RAPPORT DE PIDR

Contrôle efficace d'un grand nombre de machines sur les grilles ou le Cloud avec SSH

Projet Interdisciplinaire de Découverte à la Recherche

Thibaud Detandt

Aurélien Nagel

Encadrant : Lucas Nussbaum

Remerciements

Nous remercions notre encadrant Lucas Nussbaum maître de conférences, ainsi que Emmanuel Jeanvoine ingénieur de recherche et Tomasz Buchert doctorant dans l'équipe de recherche ALGORILLE pour leurs conseils et leurs aides apportées tout au long du projet de PIDR. Nous remercions aussi le LORIA Laboratoire Lorrain de Recherche en Informatique et ses Applications pour avoir mis à disposition le matériel nécessaire à la bonne réalisation de notre projet ainsi qu'au bon déroulement de nos expérimentations. Finalement nous remercions l'ESIAL pour nous avoir proposé ce sujet de PIDR que nous avons trouvé intéressant à réaliser.

Sommaire

Glossaire

Dans ce rapport, tous les mots suivis d'une étoile (*) seront définis dans ce glossaire par ordre d'apparition. Tous les sigles ou les abréviations seront définis en note de bas de page.

1) Cluster

Le terme cluster désigne une technique consistant à regrouper plusieurs ordinateurs indépendants (appelés nœuds, node en anglais), afin de permettre une gestion globale et de dépasser les limitations d'un ordinateur.

2) Cloud

Le cloud computing est un concept qui consiste à déporter sur des serveurs distants des stockages et des traitements informatiques traditionnellement localisés sur des serveurs locaux ou sur le poste de l'utilisateur.

3) Taktuk

TakTuk est un outil pour exécuter une commande sur plusieurs nœuds en même temps. Il peut être utilisé par les utilisateurs une fois les nœuds en leur possession pour lancer leur script simultanément sur tous les nœuds réservés.

4) Perl

Perl est un langage de programmation reprenant des fonctionnalités du langage C et du shell. C'est un langage interprété, polyvalent, et particulièrement adapté au traitement et à la manipulation de fichiers texte, notamment du fait de l'intégration des expressions régulières dans la syntaxe même du langage.

5) Ruby

Ruby est un langage de programmation libre. Il est interprété, orienté objet. C'est un langage open-source dynamique qui met l'accent sur la simplicité et la productivité. Sa syntaxe élégante en facilite la lecture et l'écriture.

6) Python

Python est un langage de programmation multi-paradigme. Il favorise la programmation impérative structurée, et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl et Ruby.

7) Shell

Le shell est un logiciel fournissant une interface pour un utilisateur. Il est la partie la plus externe du système d'exploitation, c'est l'interface utilisateur du système d'exploitation.

8) Gnuplot

Gnuplot est un programme souple qui peut produire des représentations graphiques en deux ou trois dimensions de fonctions numériques ou de données.

9) Gateway

En informatique, une gateway est un dispositif permettant de relier deux réseaux informatique de types différents.

Introduction

Lorsqu'on utilise une plate-forme distribuée comme un cluster*, une grille, ou une infrastructure de Cloud* IaaS[1] (Amazon EC2 par exemple), il est souvent nécessaire de contrôler un grand nombre de nœuds (machines) simultanément, par exemple pour y lancer des commandes ou y copier des fichiers. Cela se fait généralement en utilisant les commandes ssh[2], scp[3] ou sftp[4]. Toutefois, lorsque le nombre de nœuds est important (plusieurs centaines ou milliers), utiliser ssh[2] de manière centralisée depuis une machine « maître » est peu efficace. Taktuk* (http://taktuk.gforge.inria.fr/) permet d'utiliser un système de connexions SSH hiérarchiques (les nœuds déjà contrôlés sont utilisés pour se connecter à d'autres nœuds), et fournit d'excellentes performances, mais il a quelques défauts : son interface utilisateur est peu pratique, et il n'est pas utilisable simplement depuis d'autres langages que Perl*. À l'opposé, les bibliothèques Ruby* net-ssh et Python* Paramiko permettent d'utiliser SSH directement depuis un script dans ces langages, mais ne proposent pas d'interface permettant de contrôler un grand nombre de machines.

Les objectifs de ce PIDR sont :

- d'évaluer l'écart de performances entre l'implémentation SSH « standard » en C, et les implémentations en Ruby et en Python
- d'ajouter un mécanisme à net-ssh ou à Paramiko permettant de se connecter de manière hiérarchique à un grand nombre de nœuds, en s'inspirant de ce qui est fait dans Taktuk
- de mesurer les performances de la solution développée en se comparant avec Taktuk

[1] Infrastructure as a Service
[2] Secure SHell
[3] Secure CoPy
[4] SSH File Transfer Protocol

I) Environnement de travail

1) Loria :

a) Présentation

Laboratoire Lorrain de Recherche en Informatique et ses Applications, est une Unité Mixte de Recherche (UMR) commune à plusieurs établissements :

- CNRS, Centre National de Recherche Scientifique
- INPL, Institut National Polytechnique de Lorraine
- INRIA, Institut National de Recherche en Informatique et en Automatique
- UHP, Université Henri Poincaré, Nancy 1
- Nancy 2, Université Nancy 2

La création de cette unité a été officialisée le 19 décembre 1997 par la signature du contrat quadriennal avec le Ministère de l'Éducation Nationale, de la Recherche et de la Technologie et par une convention entre les cinq partenaires. Cette unité, renouvelée en 2001, succède ainsi au CRIN (Centre de Recherche en Informatique de Nancy), et associe les équipes communes entre celui-ci et l'Unité de Recherche INRIA Lorraine.

Le LORIA compte 24 équipes de recherche, travaillant sur les 6 thématiques suivantes :

- Calculs, simulation et visualisation à haute performance
- Qualité et sûreté des logiciels
- Systèmes parallèles, distribués et communicants
- Modèles et algorithmes pour les sciences du vivant
- Traitement de la langue naturelle et communication multimodale
- Représentation et gestion des connaissances

Dans chacune des thématiques suivantes, le LORIA développe ses recherches au niveau international.

Ces équipes sont composé de 450 personnes parmi lesquelles, 150 chercheurs et enseignants-chercheurs, un tiers de doctorants et post doctorants, des ingénieurs, techniciens et personnels administratifs.

Mais c'est aussi chaque année :

- une trentaine de chercheurs étrangers invités
- des coopérations internationales avec des pays des cinq continents
- une quarantaine de contrats industriels

b) Missions

Les missions principales du LORIA sont :

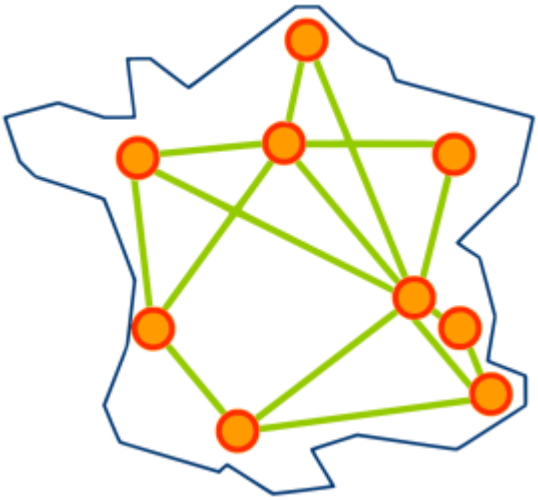
- la recherche fondamentale et appliquée au niveau international dans le domaine des Sciences et Technologies de l'Information et de la Communication
- la formation par la recherche en partenariat avec les Universités lorraines
- le transfert technologique par le biais de partenariats industriels et par l'aide à la création d'entreprises

2) Grid'5000

Afin de mener bien notre projet, nous avons utilisé la plate-forme Grid'5000. La plate-forme Grid'5000 est une infrastructure distribuée dédié à la recherche sur des systèmes parallélisés et distribués à grand échelle.

Cette plate-forme est constituée d'une grille de calcul de grande taille et a pour objectif d'atteindre le nombre symbolique de 5000 cœurs. Elle est utilisée dans le cadre de projets expérimentaux, dans le domaine de la recherche dans les systèmes informatiques distribués et à grande échelle.

Le principe est de se connecter via un réseau haut débit à une dizaine de grappes de PC de grande taille. Il est ainsi possible d'effectuer des tests réels plutôt que de ne présenter que des simulations.



Les sites français du Grid'5000

Chaque site héberge :

- un frontend, serveur permettant d'accéder aux clusters disponibles
- un serveur de données, pour centraliser les données utilisateurs
- un ou plusieurs clusters, c'est-à-dire des grappes de machines homogènes, appelées nodes.

Par exemple, le site de Nancy possède deux clusters :

graphene :

144 nœuds contenants :

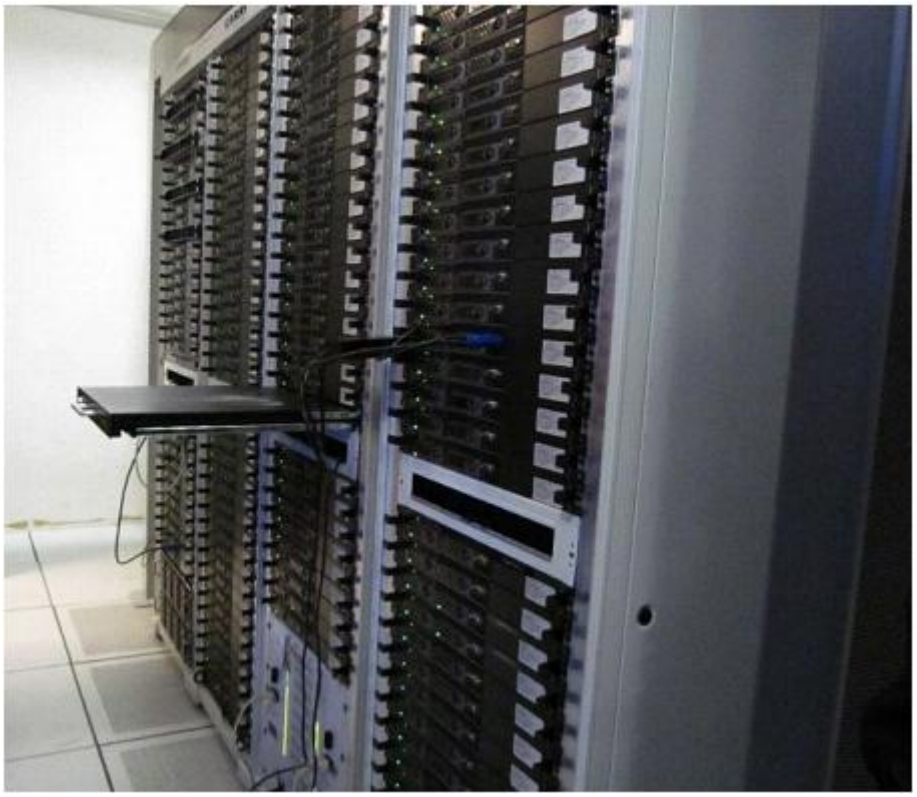
- 1 CPU Intel@2.53GHz
- 4 cores/CPU
- 16GB RAM
- 278GB DISK

griffon :

92 nœuds contenants :

- 2 CPUs Intel@2.5GHz
- 4 cores/CPU
- 16GB RAM
- 278GB DISK

Pour pouvoir utiliser les machines mise à disposition par la plate-forme Grid’5000, il faut tout d’abord se connecter en SSH au frontend d’un site. Ensuite pour accéder aux machines du site, il faut effectuer des réservations. On peut réserver des machines immédiatement ou pour une date voulu. Nous avons eu la chance de visiter la salle serveurs du site de Nancy située au Loria.



Armoires de serveurs de Nancy

II) Evaluation de performance

1) Prise en main des commandes de la plate-forme Grid'5000

La première étape de notre projet fut d'apprendre à utiliser la plate-forme Grid'5000. Pour cela notre tuteur de projet nous à conseillé d'effectuer les différents tutoriels qui se trouve sur le site principale de Grid'5000 dans la rubrique Tutorial : https://www.grid5000.fr/mediawiki/index.php/Category:Portal:Tutorial

Nous avons donc commencé par les « Starter Tutorials ». Ceci nous a tout d'abord appris quelques définitions de bases que l'on a trouvées à de nombreuse reprise par la suite de notre projet, comme par exemple la définition de « cluster », « job » et de « node ». Ensuite nous avons appris les différentes actions de bases permettant d'utiliser des nœuds.

La commande oarsub par exemple nous permet de réserver des nœuds sur un site (en créant un job). Voici un exemple d'utilisation d'oarsub, pour réserver 10 nœuds pendant 1 heure.

```
$ oarsub -I -l nodes=10, walltime=1
```

Nous pouvons aussi réserver des nœuds pour une date ultérieure en utilisant l'argument –r suivi d'une date au format YYYY-MM-DD HH :MM :SS

```
$ oarsub -I -l nodes=10, walltime=1 -r '2012-05-14 16:27:00'
```

Après cette réservation effectuée, oarsub ouvre un shell* dans lequel des variables d'environnements sont définies comme $OAR_FILE_NODE, qui contient la liste de tous les nœuds réservés $OAR_JOB_ID qui contient le numéro identifiant du job.

```
$ cat $OAR_FILE_NODES | uniq
griffon -78.nancy.grid5000.fr
griffon -81.nancy.grid5000.fr
griffon -91.nancy.grid5000.fr
```

```
$ echo $OAR_JOB_ID

389144
```

Si nous avons fini d'utiliser les nœuds et que le temps de réservation n'est pas fini, nous pouvons les libérer en utilisant la commande oardel pour permettre à d'autres personnes de les utiliser. La commande s'utilise de cette façon :

```
$ oardel 389144
```

Une fois l'environnement de travail bien maitrisé, nous sommes passé à la seconde étape de notre projet, c'est-à-dire l'apprentissage d'un nouvel langage de programmation, le Ruby*.

2) Premiers pas en Ruby

En effet un des objectifs principaux de notre sujet était de comparer la bibliothèque Ruby net-ssh qui permet d'utiliser SSH directement depuis un script à Taktuk qui permet quant à lui d'utiliser un très grand nombre de machines à la fois de manière hiérarchique.

Or ni l'un ni l'autre n'avons appris le langage Ruby par le passé. Ce fut pour nous une réelle opportunité d'apprendre un nouveau langage de programmation qui ne pourra qu'accroitre nos connaissances et améliorer notre CV. Notre tuteur Lucas Nussbaum nous a prêté un livre « Programming Ruby (2nd édition) : The Pragmatic Programmer's Guide » et nous a conseillé quelques sites web.

Pour nous améliorer et être encore plus à l'aise avec ce langage ainsi qu'avec les différentes bibliothèques disponibles par Ruby, nous avons décidé de réaliser un projet de spécialité totalement en Ruby. Il s'agit du projet de Cryptographie.

Dès que nous avions acquis quelques bases dans ce nouveau langage, nous avons pu réaliser un petit programme permettant de lancer une commande en parallèle sur les différentes machines que nous avions préalablement réservé, à l'aide de threads. Cette commande affichait tout simplement la date courante de chaque poste.

3) Tests de performances et Distem

Un des objectifs du projet était d'effectuer des tests entre les différentes solutions d'exécutions de commande à grande échelle. On a pu effectuer des tests pour Taktuk et la bibliothèque net-ssh. Mais il existe également d'autres solutions, paramiko et clustershell qui sont toutes les deux des bibliothèques python. En revanche par manque de temps nous n'avons pas effectué des tests sur ces deux dernières solutions.

Pour les tests, nous avons utilisé un outil développé par l'équipe, Distem[5] qui nous a permis de simuler 1000 nœuds virtuels en n'utilisant que 200 nœuds physiques. Pour l'utilisation de cet outil, nous avons eu l'aide de Tomasz Buchert, un doctorant de l'équipe qui nous a fournis et expliqué comment utiliser les différents scripts qu'il a développé pour utiliser Distem.

[5] DISTributed systems EMulator

Pour les mesures nous avons écrit trois scripts, un en bash[6] qui permet de lancer les deux autres plusieurs fois avec les différents paramètres que nous avons voulu tester, soit pour net-ssh le nombre de gateways* et la taille de la fenêtre (le nombre de connexion simultanée), pour Taktuk différents modes, le mode sans option particulier, un avec la propagation du code de Taktuk sur chaque nœud, un autre sans le dénombrement (ce qui nous empêche de savoir le nombre d'instance déployer correctement) et un dernier avec les deux modes précédents.

Les deux autres scripts ont été écrits en Ruby, le premier utilise la bibliothèque net-ssh et le deuxième fait un appel système à Taktuk. Ces deux scripts créent un fichier avec le temps mis pour exécuter la commande sur les différents nœuds qu'on leur a donnés en paramètre.

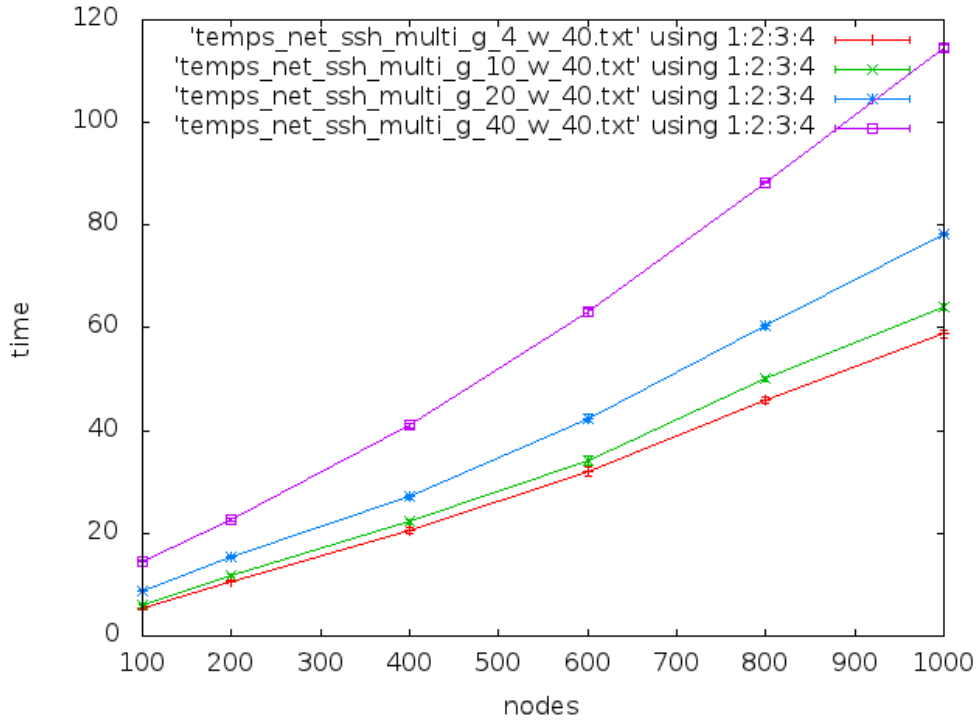
Ensuite pour traiter tous ces fichiers, nous avons également écrit un script qui permet de les rassembler selon leurs paramètres en un seul fichier avec la moyenne des valeurs ainsi qu'un intervalle de confiance à 95%.

Pour visualiser nos résultats sous forme de graphe nous avons utilisé gnuplot*, qui nous a permis d'afficher des intervalles de confiance assez facilement.

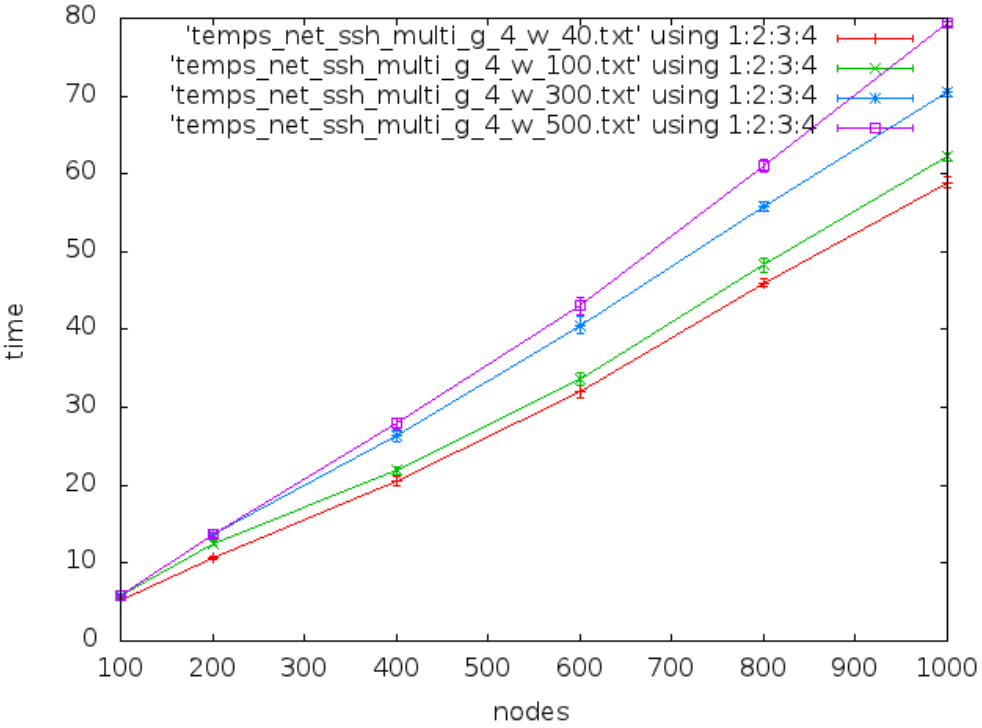
```
set terminal png
set xlabel "nodes"                          #nom de l'abscisse
set ylabel "time"                           # nom de l'ordonnée
set output "net_ssh_multi_g_04.png"         #fichier de sortie
plot 'temps_net_ssh_multi_g_4_w_40.txt' using 1:2:3:4 with yerrorlines,
'temps_net_ssh_multi_g_4_w_100.txt' using 1:2:3:4 with yerrorlines,
'temps_net_ssh_multi_g_4_w_300.txt' using 1:2:3:4 with yerrorlines,
'temps_net_ssh_multi_g_4_w_500.txt' using 1:2:3:4 with yerrorlines
```

Nous avions déjà fait des tests au préalable mais de manière moins rigoureuse, par conséquent nous savions déjà que Taktuk était beaucoup plus performant en termes de temps. Voici les résultats de notre expérience en utilisant net-ssh :

[6] Bourn-Again SHell

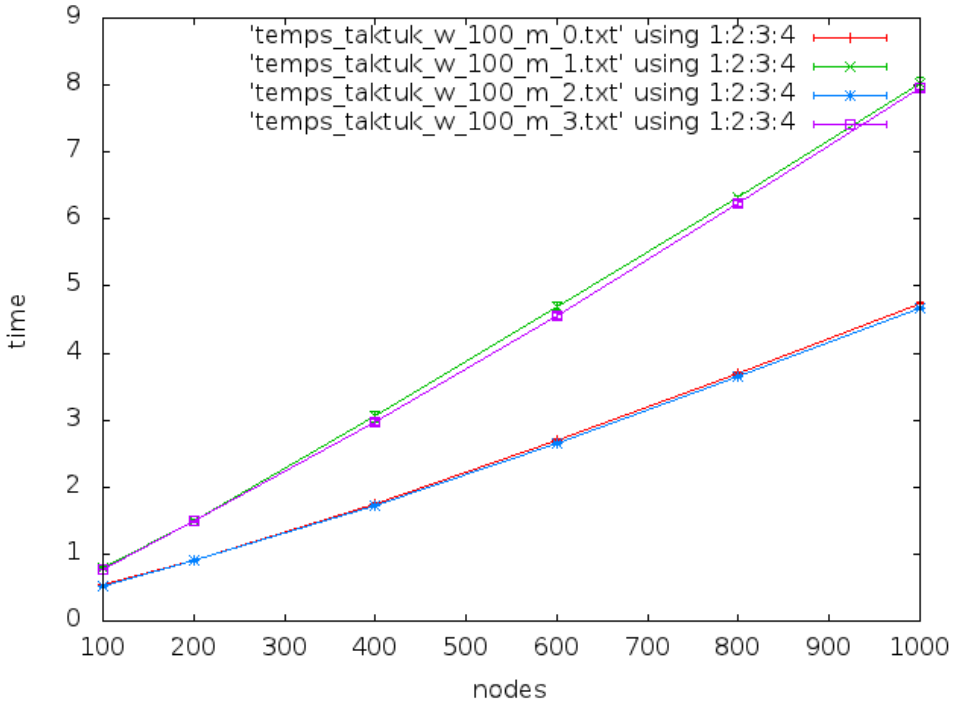


Sur ce graphe nous remarquons que l'utilisation de quatre gateways est le meilleur choix pour réduire le temps et cela peut importe le nombre de nœuds choisi. Nous avons par la suite effectué les tests sur la taille de fenêtre avec quatre gateways.

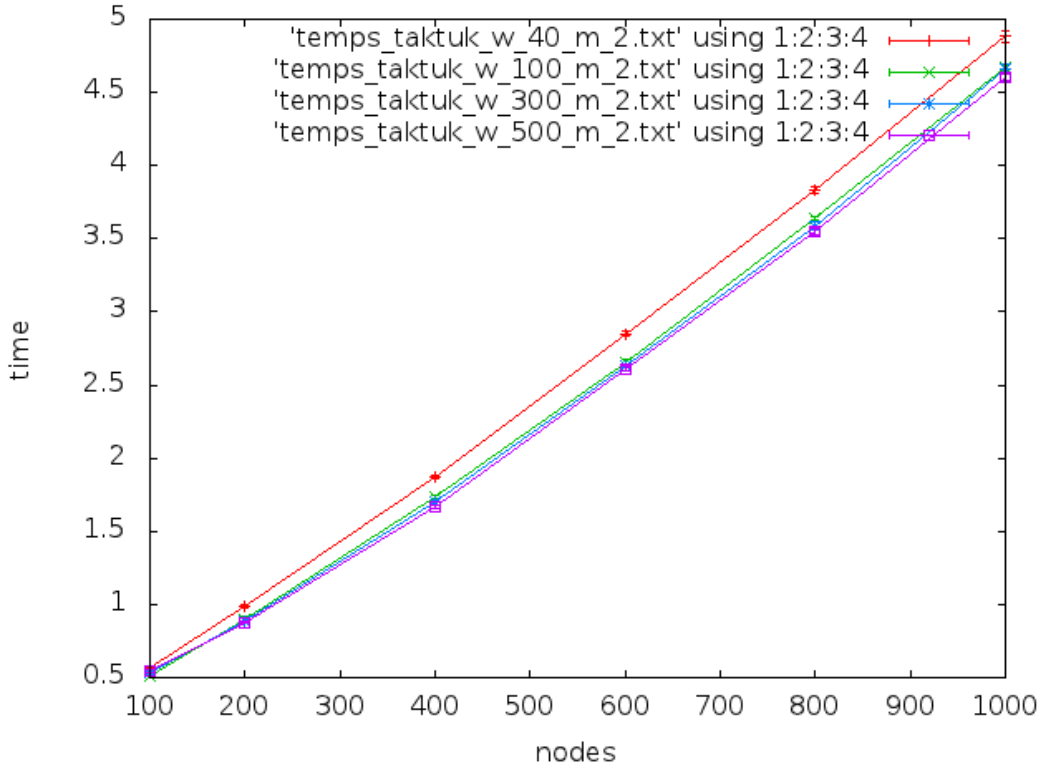


Sur le graphe précédent, l'utilisation d'une taille de fenêtre de 40 est optimale. En effet nous gagnons environ 20 secondes par rapport à une taille de fenêtre de 80.

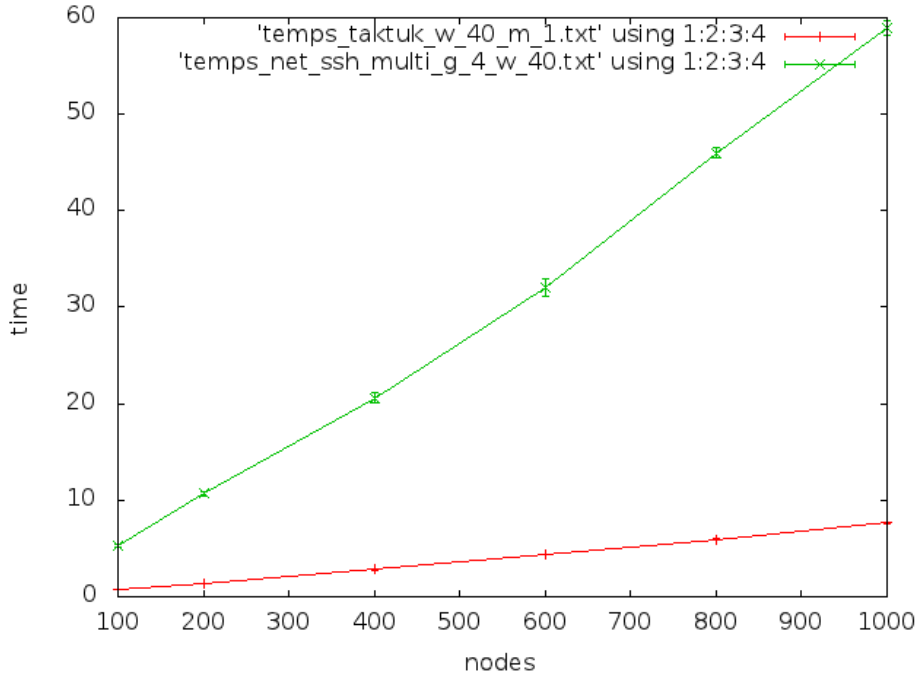
Une fois que nous avons trouvé les meilleurs paramètres pour net-ssh, il ne nous restait plus qu'à trouver les meilleurs paramètres pour Taktuk :



Premièrement nous devions choisir quel mode de Taktuk utiliser. En faisant varier les quatre modes décrits précédemment, nous nous sommes aperçus que le mode numéro deux (sans le dénombrement) était le plus performant.



Puis nous avions fait varier le nombre de fenêtre et sommes arrivé à la conclusion qu'une taille de fenêtre de 500 était le choix optimal.



Finalement nous avons comparé Taktuk et net-ssh et avons remarqué une nette différence de temps entre les deux outils. Néanmoins, net-ssh permet de conserver les connexions entre les nœuds ouvertes et est plus simple à utiliser.

III) L'API REST

Pour la fin de notre projet, nous avons eu à programmer une API[7] REST[8] qui est un style d'architectural notamment le style original du web. Cette API utilise des URLs[9] ainsi que l'HTTP[10] pour les différentes requêtes. Dans notre cas cette API permet de réserver et d'exécuter des commandes sur tous les nœuds en utilisant un client REST qui est disponible dans différents langages. Pour faire cette API, nous avons eu à utiliser une bibliothèque pour service web en Ruby qui s'appelle Sinatra. Avec cette bibliothèque on peut exécuter du code en utilisant des requêtes HTTP.

Dans notre cas on a trois requêtes possibles, la première qui nous connecte sur la plate-forme, la seconde qui nous permet de réserver les nœuds et la troisième pour lancer des commandes. En utilisant le format JSON[11] en tant que réponse aux requêtes, on arrive à avoir un tableau dans notre application cliente qui permet de récupérer le résultat d'une commande, le numéro de la session, un tableau des nœuds, etc…

[7] Application Programming Interface
[8] REpresentational State Transfert
[9] Uniforme Resource Locator
[10] HyperText Transfert Protocol
[11] JavaScript Object Notation

Nous avons aussi eu la chance d'assister à trois conférences au sein du LORIA. L'une d'entre elle a été présentée par un chercheur de l'équipe ALGORILLE à laquelle nous étions rattachés. Elle nous a permis de voir les travaux effectués au sein de l'équipe et l'aboutissement d'un projet de recherche complet. Une autre a été présenté par un chercheur de Grenoble qui travaille sur les mêmes sujets. Et une dernière par un chercheur de l'université du Luxembourg.

Conclusion

Le projet de PIDR que nous avons effectué au sein du LORIA et plus précisément dans l'équipe ALGORILLE s'est déroulé dans de très bonnes conditions. Ce stage nous a fait découvrir le monde de la recherche. Il sera, j'en suis sûr, très bénéfique pour notre future vie professionnelle.

Globalement, ce projet a été un très bon complément à notre formation d'informatique appliqué. Nous nous sommes vraiment aperçus de l'utilité et de l'importance des enseignements théoriques reçus à l'ESIAL. Nous avons en effet pu mettre en pratique de nombreux éléments appris lors de nos deux années à l'ESIAL. Les divers cours tels que le cours de Shell, la POO ou encore le C ont vraiment été une aide bénéfique pour la bonne réalisation de ce projet. Nous avons surtout dépassé nos appréhensions aux nouveaux langages à force de persévérance et d'aides diverses.
Ce projet nous a appris les différentes tâches quotidiennes d'un chercheur en informatique, nous avons en effet pu discuter plusieurs fois avec différents chercheurs au sein du LORIA à ce propos.

Ce projet nous a également permis d'acquérir des nouvelles compétences dans le domaine professionnel, notamment la connaissance d'autres langages informatique que ceux appris à l'ESIAL, et le suivi complet d'un projet professionnel de sa conceptualisation à sa réalisation finale. J'ai pu constater que la réflexion et l'analyse d'un projet en amont occupent une place très importante dans la réalisation de ce même projet. Chaque étape d'un projet doit être effectuée avec beaucoup de rigueur et dans une bonne organisation.

Finalement ce projet nous a aussi permis d'avoir un certain sens critique, une capacité rédactionnelle, et un esprit de synthèse qui nous aideront pour notre future vie professionnelle.

Bibliographie et Webographie

- Programming Ruby The Pragmatic Programmers (2nd Version)
  Edition The Pragmatic Programmers
  Auteurs: Dave Thomas avec Chad Fowler et Andy Hunt

- Sinatra Up and Running
  Edition O'REILLY
  Auteurs: Alan Haris et Konstantin Haase
  ISBN : 978-1-449-30423-2

- Site de collaboration pour les programmeurs : *http://www.stackoverflow.com*
  C'est un site en anglais sous forme de question réponse. Si vous avez une question dans n'importe quel langage de programmation, vous pouvez la poser et une autre personne pourra ainsi vous répondre

- Site du Grid'5000 : *https://www.grid5000.fr*
  Il s'agit du site officiel de la plate-forme grid'5000.

- Site de Sinatra : *http://www.sinatrarb.com/*
  Ce site contient toute la documentation nécessaire pour utiliser sinatra.

Annexes

- net_ssh_multi.rb

Il s'agit du script qui nous a permis de tester la bibliothèque net-ssh écrite en Ruby avec les différentes options qu'elle propose.

```
#!/usr/bin/ruby
require 'rubygems'
$: << "#{ENV['HOME']}/.gem/ruby/1.8/gems/net-ssh-multi-1.1/lib"
$: << "#{ENV['HOME']}/.gem/ruby/1.8/gems/net-ssh-gateway-1.1.0/lib"
$: << "#{ENV['HOME']}/.gem/ruby/1.8/gems/net-ssh-2.3.0/lib"
require 'net/ssh'
require 'net/ssh/multi'

if ARGV.length != 3
      puts "usage : net_ssh_multi.rb nombre_noeuds nombre_gateway
nombre_connexion"
      Process.exit
end
$tab_host = []
machine = File.open("machines",'r')
machine.each do |ligne|
        $tab_host << ligne.chomp
end


def connect(i,gateway,window)
      tdeb = Time.now
      Net::SSH::Multi.start(:concurrent_connections =>window) do |session|

            nbGateway = gateway
            if nbGateway != 0
                  session.via $tab_host[0].to_s, 'root', :paranoid =>
Net::SSH::Verifiers::Null.new
                  for k in nbGateway..i/nbGateway do
                        session.use $tab_host[k].to_s, :user => "root",
:paranoid => Net::SSH::Verifiers::Null.new, :via => nil
                  end

                  for j in 2..nbGateway do
                        session.via $tab_host[j-1].to_s, 'root', :paranoid
=> Net::SSH::Verifiers::Null.new
                        for k in i/nbGateway+1..i*j/nbGateway do
                              session.use $tab_host[k].to_s, :user =>
"root", :paranoid => Net::SSH::Verifiers::Null.new, :via => nil
                        end
                  end
            else
                  for k in 0..i do
                        session.use $tab_host[k].to_s, :user => "root",
:paranoid => Net::SSH::Verifiers::Null.new
                  end
            end
```

```
            session.exec "hostname"
      end
      tfin = Time.now
      tdiff = tfin -tdeb
      puts(tdiff.to_s)
      $resultat.write(tdiff.to_s+"\n")
end

nb_machines = ARGV[0].to_i
nb_gateways = ARGV[1].to_i
window_size = ARGV[2].to_i
nom =
"temps_net_ssh_multi_n_#{nb_machines}_g_#{nb_gateways}_w_#{window_size}.txt
"
$resultat = File.open(nom,"a")
connect(nb_machines, nb_gateways,window_size)
$resultat.close
```

- taktuk.rb

Il s'agit du script qui nous a permis de tester le programme Taktuk écrit en Perl avec les différentes options qu'il propose.

```
#!/usr/bin/ruby

$tab_host = []
machines=File.open("machines","r")
machines.each do |ligne|
      $tab_host << ligne.chomp
end

def proc(nb_machines,window_size,mode)
      puts("Debut")
      t = Time.now.to_f

      temp = File.open("_machines_temp","w")
      nb_machines.times do |node|
            temp.write($tab_host[node] + "\n")
      end
      temp.close
      if mode == 0 then
            system("taktuk -d -1 -w #{window_size} -l root -f
_machines_temp broadcast exec [ hostname ]")
      #self propagate
      elsif mode == 1 then
            system("taktuk -s -d -1 -w #{window_size} -l root -f
_machines_temp broadcast exec [ hostname ]")
      #no-numbering : This has the advantage of removing the global
synchronization occuring at the end of the deployment and making the
deployment more efficient.
      elsif mode == 2 then
            system("taktuk -n -d -1 -w #{window_size} -l root -f
_machines_temp broadcast exec [ hostname ]")
      #mode 1 et 2 en même temps
      elsif mode == 3 then
            system("taktuk -n -s -d -1 -w #{window_size} -l root -f
_machines_temp broadcast exec [ hostname ]")
      end
      t = Time.now.to_f - t
      myFile =
```

```
File.open("temps_taktuk_n_#{nb_machines}_w_#{window_size}_m_#{mode}.txt","a
")
      myFile.write(t.to_s + "\n")
      myFile.close
      puts("end")
end

proc(ARGV[0].to_i, ARGV[1].to_i,ARGV[2].to_i)
```

- net-ssh-taktuk_exp.sh

  Ce script bash nous permet de lancer les deux scripts au-dessus avec différents paramètres, plusieurs fois pour avoir plusieurs mesures et ainsi avoir des résultats qui se rapprochent plus de la réalité.

```
#!/bin/bash

for k in `seq 1 30`; do
      for n in 100 200 400 600 800 1000 ; do
            for w in 40 100 300 500 ; do
            taktuk -d -1 -w $w -l root -f /home/tdetandt/_machines_temp
broadcast exec [ hostname ] ;
                  for g in 4 10 20 40 ; do
                        ruby ../net_ssh_multi.rb $n $g $w ;
                  done
            done
      done
done
```

- average.rb

Ce programme traite les données récupérées. Il est expliqué plus précisément dans le rapport.

```
#!/bin/ruby

def average(array)
      sum = 0
      for t in array do
            sum = sum + t
      end
      average = sum / array.length
end

def ecart_type(tab)
      var = 0
      average = average(tab)
      for t in tab do
            var = var + (t - average)**2
      end
      ecart_type = Math.sqrt(var/tab.length)
end
```

```
def confidence_interval(array)
      tmp = ecart_type(array)/Math.sqrt(array.length)
      interval1 = average(array) - 1.96*tmp
      interval2 = average(array) + 1.96*tmp
      array = [interval1,interval2]
end

folder = ARGV[0]
if folder[folder.length-1..folder.length-1]=='/'
then folder = folder[0..folder.length-2]
end

files_ls = IO.popen("ls #{folder}/*.txt")
files_ls = files_ls.readlines
node_tab = []

files_ls.each do |file|
      file_name = file.chop.gsub(/_n_[0123456789]*/,'')
        index = file.chop.index('_n_')
        num_nodes = ""
        num__ = 0
        while( num__ <= 2 ) do
                if file[index..index]=="_"
                then    num__ = num__ + 1
                else
                        num_nodes = "#{num_nodes}#{file[index..index]}"
                end
                index = index + 1
        end
        num_nodes = num_nodes[1..num_nodes.chop.length].to_i
      if node_tab.count(num_nodes) == 0
      then node_tab << num_nodes
      end
end

node_tab.sort!
new_folder =""
node_tab.each do |num|
      files = IO.popen("ls #{folder}/*_n_#{num}_*")
      files = files.readlines
      for file in files do
            o_file = File.open("#{file[0..file.length-2]}")
            tab = []
            res = 0
            o_file.each_line{ |line|
                        tab << line.to_f
                        }
            interval = confidence_interval(tab)

            file_name = file.chop.gsub(/_n_[0123456789]*/,'')
            index = file.chop.index('_n_')
            num_nodes = ""
            num__ = 0
            while( num__ <= 2 ) do
                  if file[index..index]=="_"
                  then  num__ = num__ + 1
                  else
                        num_nodes = "#{num_nodes}#{file[index..index]}"
                  end
                  index = index + 1
            end
```

```
            num_nodes = num_nodes[1..num_nodes.chop.length]
            file_name = file_name[5..file_name.length-1]
            new_folder = "#{folder}_result"
            Dir.mkdir(new_folder) unless File.directory?(new_folder)
            new_file = File.open("#{new_folder}/#{file_name}",'a') do |f|
                  f.puts("#{num_nodes} #{average(tab)} #{interval[0]}
#{interval[1]}")
            end
      end
end

puts "your results are in \"#{new_folder}\""
```

Pour l'API REST, vous pouvez accéder au code source sur la page :
https://github.com/pidrgrid/pidr