cluster
lyon_nodes=$all["lyon"]
lyon_nodes.gateway
```



Resource Set

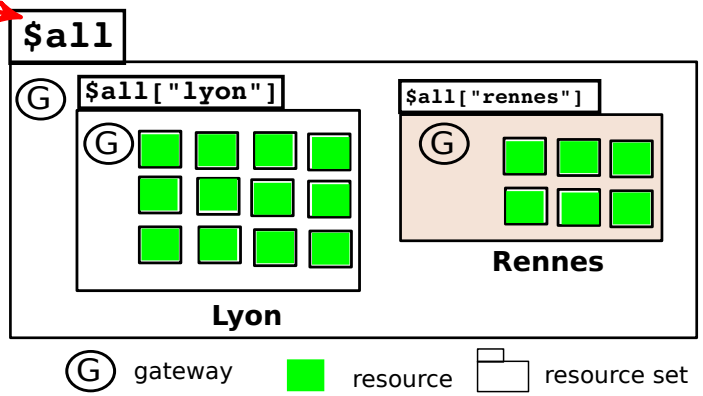
```
require 'g5k_api'
reserv=ExpoEngine::new(@connection)
reserv.site=["lyon","rennes"]
reserv.resources=["nodes=12","nodes=6"]
reserv.walltime=7200

reserv.run!

# All The resources reserved

$all


$all.gateway
# New resource set with only
# resources from lyon cluster
lyon_nodes=$all["lyon"]
lyon_nodes.gateway
```



Resource Set

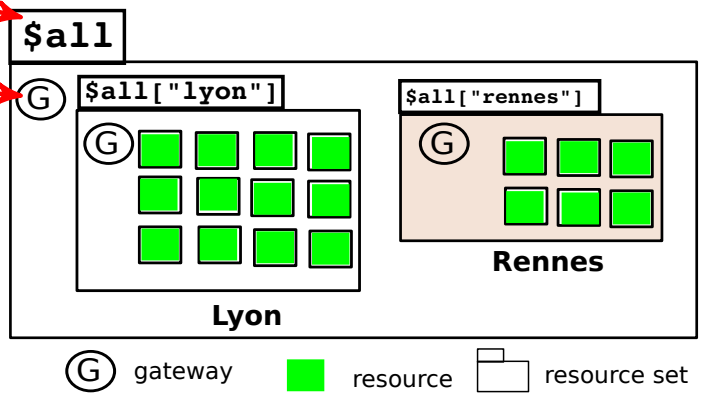
```
require 'g5k_api'
reserv=ExpoEngine::new(@connection)
reserv.site=["lyon","rennes"]
reserv.resources=["nodes=12","nodes=6"]
reserv.walltime=7200

reserv.run!

# All The resources reserved

$all


$all.gateway
# New resource set with only
# resources from lyon cluster
lyon_nodes=$all["lyon"]
lyon_nodes.gateway
```



Resource Set

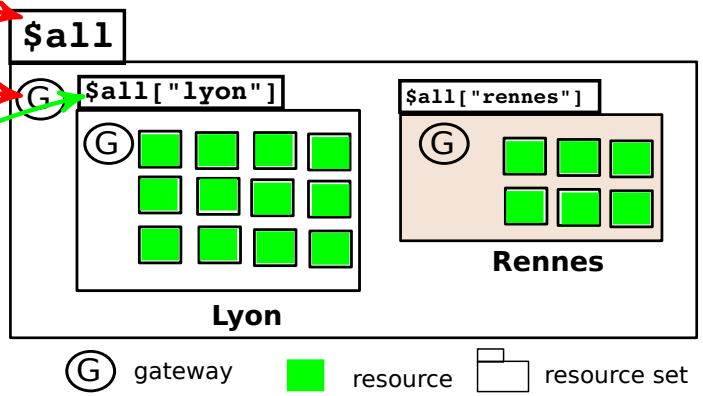
```
require 'g5k_api'
reserv=ExpoEngine::new(@connection)
reserv.site=["lyon","rennes"]
reserv.resources=["nodes=12","nodes=6"]
reserv.walltime=7200

reserv.run!

# All The resources reserved

$all


$all.gateway
# New resource set with only
# resources from lyon cluster
lyon_nodes=$all["lyon"]
lyon_nodes.gateway
```



Resource Set

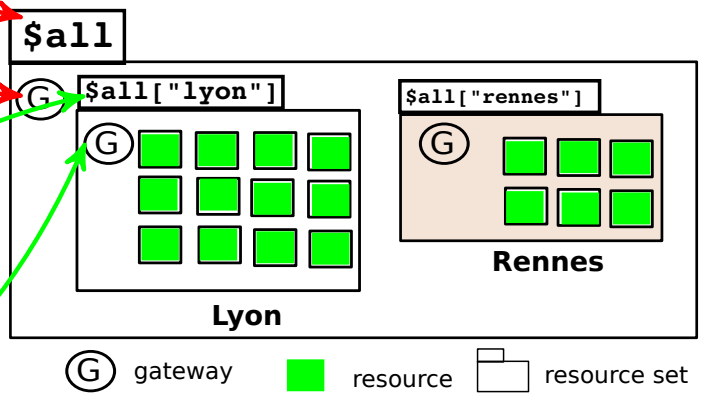
```
require 'g5k_api'
reserv=ExpoEngine::new(@connection)
reserv.site=["lyon","rennes"]
reserv.resources=["nodes=12","nodes=6"]
reserv.walltime=7200

reserv.run!

# All The resources reserved

$all


$all.gateway
# New resource set with only
# resources from lyon cluster
lyon_nodes=$all["lyon"]
lyon_nodes.gateway
```



Resource Set Operations

Operation	Description
[]	Subset of the resource Set, nodes[1..4] creates a subset with 4 resources.
to_resources	Returns an array composed of resource objects.
length()	Returns the number of resources in the ResourceSet.
each_slice_power2	Iterates over subsets composed of power of 2 resources.
resource_file	Creates a file with the resources' hostnames.

Table: Resource Set Operations

DSL: Expo Commands

Command	Description
task (node,options = , &block)	execute command
barrier	wait for all asynchronous task
put (file,node, path={})	Copy file to node
parallel_section (&block)	execute section in parallel
sequential_section(&block)	execute code sequentially

Table: Expo Commands

Expo architecture

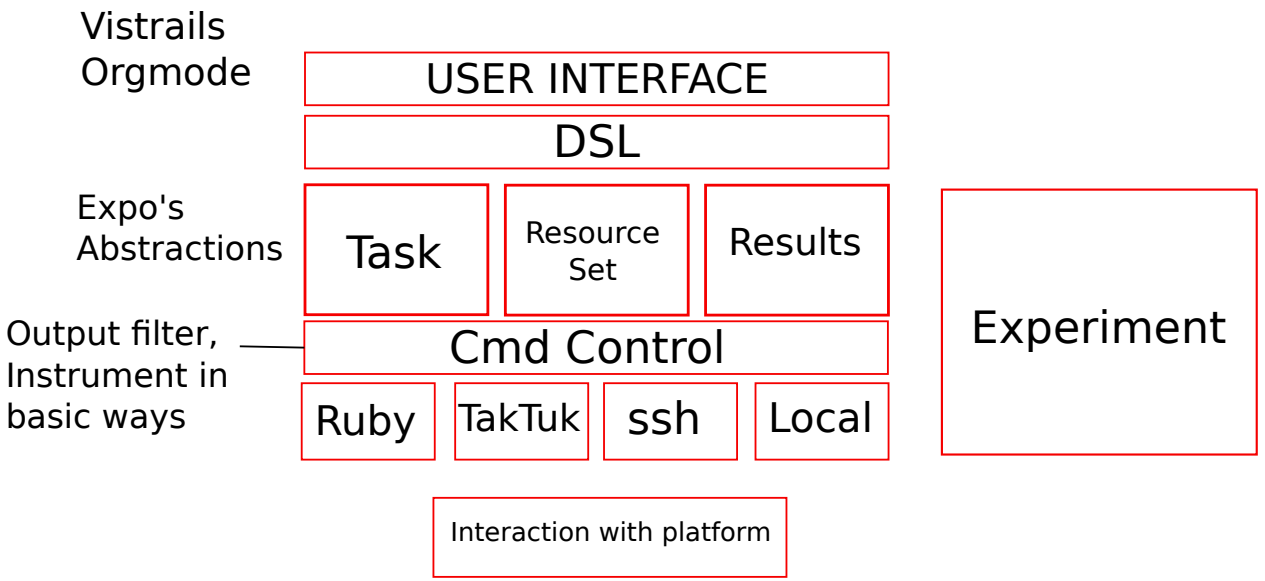
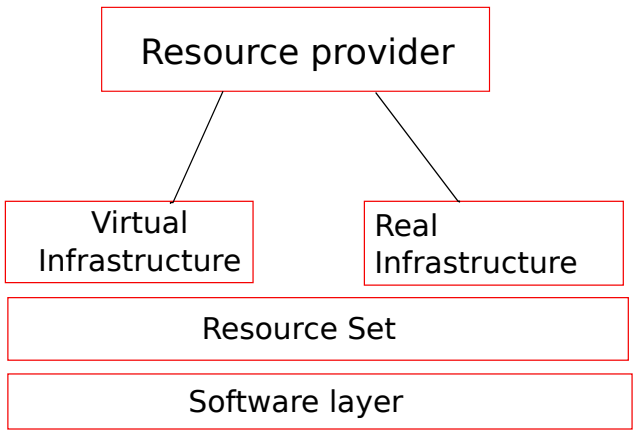


Figure: architecture

Complex experiments

PlanetLab
OAR or other
Cluster Batch Scheduler
Cloud provider



Provisioning the resources

Well known software => Chef

Patches or user software => Custom scripts

Expo Example

Expo Simple Example.

```
reserv=ExpoEngine::new(@connection)
reserv.site=["lille","grenoble"]
reserv.resources=["nodes=2","nodes=3"]
reserv.walltime=200



reserv.run!



task :test, :target => Experiment.resources
       run("hostname")
end



res.each { |r| puts r.duration }
puts "mean : " + res.mean_duration
```

Expo Example

Expo Simple Example.

```
reserv=ExpoEngine::new(@connection)
reserv.site=["lille","grenoble"]
reserv.resources=["nodes=2","nodes=3"]
reserv.walltime=200
```

Resource Definition

```
reserv.run!
```

```
task :test, :target => Experiment.resources
       run("hostname")
end
```

```
res.each { |r| puts r.duration }
puts "mean : " + res.mean_duration
```

Expo Example

Expo Simple Example.

```
reserv=ExpoEngine::new(@connection)
reserv.site=["lille","grenoble"]
reserv.resources=["nodes=2","nodes=3"]
reserv.walltime=200
```

Resource Definition

```
reserv.run!
```

Running the reservation

```
task :test, :target => Experiment.resources
       run("hostname")
end
```

```
res.each { |r| puts r.duration }
puts "mean : " + res.mean_duration
```

Expo Example

Expo Simple Example.

```
reserv=ExpoEngine::new(@connection)
reserv.site=["lille","grenoble"]
reserv.resources=["nodes=2","nodes=3"]
reserv.walltime=200
```

Resource Definition

```
reserv.run!
```

Running the reservation

```
task :test, :target => Experiment.resources
       run("hostname")
end
```

Task definition and execution

```
res.each { |r| puts r.duration }
puts "mean : " + res.mean_duration
```

Expo Example

Expo Simple Example.

```
reserv=ExpoEngine::new(@connection)
reserv.site=["lille","grenoble"]
reserv.resources=["nodes=2","nodes=3"]
reserv.walltime=200
```

Resource Definition

```
reserv.run!
```

Running the reservation

```
task :test, :target => Experiment.resources
       run("hostname")
end
```

Task definition and execution

```
res.each { |r| puts r.duration }
puts "mean : " + res.mean_duration
```

Results

Expo Examples

```
require 'g5k_api'
reserv=ExpoEngine::new(@connection)
reserv.site=["lille"]
reserv.resources=["cluster='chimint'}/nodes=4"]
reserv.walltime=7200
reserv.run!

$all.each{|node| copy  node.nodefile, node, param=>"tmp"}

task_tlm=Task::new(
   "~/tlm/run_mpi 1 400 20 10 50 matched /tmp/machines",
   $all, #resource set
   "Tes1")

File.open("test_experiment.txt",'w') { |f|
  20.times.each{ |x|
     id,res=task_tlm.execute
     res.each{ |result| f.write
         "#{result['host_name']}  #{result.duration} \n"}
    }
}
cleanup()
```

Expo Examples

```
require 'g5k_api'

base_url = "http://public.grenoble.grid5000.fr/~cruizsanabria/"
environment = "tlm_simulation.env"

reserv=ExpoEngine::new(@connection)
reserv.site=["nancy","rennes","lille","grenoble","sophia"]
reserv.resources=["nodes=1"]
reserv.name = "TLM_code"
reserv.walltime=2000
reserv.environment=base_url+environment

reserv.run!

$all.gen_keys

copy $all.node_file, $all.first, :path=> "/root/nodes.deployed"

id, res = task $all.first, "./lancer_grid 1 3869 192 561 25 1 sc"
get_results($all,"/root/tlm/bin/profile.*","~/profiles")
```

Expo Examples: Planetlab Nodes Availability

Monitoring Planetlab.

```
require 'expo_planetlab'
## getting resources form planetlab Slice
get_resources

task_mon=Task::new("hostname",$all,"Monitoring")

File.open("Planetlab_avail.txt",'w+'){|f|
  f.puts "Date \t Time \t Num_Res"

  240.times{
    data_me = Time::now.to_i
    id, res = task_mon.execute
    time = res.total_duration
    f.puts "#{data_me} \t #{time} \t #{res.length}"
    sleep(60)
    }
}
```

Planetlab Nodes availability

From slice of 354 nodes.

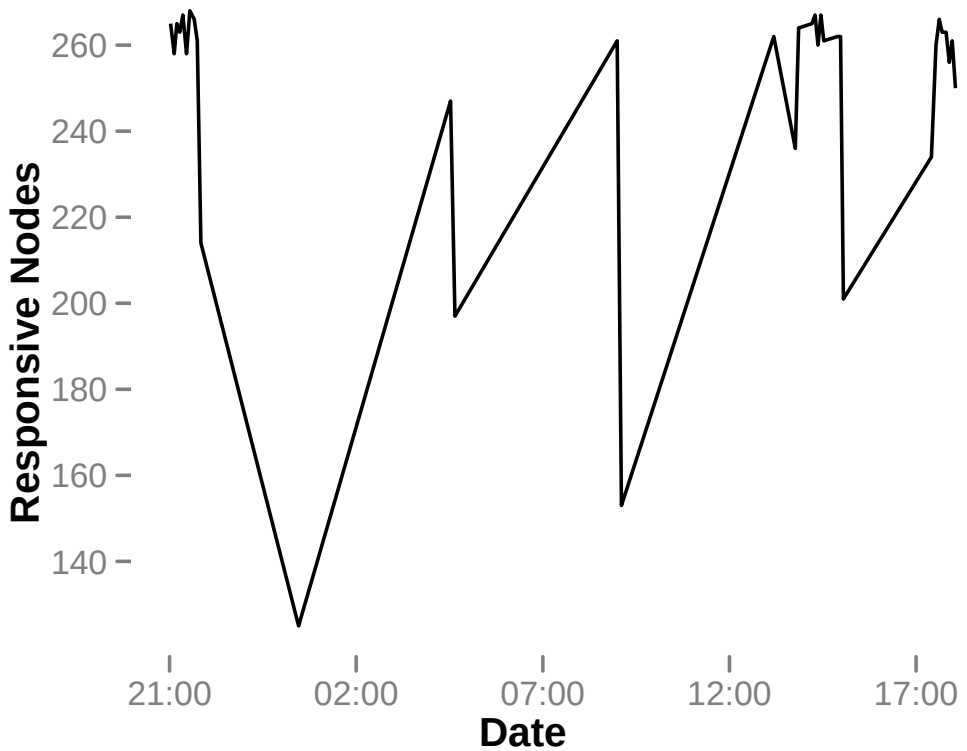


Figure: Planetlab nodes availability

Expo Console

Setting up an experiment implies the management of different software packages and the testing of different parameters

Very hard and error prone process

```
Expo Console > sall
=> #<Expo::ResourceSet:0x7f3eb4c0dfb0
 @properties={},
 @resource_files={},
 @resources=
  [#<Expo::ResourceSet:0x7f3eb2076690
    @properties=
     {:alias=>"genepi",
      :gateway=>"frontend.grenoble.grid5000.fr",
      :site=>"grenoble",
      :name=>"ExpoEngine",
      :id=>1415645},
    @resource_files={},
    @resources=
     [#<Expo::Resource:0x7f3eb2076438
       @properties=
        {:gateway=>"frontend.grenoble.grid5000.fr",
         :site=>"grenoble",
         :name=>"genepi-28.grenoble.grid5000.fr"},
       @type=:node>],
    @type=:resource_set>],
 @type=:resource_set>
Expo Console > task sall.first, "hostname"
genepi-28.grenoble.grid5000.fr
2013-02-04 14:16:24 +0100 [INFO] [ Sequential Task:0 ]  [ Executed ]   hostname
2013-02-04 14:16:24 +0100 [INFO] [ Sequential Task:0 ]  [ On Node ]  genepi-28.grenoble.grid5000.fr
2013-02-04 14:16:24 +0100 [INFO] [ Sequential Task:0 ]  [ Elapsed Time ] 0.00389 secs
=> [0,
 [{"stdout"=>"genepi-28.grenoble.grid5000.fr\n",
   "end_time"=>Mon Feb 04 14:16:24 +0100 2013,
   "host_name"=>"genepi-28.grenoble.grid5000.fr",
   "stderr"=>"",
   "rank"=>"1",
   "status"=>"0",
   "command_line"=>"cd . ; hostname ",
   "start_time"=>Mon Feb 04 14:16:24 +0100 2013}]]
Expo Console >
```

Comparison Plush vs Expo

Plush Vs Expo, two metrics:

- Experiment Description
- Scalability of the execution

Expo vs Plush, Experiment description

Listing 1: Plush

```
<?xml version="1.0" encoding="utf-8"?>
<gush>
  <project name="Testing overhead">
    <component name="Cluster1">
      <rspec>
        <num_hosts>20</num_hosts>
      </rspec>
      <resources>
        <resource type="ssh" group="local"/>
      </resources>
    </component>

      <experiment name="simple">
        <execution>
          <component_block name="cb1">
            <component name="Cluster1"/>
            <process_block name="p2">
              <process name="test">
<path>hostname</path>
                <cmdline>
                  <arg></arg>
                </cmdline>
              </process>
            </process_block>
          </component_block>
        </execution>
      </experiment>
  </project>
</gush>
```

Listing 2: Expo

```
reserv=ExpoEngine::new(@connection)
reserv.site=["nancy","sophia"]
reserv.resources=["nodes=200","nodes=100"]
reserv.name = "Expo Scalability"
reserv.walltime=2000

reserv.run!

sizes=[10,50,100,200,300]

$all.each_slice_array(sizes) do |nodes|

  task_mon= Task::new("hostname",nodes," Mon")
  (30).times{
    id,res = task_mon.execute
    puts " #{res.length} : #{res.duration}"
  }

end
```

Expo vs Plush, Experiment description

Listing 3: Plush

```
<?xml version="1.0" encoding="utf-8"?>
<gush>
  <project name="Testing overhead">
    <component name="Cluster1">
      <rspec>
        <num_hosts>20</num_hosts>
      </rspec>
      <resources>
        <resource type="ssh" group="local"/>
      <
                <component name="Cluster1"/>
                <process_block name="p2">
                  <process name="test">
<path>hostname</path>
                    <cmdline>
      </experiment>
  </project>
</gush>
```

Does not have the resources definition

Only one instance of the experiment

Listing 4: Expo

```
reserv=ExpoEngine::new(@connection)
reserv.site=["nancy","sophia"]
reserv.resources=["nodes=200","nodes=100"]
reserv.name = "Expo Scalability"
reserv.walltime=2000

reserv.run!

sizes=[10,50,100,200,300]

$all.each_slice_array(sizes) do | nodes|

  task_mon= Task::new("hostname",nodes," Mon")
  (30).times{
    id,res = task_mon.execute
    puts " #{res.length} : #{res.duration}"
  }

end
```

Scalability

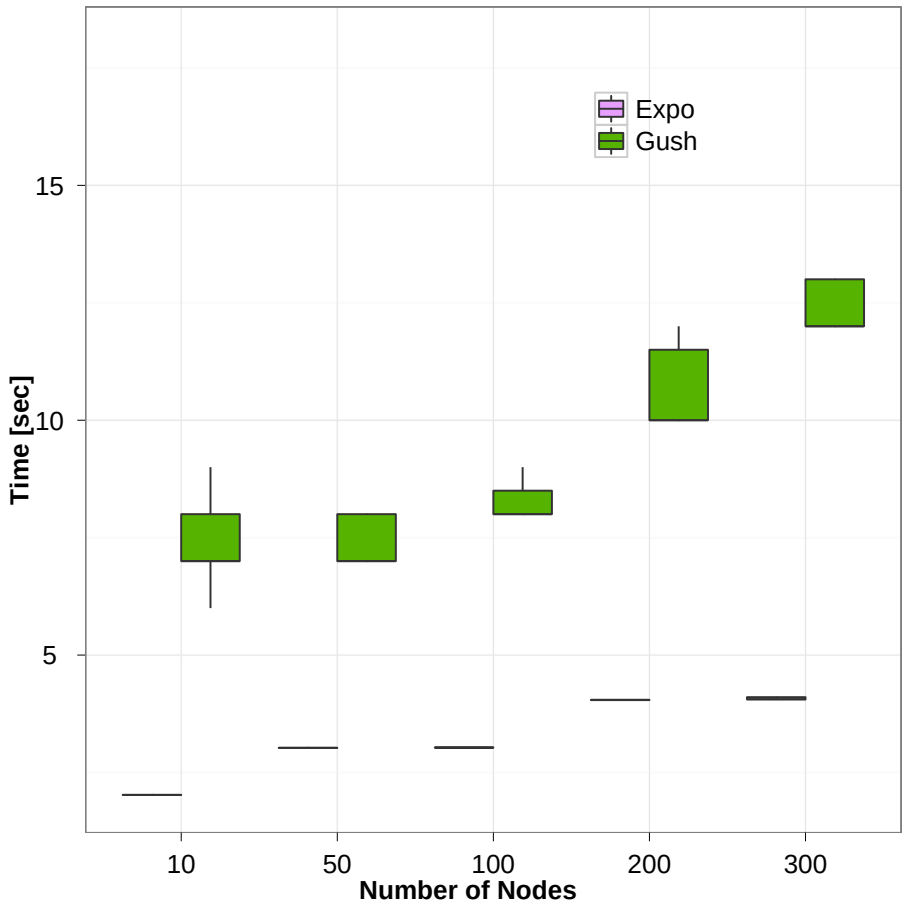


Figure: Plush vs Expo

Drawbacks

- Interface with the user, it could be complicated for a non expert user.
- tracking of changes in the experiment.

Expo current work

- Integration with other tools: Vistrails[1], Chef[2], Capistrano[3].
- Extend the concepts of Task and Result abstractions.

[1] http://www.vistrails.org
[2] http://www.opscode.com/chef/
[3] https://github.com/capistrano/capistrano

Kameleon

Reproducible appliance Generator

Motivations

Infrastructures such as FutureGrid and Grid5000 offer the possibility of deploying an image which comprises: OS and an application layer => Appliance.

This image is deployed can be loaded whether onto "bare metal" (real hardware) or on virtualized hardware using a hypervisor.

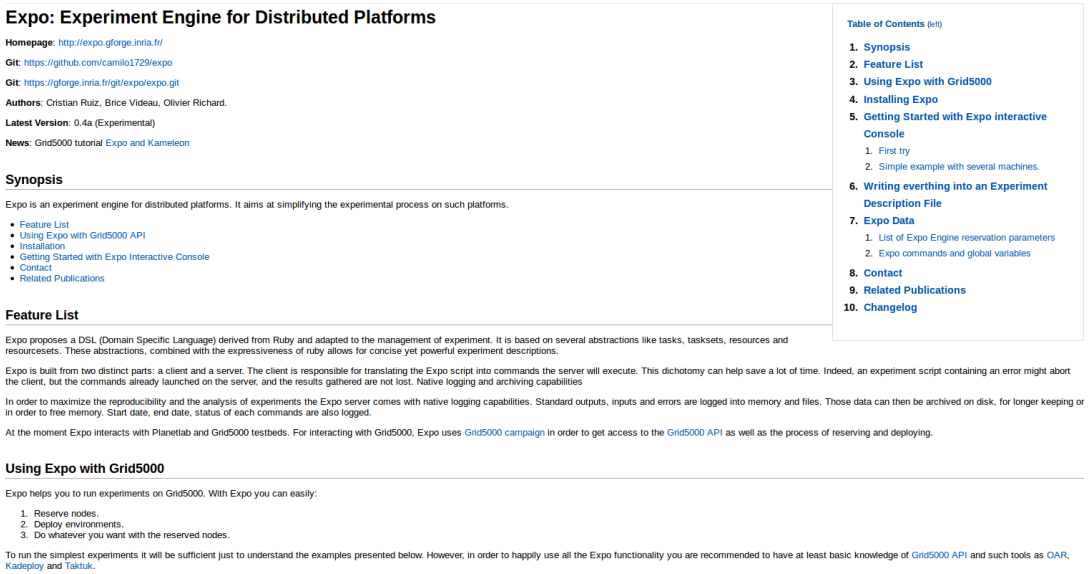
Conclusions

- Automating the experimentation process will allow others to repeat the experiment.
- The use of experiment tools will save user time, cost and will allow others to gain deeper insights.

Conclusions

- Experiment tools will foster the evaluation of systems under real conditions.
- It is important to encourage the culture of experiment reproducibility, which is acknowledged to be a shortcoming in computer experimentation.

Expo project page



Expo: Experiment Engine for Distributed Platforms

Homepage: http://expo.gforge.inria.fr/

Git: https://github.com/camilo1729/expo

Git: https://gforge.inria.fr/git/expo/expo.git

Authors: Cristian Ruiz, Brice Videau, Olivier Richard.

Latest Version: 0.4a (Experimental)

News: Grid5000 tutorial Expo and Kameleon

Synopsis

Expo is an experiment engine for distributed platforms. It aims at simplifying the experimental process on such platforms.

- Feature List
- Using Expo with Grid5000 API
- Installation
- Getting Started with Expo Interactive Console
- Contact
- Related Publications

Feature List

Expo proposes a DSL (Domain Specific Language) derived from Ruby and adapted to the management of experiment. It is based on several abstractions like tasks, tasksets, resources and resourcesets. These abstractions, combined with the expressiveness of ruby allows for concise yet powerful experiment descriptions.

Expo is built from two distinct parts: a client and a server. The client is responsible for translating the Expo script into commands the server will execute. This dichotomy can help save a lot of time. Indeed, an experiment script containing an error might abort the client, but the commands already launched on the server, and the results gathered are not lost. Native logging and archiving capabilities

In order to maximize the reproducibility and the analysis of experiments the Expo server comes with native logging capabilities. Standard outputs, inputs and errors are logged into memory and files. Those data can then be archived on disk, for longer keeping or in order to free memory. Start date, end date, status of each commands are also logged.

At the moment Expo interacts with Planetlab and Grid5000 testbeds. For interacting with Grid5000, Expo uses Grid5000 campaign in order to get access to the Grid5000 API as well as the process of reserving and deploying.

Using Expo with Grid5000

Expo helps you to run experiments on Grid5000. With Expo you can easily:

1. Reserve nodes.
2. Deploy environments.
3. Do whatever you want with the reserved nodes.

To run the simplest experiments it will be sufficient just to understand the examples presented below. However, in order to happily use all the Expo functionality you are recommended to have at least basic knowledge of Grid5000 API and such tools as OAR, Kadeploy and Taktuk.

Table of Contents (left)

1. Synopsis
2. Feature List
3. Using Expo with Grid5000
4. Installing Expo
5. Getting Started with Expo interactive Console
   1. First try
   2. Simple example with several machines.
6. Writing everthing into an Experiment Description File
7. Expo Data
   1. List of Expo Engine reservation parameters
   2. Expo commands and global variables
8. Contact
9. Related Publications
10. Changelog

Figure: `http://expo.gforge.inria.fr/`

The end

Thank you!
Questions?


Jeannie Albrecht, Christopher Tuttle, Ryan Braud, Darren Dao, Nikolay Topilski, Alex C. Snoeren, and Amin Vahdat.
Distributed application configuration, management, and visualization with plush.
ACM Trans. Internet Technol., 11(2):6:1–6:41, December 2011.


Eric Eide, Leigh Stoller, and Jay Lepreau.
An experimentation workbench for replayable networking research.
In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, NSDI'07, pages 16–16, Berkeley, CA, USA, 2007. USENIX Association.


Lorenzo Leonini, Étienne Rivière, and Pascal Felber.
Splay: distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze).

In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI'09, pages 185–198, Berkeley, CA, USA, 2009. USENIX Association.

Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar.
Omf: a control and management framework for networking testbeds.
SIGOPS Oper. Syst. Rev., 43(4):54–59, January 2010.

Yanyan Wang, Matthew J. Rutherford, Antonio Carzaniga, and Alexander L. Wolf.
Automating experimentation on distributed testbeds.
In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 164–173, New York, NY, USA, 2005. ACM.