Université Henri Poincaré, Nancy 1
ESIAL

Placement supervisor : Lucas Nussbaum (AlGorille)
Academic supervisor : Charles Despres (ESIAL)

Conducting experiments on a large-scale research testbed with a workflow management system

Internship report

AlGorille - LORIA **Yoanne Girardin** 10 septembre 2011

Thanks

I would like to thank Lucas NUSSBAUM for giving me the opportunity to work in the Al-Gorille team, Sebastien BADIA (aka Admin-staff) for his help on various system administration problems, Marion GUTHMULLER for this LaTeX template, and more generally the AlGorille team for their warm welcome.

I would also thank Charles DESPRES, my ESIAL tutor.

Table des matières

Introduction

When conducting experiments, it is very important to be able to reproduce it at will. In computer science, it is generally achieved by using something called ”scripts”. These little piece of program are executed line by line and are generally used for automating jobs. Scripts are very useful, but generally written in a ”dirty” way, using shortcuts to make it work as we would like to. In order to launch experiments with ease and in a ”clean” way, Lucas Nussbaum and François Charoy thought it could be a good idea to use the workflow formalism and a workflow engine, a technique already existing for other scientific domain like the bioinformatic.

To describe the eleven weeks of my internship, I first will talk about the LORIA and the AlGorille team, then I will explain what is a workflow, the BMPN and what is Ruote and how does it work. Finally I will present the work I have done on using Ruote to run experiments on Grid5000.

1 Work environment

1.1 The LORIA laboratory

LORIA, "*Laboratoire Lorrain de Recherche en Informatique et ses Applications*" (Lorraine Laboratory of IT Research and its Applications), is a "unité de recherche mixte (UMR)" shared by several establishments

– Loria
– CNRS, Centre National de Recherche Scientifique (National Centre of Scientific Research)
– INPL, Institut National Polytechnique de Lorraine (National Polytechnic Institute of Lorraine)
– INRIA, Institut National de Recherche en Informatique et en Automatique (National Research Institute for IT and Robotics)
– UHP, Henri Poincaré University, Nancy 1
– Nancy 2, Nancy 2 University

LORIA is a Laboratory of more than 450 individuals organized in :

– research team
  – more than 150 researchers and teaching-researchers
  – a third of doctorate students and post-doctorate
  – engineers
– supports services
  – engineers
  – technicians
  – administrative staffs

This also means :

– more than 40 industrial contracts underway
– more than 125 cooperation projects with more than 32 different countries
– more than 30 guests from varying countries each year

Missions

Fundamental and applied research in the field of information and communications sciences and technology on an international basis Training in Engineering colleges, the Universities of Nancy and Doctorate schools Transfer of technology *via* industrial partnerships, assistance in the creation of companies and a Club of partners

1.2 The AlGorille team

AlGorille stands for "Algorithmes pour la grille", or Algorithms for the grid. This team's domain is about Networking, System & Services and Distributed Computing. The main research subject treats of the distributed computing and high performance applications. Its work is based on three main axis :

– Transparent resource handling :
– Software structuration for scalability
– Experimental validation : reproducibility, extension possibility and simulation applicability, emulations and *in situ* experiments

AlGorille is composed of 6 permanent members/researchers, led by Jens GUSTEDT.

2 Context

2.1 Introduction

Doing an experiment on a computing grid such as Grid5000 is not a simple exercise. Dealing with multiple computers, gathering datas from everyone of them, and all of that in an automated way can be done easily with a few scripts. However, these scripts generally are written in such a manner it is nearly impossible to re-use them. Given that, Lucas Nussbaum (AlGorille) and François Charoy (Score) thought of using techniques and models coming from the industry and used by many researcher, and more especially the "workflow" system.

2.2 Grid5000

Grid'5000 is a scientific instrument for the study of large scale parallel and distributed systems. It aims at providing a highly reconfigurable, controlable and monitorable experimental platform to its users. The initial aim (circa 2003) was to reach 5000 processors in the platform. It has been reframed at 5000 cores, and was reached during winter 2008-2009.
Nineteen laboratories are involved in France with the objective of providing the community a testbed allowing experiments in all the software layers between the network protocols up to the applications.
The current plans are to extend from the 9 initial sites each with 100 to a thousand PCs, connected by the RENATER Education and Research Network to a bigger platform including a few sites outside France not necessarily connected through a dedicated network connection. Site in Luxembourg should join shortly, and Reims has now joined.
To manage Grid5000, a committee has been created : Aladin-G5K. It is in charge of developing and maintaining the Grid5000 architecture.

Figure 1 – Grid5000 cluster

2.3 Workflow and BPMN

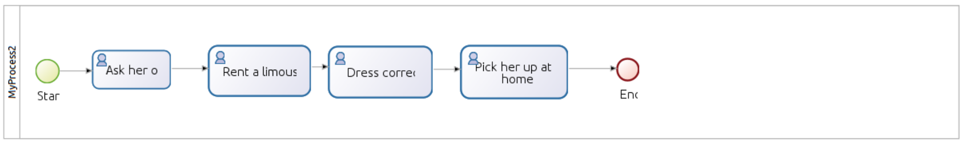
Workflows are widely used in the scientific domain. For instance, the website `www.myexperiment.org` allows any scientist to share his workflow with others. The business process model and notation is a specific type of workflow used to design business processes. It aims at make business process representation simple so both manager or analyst and technicians can understand it.

2.3.1 Workflow

A workflow is a way to represent a process. This process is decomposed in multiple tasks, each one of them being performed by a participant (human or machine). For instance, you can image the process ”ask a girl out at the prom' night” in a typical american soap movie. Then, decompose the task which have to be done in order (ask her out, choose correct clothing...). You finally end up with something like that :

Process Description

- Name : Ask a girl out
- Participant : you
- Tasks :
  - ask her out (of course, if this task fails, the game is over)
  - dress correctly
  - rent a limousine (eh...)
  - pick her up at her house



When you have to share informations across the tasks (and possibly different participant), you need a medium. In the workflow representation, this medium is called a **workitem**. It contains all the datas needed for the process to get to its end. Each tasks can get its inputs from a workitem or from one or more external sources, and then it can modify the workitem with the data it has computed. Let's see with an example.

Task : rent a limousine
Input data in workitem : amount of money in your piggy bank
Input data from the rental shop : costs of a limousine rental
Computation (tests if you have enough money, and then substract the cost from what you have)

The amount of money in the workitem has changed. If you add a task ”Rent a smoking” after your call to the car rental, this data can be reused later.

2.3.2 Business Process Model and Notation

For now, we are using just a few elements from the BPMN, the **flow objects** and the **connecting objects**. In these two class of objects we have all the elements needed to draw our workflow :

– event : represented with a circle, it denotes something that happens,

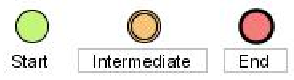


FIGURE 2 – source : wikipedia

– activity : represented with a rounded-corner rectangle, it describes the work that must be done,

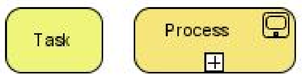


FIGURE 3 – source : wikipedia

– gateway : represented with a diamond shape, allow to fork or merge the path based on the condition expressed within the symbol,

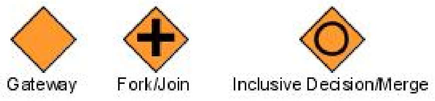


FIGURE 4 – source : wikipedia

– sequence flow : represented with a solid line and arrowhead, it shows in which order the activities will be performed.

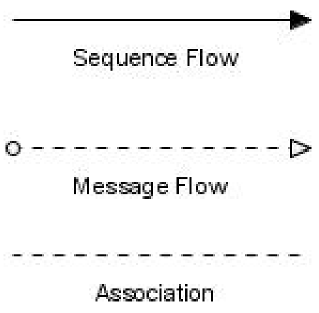


FIGURE 5 – source : wikipedia

Now that we know how to describe an experience and draw it on a whiteboard, we need a software to run it on a computer, a workflow engine. You could find many software on the internet, among one of them is Bonita and one of the developer is François CHAROY. Since the Grid5000 API is documented in Ruby, many users wrote Ruby scripts to launch their experiments. Hence, Ruote appeared to be the ideal workflow engine to use.

2.4 Ruote, a ruby workflow engine

A workflow engine is a software which gives the ability to create and execute modelled computer process. Most of the time, you have a nice graphical user interface to "draw" your workflow and some functions to run it. For instance, the software *Bonita*, an open source solution. But a real programmer want to be able to code his workflow ! That's why Ruote has been chosen.

```
#!/usr/bin/env ruby

require 'ruote'

engine = Ruote::Engine.new(Ruote::Worker.new(Ruote::HashStorage.new()))

engine.register_participant :alice do |workitem|
        workitem.fields['welcome'] = "Hi !!"
end

engine.register_participant :bob do |workitem|
        puts workitem.fields['welcome']
end

pdef = Ruote.define do
        participant :alice
        participant :bob
end

wfid = engine.launch(pdef)
engine.wait_for(wfid)
```

FIGURE 6 – Very basic Ruote example

Ruote is composed of four main components : engine, worker, storage and participant.

2.4.1 Engine

Before the 2.x version of Ruote, the worker and storage components were parts of the engine. Now that they are detached from it, a better name for the engine would be "dashboard". Its purpose is to launch a workflow. There are also methods for fixing issues with stalled processes or processes stuck in errors. An engine is bound to a storage. You cannot have two engines sharing the same storage.

2.4.2 Worker

A worker perform the job described in the workflow. It handles messages transiting *via* the storage and executes the action requested in them. It is more or less the core of the framework.

Those actions include "apply" (creating and applying a new flow expression), "reply" (replying to a [parent] flow expression), "cancel" (cancelling a flow expression, or a whole process instance). . . Basically, the worker performs each atomic action that take the workflow from its start to its end.

2.4.3 Storage

The storage is where the workflows are persisted. It's also where the message (that trigger the actions taken by workers) and the schedules (also triggering worker actions) are stored. Coming with the vanilla version of Ruote are two type of storage, HashStorage and FsStorage. While the first store everything into the RAM and should be use for testing purpose only, the second one use the hard drive and is persistant. There is a third "type" of storage, with special abilities : CompositeStorage. Its main purpose is to split the different messages within different storages. It could be used to store the workitem datas on a FsStorage, and thus making it persistant, and use a HashStorage for everything related to the execution of the flow.
An interface is also provided to let you implement your own storage. Moreover, it is possible to test it with a set of unit and functional tests already written. As of now, multiple types of storage have been implement, including Redis, couchDB, Sequel, MongoDB... All these storage have they're intrinsic pros and cons summed up in table 1.

Storage	Multiple workers	Remote worker	Speed	Comments
HashStorage	no	no	best	in-memory storage, limited to the current ruby process, totally transient
FsStorage	yes	no	2nd	hierarchy of JSON files, uses file locks to prevent collisions when multiple workers
Redis	yes	yes	1st	
Sequel	yes	yes	4th	
DM	yes	yes	5th	
couchDB	yes	yes	slowest	
MongoDB	no	yes	like redis	
BeanStalk	yes	yes	3rd	FsStorage based persistence with a Beanstalk front

TABLE 1 – source : ruote.rubyforge.com (including the weird speed values)

2.4.4 Participant

Participants are handed workitems by the Ruote engine and are expected to perform a task with them. Usually, some piece of information found in the payload of the workitem defines what/which task should be performed. As for the storage, there is different type of participant available, and you can add your own type.
In the BPMN formalism, a participant could be seen as a mix of an activity and the entity which perform the task.

2.4.4.1 BlockParticipant Really easy to implement. Even though it works just as perfect as other participants, it should only be used for tests purpose as the code is written directly within the script containing the engine definition.
As you could guess from its name, a BlockParticipant is described in a ruby block.

If you want your code to be more flexible, you should use a LocalParticipant instead.

```
engine.register_participant :alpha do |workitem|
        workitem.fields['message'] = "hello"
end
```

FIGURE 7 – A block participant

2.4.4.2 LocalParticipant It is called "local" because it has access to the storage. This abstract class provides methods to create your own type of participant. Among these methods, two are very important :

– consume(workitem) : this method allow you to program what this participant will do. Basically, this is where the job this participant has been called for is done. With an access to the workitem, you can gather any information you need, provided it has been given by other participants or at the initialization of the engine, or fill the workitem with any datas.
– reply_to_engine(workitem) : this helper hands the workitem back to the engine. Useful if you want your workflow to continue.

```
Class HelloParticipant
        include Ruote::LocalParticipant

        def consume(workitem)
                name = workitem.fields['name']
                puts "Hello #{name}"
                workitem.fields["#{name}_greeted"] = true
                reply_to_workitem(workitem)
        end
end
```

FIGURE 8 – A local participant

2.4.4.3 EngineParticipant This participant is a gateway to launch sub-processes on other engines. It works by giving the participant the connection information to the storage of the other engine. It is very useful in our context of distributed architecture.

```
master = Ruote::Engine.new(Ruote::Worker.new(Ruote::Redis::Storage.new(
                'host' => 127.0.0.1,
                'db' => 2,
                'thread_safe' => true,
                'engine_id' => 'master')))
slave = Ruote::Engine.new(Ruote::Worker.new( Ruote::Redis::Storage.new(
                'host' => 127.0.0.1,
                'db' => 3,
                'thread_safe' => true,
                'engine_id' => 'slave')))

master.register_participant(
  'slave',
  Ruote::EngineParticipant,
  'storage_class' => Ruote::Redis::Storage,
  'storage_args' => {
    'host' => '127.0.0.1',
    'db' => 3,
    'thread_safe' => true })
slave.register_participant(
  'master',
  Ruote::EngineParticipant,
  'storage_class' => Ruote::Redis::Storage,
  'storage_args' => {
    'host' => '127.0.0.1',
    'db' => 2,
    'thread_safe' => true })
```

Figure 9 – Two engine participants

In the figure 9, we declare two different engines using a Redis storage. It is important to notice that both have their own storage with the `'db' => number` line. Then, we register each one of them as participant within the other engine. From this moment, they can share informations *via* the workitem.

2.4.4.4 Other types There are some more predefined participants within the Ruote engine or written by other developers and then added to the list : SmtpParticipant to send mails, JigParticipant to interact *via* HTTP request. . . The documentation can be found at `http://ruote.rubyforge.org/part_implementations.html`

3 Contributions

My assignment was to test Ruote and see what could be done with it. I first had to program a small script representing a very basic experience. Once this had been done, I wrote a more complicated one which goal is to detect bug in the Grid5000 configuration. At the end of my internship, I began to look at how we could use Ruote with a distributed architecture.

3.1 A first experiment on Grid5000

I first had to familiarize myself with my environment by doing the tutorials available at `https://www.grid5000.fr/mediawiki/index.php/Category:Portal:Tutorial`. I discovered what is a distributed platform and the mechanisms to use it. As the API tutorials are documented in Ruby, I started to read books about that scripting language to understand the examples. Ruby is an object-oriented language and I re-used what I learned at ESIAL about such type of language to quickly understand the basic mechanisms of Ruby.

With all the mandatory knowledge to work with Grid5000, I moved on my very first experience with Ruote : run two benchmark tools on a few nodes of the grid and print the results on the screen. I read a lot of documentation on the Ruote website[1] and tried to understand how the few examples given work. That is where I started to discuss with John Mettraux, the maintainer of Ruote. He is available on IRC, or you can send questions on a mailing list. John is very dynamic and always answer as quickly as possible.

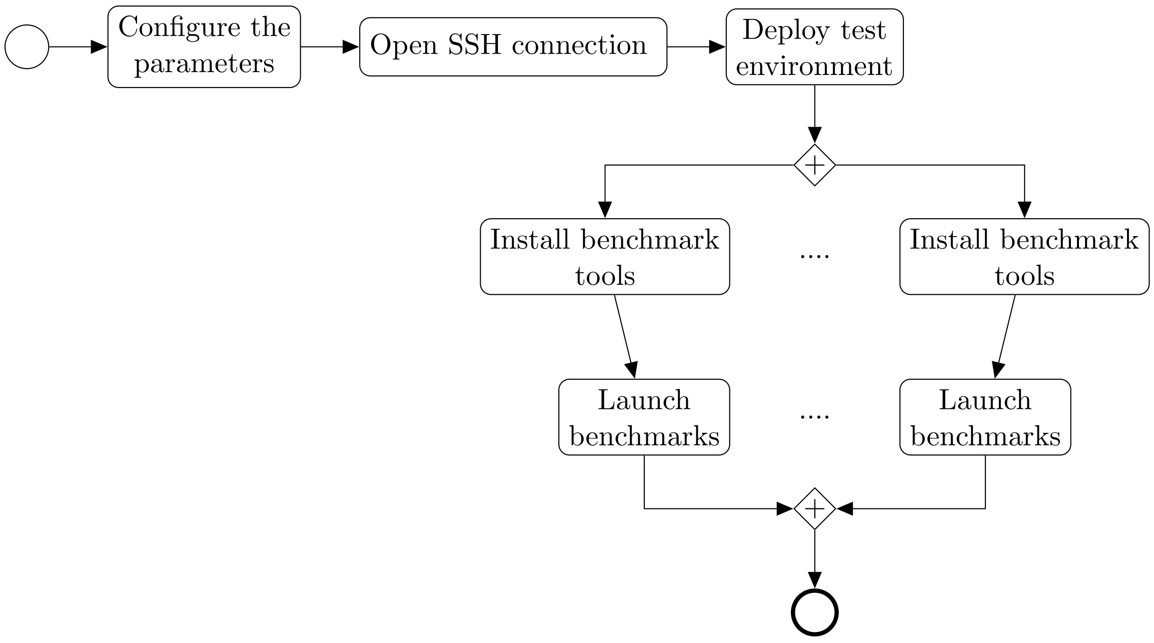


FIGURE 10 – Workflow of this first experiment

3.2 A real experiment : finding bugs on Grid5000 configuration

Grid5000 is a continually evolving project. For instance, during my internship, the Bordeaux's network has been completely refurbished. This lead to a constant bug tracking by the development team. The idea of this experiment was to launch a series of tests described in an old bug to help the aforementioned team resolving it.

I did 3 different versions of the script, modifying the way it handles the resource management. The first one without using the Grid5000 API, the second one using RESTfully, and finally with the common ruby restclient and calls to the API.

3.2.1 Goal of the experiment

This bug, codename "983", has been opened in January 2008. Its purpose is to see if we can configure a nodes with a special IP address and then ping some servers on every sites or use the local DNS. We added a few more tests to this : ability to mount a NFS drive, use of the proxy

1. http ://ruote.rubyforge.org

to get an access to Internet, configuring the clock with an NTP[2] request and finally doing a "cross-ping" from each node to every others.

3.2.2 Resource discovery using Grid5000 API

We wanted a script as generic as possible. To do so, the script should be able to discover how many resources are available at a given time on the grid. For instance, I had to reserve one node on each cluster. If tomorrow one or more cluster are added to Grid5000, one should not have to rewrite the script. This was not possible with the first version because I did not use the API.

The second step was to use RESTfully, a REST client wrote in ruby. I chose it because of the documentation available on the wiki, and because it was very easy to use. It wraps the REST request to the API into a nice and user-friendly ruby object. It can be used within a script by including the library, or interactively on the command line as an executable. It was very helpful while I was programing. I always had a shell opened with the RESTfully executable to test what I could do. Once I had found the correct syntax, I could simply transpose it into my script.
Being a ruby object was also the biggest problem of RESTfully in a Ruote environment. A workitem only accept plain text, thus making impossible to send a RESTfully object. This problem led to the last rewriting.

Grid5000 API is well documented, with a lot of examples. I mainly rewrite a few participants to adapt them with API calls. I also modified the workflow to allow a finer-grained control of the resource reservation.

3.2.3 Subnet handling

On each site, a network with a /14 netmask is allocated for experiments. Since a few months ago, it is possible to reserve a sub-network on Grid5000 with a software called *g5k-subnet*. It allows someone to use a specific network for his experimentation. G5k-subnet returns a /22 network nested within the virtualization network. I thought I had to configure the interfaces on my nodes with a /22 sub-network. After many hours of investigation, and with the help of Sebastien BADIA, one of the engineer of the team, we finally found that you do not reserve a /22 sub-network with the subsequent netmask, but you are allowed to use a portion of the virtualization network, and thus use a /14 netmask.

3.2.4 Interactions with the technical A-team

It is inevitable, when programming something to find bugs, to ask himself something : is this a bug I should report, or is this due to a flaw in my program ? Constantly working with the technical team, I made my way through empirical tests and discussion with the team to determine what kind of bug it was. They also helped me a lot with functionality I did not know how to use like NFS[3] or network interface configuration. Only a few misconfiguration were found and immediately reported. For instance, each site should have only one production network for all the clusters. But at Toulouse, there was two different networks, one for each cluster. Another example : the migration of the Nancy network to something cleaner (called "Golden rules" by the Grid5000 technical team) introduce a bug. A node withing the virtualization network could not ping a specific server also withing the virtualization network. After a quick investigation, the engineer in charge of the Nancy site found a problem in the configuration and fixed it.

2. http ://en.wikipedia.org/wiki/Network_Time_Protocol
3. http ://en.wikipedia.org/wiki/Network_File_System_*protocol*

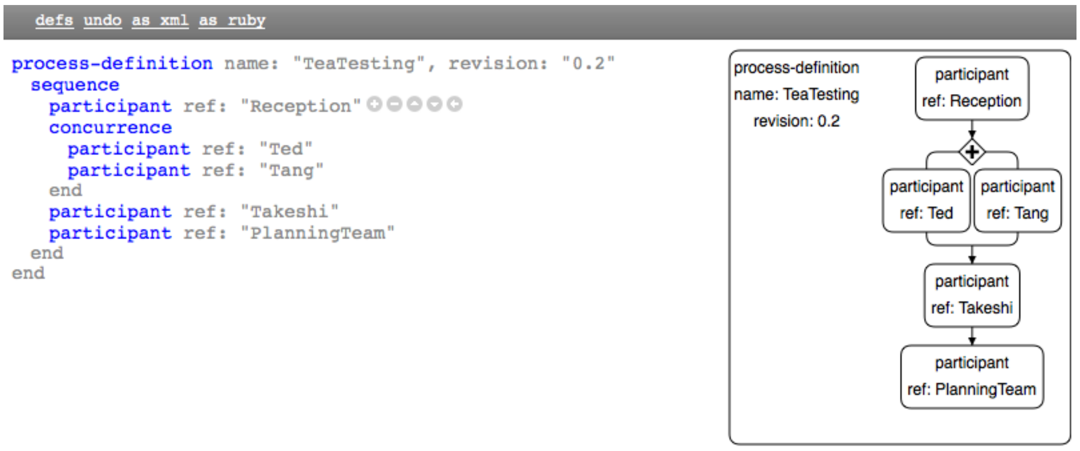
3.3 Ruote distributed

With a fully functional test working on my laptop, the next step has been to find a way to execute Ruote in a distributed way. Reading through the documentation, we found the Engine-Participant. This particular participant is used to register an engine in another one's participant list. It thus give the possibility to launch subprocess on another engine.The experiment on bug 983 was not resource consuming enough to need a distributed architecture, my laptop was sufficient. But we could imagine a test on a peer-to-peer software with hundreds of nodes involved.

After asking many questions to John, and reading the ruby documentation of Ruote class, I decided to try the architecture as follow : the main engine deal with the resource discovery and initialization of the workitem. Then, it determines a number of slave where the slave engines will be executed. The main engine copies the slave ruby script on every selected nodes, and launch with a SSH command. That is where the things are getting tricky. After several tests, it appears that a slave engine works well if I connect to the node *via* SSH and launch the command. But it doesn't work at all if the main engine does the same try to launch a slave with an SSH command. I did not have enough time to examine the cause of this bug.

3.4 Visualizing workflows

I have been asked to find a way to draw the workflow state while it is executed. Looking for hint in the documentation, I first it could be done by using some methods of the framework. Since a workflow is primarily designed for business processing, I search into the mailing list to see if somebody already thought about something like that. I finally found something called ”ruote-fluo”, developed by the one who is maintaining Ruote. However, it only gives someone the ability to edit a process or draw a process someone already wrote. It works as a browser-based service. After the installation, a web-service is launched on the port 4567. You can connect to it with your favourite browser and have access to an online process editor. You also can upload a process you created and see a graphical representation. Unfortunately, an image created from a complex workflow is a lot less clear to read (see Annex A).



```
process-definition name: "TeaTesting", revision: "0.2"
  sequence
    participant ref: "Reception"
    concurrence
      participant ref: "Ted"
      participant ref: "Tang"
    end
    participant ref: "Takeshi"
    participant ref: "PlanningTeam"
  end
end
```

FIGURE 11 – source : http ://jmettraux.wordpress.com

3.5 Conclusion on the work done

The goal of my internship was to discover if a workflow engine, Ruote in our case, could be used to conduct experimentations on Grid5000. After only a few tests, it appeared it could be done pretty much easily for simple experiment. It only took a few days to get used to the ruby language, the workflow engine and launch a quick test to see if the engine works as intended. Since Grid5000 is a testing environment for distributed applications, the main question was to know how we could gather the datas recovered from multiple parallel instance of participant. Using the documentation and with the help of John Mettraux, the developer of Ruote, we found a solution to our problem. From this time on, we were able to launch as many parallel processes as we want to, and still be able to gather the datas coming from each instance. To test this, Lucas asked me to write a script to help resolving an old bug. It took one month to come up with a functional solution albeit not bug free. The biggest part of the job has not been to design a workflow, but to correctly implement the different participants which test every aspect of the bug. Sebastien BADIA helped me a lot with problems related to system administration.

I did not have enough time to look deeper into the distributed mechanism of Ruote. I had a hard time trying to understand how it works. After two weeks of reading documentation, I asked the main developer of Ruote to help me. The problem was with the EngineParticipant part. I thought that in order to share their workitems, engines should have the same storage. In our case, it was the same Redis database. He explained me that an engine is bound to a storage. If two engines use the same one, they are considered as one big engine by Ruote. He also send me an example which I adapted on my laptop. Working on this part of the project will be the main axis of development. The next step would be to create Participant class in such a way it could be re-used as is by anyone working on Grid5000. For instance, the participant who reserve nodes. We could imagine to give him the information it needs from the workitem (how many cluster, how many nodes, how many nodes per cluster...). The only thing that someone who would use our participant would be to give datas with the correct formalism. Hence, it would not have to write its own class. The final step of the roadmap would be to visualize a workflow described in a Ruote script. A tool already exists to do this, but it does not works well with a process definition containing many options.

4 Conclusion

This project was a great experience for me, in various domain. First, I had to learn how to use Grid5000, and more specifically a middleware called OAR, which allows users to reserve computers (the "nodes") for a certain amount of time and kadeploy3 to deploy a testing environment on the nodes. The Grid5000 wiki has a great set of tutorials which guide you step by step. I also learned a new language never seen during my scolarship : Ruby. Using my former experience with other type of object-oriented languages such as Java, I was able to write small scripts quickly.

I had to do a weekly report to my tutor including what I did during the past week, and what were my objectives. This helped me a lot keeping track of what I had done, and see if I could follow the goal . It is very helpful right now as I am writing this report. In parallel, I have kept a text file where I have written every thought, every problem encountered, every solution found. In addition to the weekly report, I had to use git, a versioning system.

Finally, working in a research laboratory has been really exciting. I had the opportunity to attend to the PhD defense of Thibault Cholez, working on revocation mechanisms in dynamic networks. It gives me the taste for working in the research world.

A A complex ruote workflow as drawn by ruote-fluo

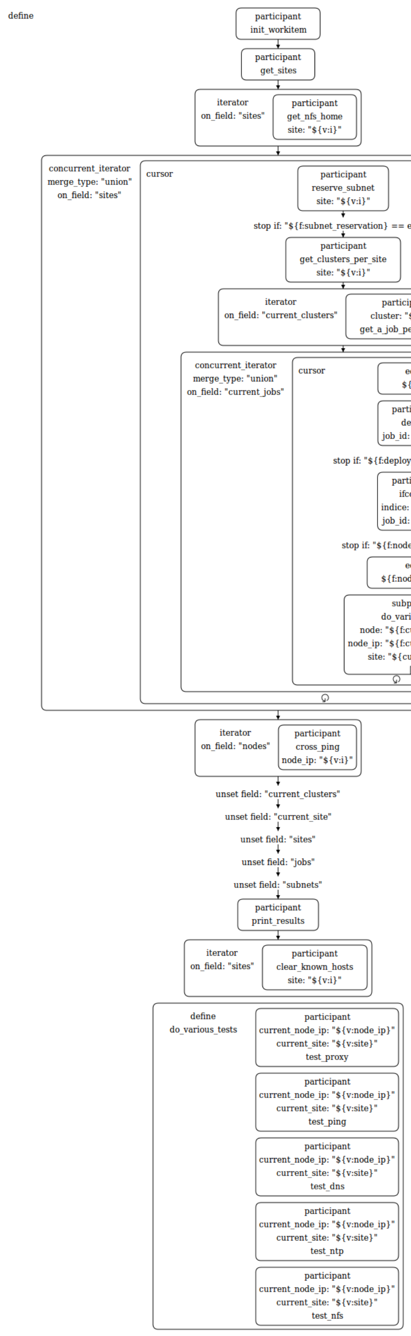


FIGURE 12 –