DÉPARTEMENT INFORMATIQUE - IUT 2 GRENOBLE

Année Universitaire 2006-2007

MÉMOIRE DE STAGE

GESTION DE LA QUALITÉ DANS UNE GRILLE DE CALCUL EXPÉRIMENTALE

Laboratoire Informatique de Grenoble

(Stage du 2 avril au 15 juin 2007)

Présenté par

Arnaud Fontaine

Promotion 2A

Jury

IUT : Anne Lejeune et Éric Fontanet

Laboratoire : Lucas Nussbaum et Olivier Richard

Sommaire

Remerciements

Je tiens tout d'abord à remercier Lucas Nussbaum et Olivier Richard qui m'ont donné l'opportunité d'effectuer un stage au sein du Laboratoire Informatique de Grenoble (LIG). Je tiens aussi à les remercier pour leur encadrement et le temps qu'ils ont eu l'amabilité de me consacrer.

Je remercie également l'ensemble des membres du laboratoire ainsi que l'équipe de Grid'5000 pour leur accueil chaleureux, mais aussi pour les conseils et explications qu'ils ont su me fournir.

Introduction

Les ressources de calcul scientifique hautes performances, traditionnellement constituées de grappes de calcul, sont de plus en plus mises en commun au sein de grilles de calcul. Ces grilles de calcul, qui rassemblent des milliers de systèmes hétérogènes et distribués géographiquement, sont d'une complexité importante : s'assurer de leur bon fonctionnement et détecter de manière satisfaisante les pannes est un enjeu primordial et reste très difficile.

Le projet Grid'5000 a pour objectif la création d'une grille expérimentale, dédiée à l'étude des systèmes distribués. Elle est actuellement constituée de plus de 2500 processeurs, répartis sur une quinzaine de clusters dans 9 sites français. Mais le niveau de stabilité actuel de la plate-forme rend difficile la réalisation d'expériences complexes à grande échelle. De plus, les administrateurs disposent actuellement de peu d'outils pour détecter les pannes : la plupart du temps, ce sont les utilisateurs rencontrant des problèmes qui avertissent les administrateurs.

Mon stage effectué du 2 avril au 8 juin 2007 au LIG (Laboratoire Informatique de Grenoble) dans le cadre du projet Grid'5000 a consisté à développer des outils afin d'améliorer la stabilité de la plate-forme. J'ai tout d'abord commencé par étudier la plate-forme et les différents outils spécifiques. Cela m'a permis de vérifier que les documentations à propos de Grid'5000 sont à jour sur le site et de remonter quelques erreurs aux auteurs. Puis, j'ai écrit deux outils afin de détecter des problèmes sur la plate-forme.

En première partie, je vais tout d'abord présenter le laboratoire puis le projet Grid'5000. En seconde et troisième partie, je détaillerai les deux outils que j'ai réalisé, en donnant leur objectif puis en détaillant leur implémentation, leur fonctionnement et enfin les résultats obtenus grâce à ces outils.

Chapitre 1

Présentation du laboratoire

Le laboratoire ID (Information et Distribution) créé en 1999 a fusionné avec d'autres laboratoires pour créer le LIG (Laboratoire Informatique de Grenoble) en Janvier 2007. L'ancien laboratoire ID est maintenant l'antenne de Montbonnot du LIG, et emploie actuellement une soixantaine de personnes dont 20 permanents.

Le LIG est dirigé actuellement par Brigitte Plateau et financé par le CNRS (Centre National de la Recherche Scientifique), l'INPG (Institut National Polytechnique de Grenoble), l'UJF (Université Joseph Fourier), l'INRIA (Institut de Recherche en Informatique et en Automatique) ainsi que l'UPMF (Université Pierre Mendès France).

L'antenne de Montbonnot du LIG est constituée de deux équipes qui travaillent actuellement sur les axes de recherche suivants :

- **programmation parallèle** : algorithmes et techniques de programmation pour les calculs de haute performance ;
- **grappes et grilles** : architectures, services de base et intergiciels pour l'exploitation efficace et fiable des grappes et grilles ;
- **évaluation de performances** : modèles et logiciels pour l'étude et l'analyse de systèmes parallèles, distribuées et à grande échelle ;
- **optimisation et ordonnancement** : problèmes difficiles pouvant tirer parti du parallélisme, les modèles et les stratégies d'ordonnancement.

Chapitre 2

Grilles de calcul et Grid'5000

Dans cette partie, les grilles de calcul seront présentées puis Grid'5000 ainsi que ses caractéristiques.

2.1 Grilles de calcul

Avant de présenter Grid'5000, il est nécessaire de définir les notions de grille de calcul (ou grille de grappes) et de cluster (ou grappe). Un cluster rassemble un ensemble d'ordinateurs et possède les caractéristiques suivantes :

- **homogène** car les machines du cluster, communément appelées noeuds, ont une configuration matérielle similaire ;
- **localisé** car les machines sont reliées à travers un réseau local (LAN).

Une grille de calcul rassemble des clusters répartis sur plusieurs sites distants géographiquement et possède les caractéristiques suivantes :

- **hétérogène** car les différents clusters ne possèdent pas forcément les même caractéristiques techniques ;
- **distribué** car les différents clusters sont généralement distants géographiquement.

L'objectif des grilles de calcul est de fournir une puissance de calcul équivalente à celle des super-calculateurs en favorisant le coût et la modularité. En effet, un super-calculateur revient très cher alors qu'une grille de calcul est constituée de machines moins onéreuses mais représentant dans l'ensemble une puissance de calcul équivalente. L'utilisation de ce type de systèmes est de plus en plus importante dans la communauté scientifique, par exemple en physique atomique à des fins de simulation mais aussi en aéronautique, etc.

Ainsi, le LCG (LHC Computing Project) actuellement mis en place au CERN deviendra l'une des grilles possédant la plus importante puissance de calcul au monde et permettra à des physiciens d'effectuer des simulations et de traiter un nombre très important de données provenant d'accélérateurs de particules, afin par exemple de découvrir de nouvelles particules (figure 2.1).

2.2 Grid'5000

Grid'5000 a pour objectif de mettre en place une grille de calcul à grande échelle à des fins de recherche en informatique.

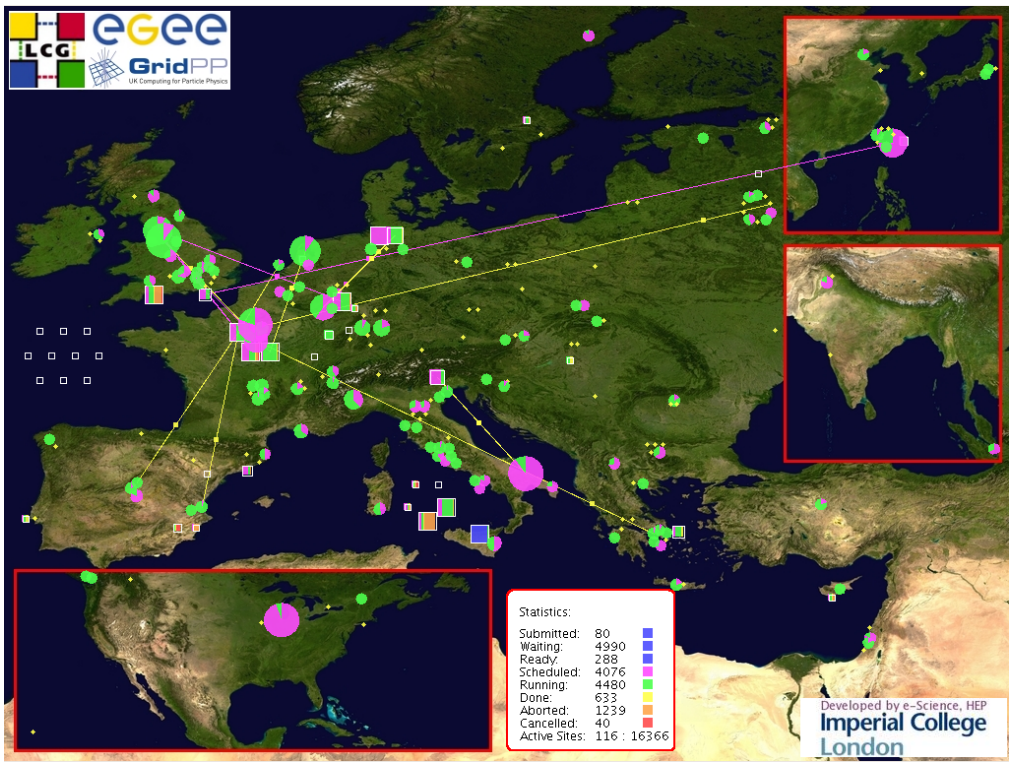

FIG. 2.1 – Grille de calcul LCG

2.2.1 Architecture de Grid'5000

Grid'5000 est une grille de calcul dont les clusters sont répartis sur les neuf sites suivants (figure 2.2) : Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia et Toulouse.

Chaque site comprend un ou plusieurs clusters. Actuellement, Grid'5000 dispose d'un total de 1500 noeuds, chacun composé de deux à quatre coeurs[1] (la plupart des noeuds sont bi-processeurs, certains processeurs contenant jusqu'à deux coeurs). Au total, il y a donc environ 3500 coeurs. À l'exception d'un des clusters de Grenoble encore en 32 bits, tous les processeurs fonctionnent en 64 bits. On trouve ainsi des processeurs reposant sur l'architecture X86-64 (AMD OPTERON, INTEL XEON), mais aussi sur IBM POWERPC et INTEL ITANIUM2. Les clusters vont d'une trentaine de noeuds (figure 2.3) à 350 noeuds.

La connexion réseau entre les clusters est assurée par RENATER (Réseau National de télécommunications pour la Technologie, l'Enseignement et la Recherche). Initialement, les clusters étaient connectés entre eux en 1Gbit/s, mais disposent actuellement d'un débit allant jusqu'à 10Gbit/s (figure 2.4).

Les machines composant Grid'5000 utilisent GNU/LINUX. La plupart des outils utilisés sont standards (SSH[2],

[1] Permet de combiner un ou plusieurs processeurs en une seule puce permettant d'augmenter la puissance de calcul sans augmenter la fréquence du processeur et permet de réduire la dissipation thermique.

[2] Programme informatique et protocole de communication sécurisé, particulièrement utilisés pour l'administration de machines à distance.

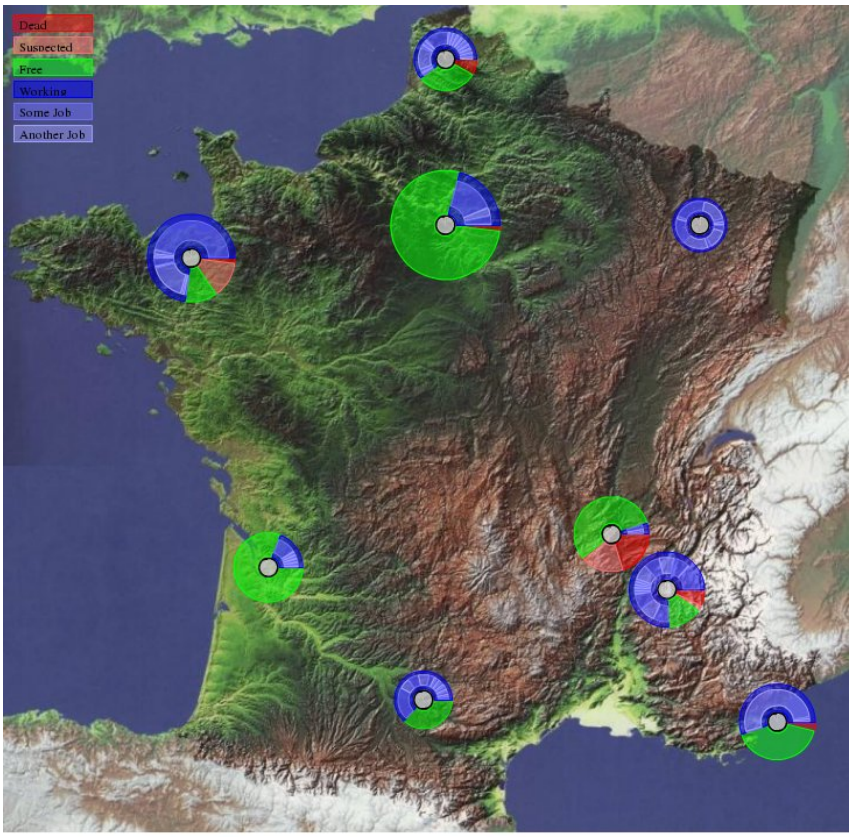

FIG. 2.2 – Répartition des noeuds et leur état sur la France

NFS[3], LDAP[4], etc.). De plus, des outils spécifiques ont été développés par les laboratoires participant au projet afin de pouvoir exploiter la grille de calcul, notamment OAR et KADEPLOY présentés dans les sections suivantes. Chaque site dispose d'un serveur NFS commun à ses clusters permettant de disposer de son espace utilisateur quelque soit le cluster du site.

2.2.2 Réservation de ressources (OAR)

Afin de pouvoir exécuter une tâche (ou *job*) sur une grille de calcul, il est nécessaire de passer par un gestionnaire de ressources servant à ordonnancer les tâches qui vont être exécutées par les différents utilisateurs. OAR, développé à Grenoble, est le gestionnaire de ressources utilisé sur Grid'5000. Il utilise un système de type *batch*. Ce qui signifie que la gestion des tâches est organisée en file d'attente, ainsi, chaque tâche doit se terminer avant qu'une autre débute. Il est libre, écrit en utilisant le langage de programmation PERL et nécessite une base de données MYSQL.

D'un point de vue utilisateur, il est possible de réserver sur OAR un ensemble de noeuds pour une tâche donnée

[3]Système de fichiers en réseau permettant, au niveau de Grid'5000, de rendre disponible les répertoires utilisateurs sur les différents clusters d'un même site.

[4]*Lightweight Directory Access Protocol* - protocole permettant l'interrogation et la modification des services d'annuaire (généralement gestion de comptes utilisateur).

FIG. 2.3 – Photo d'un des clusters de Sophia

grâce à la commande `oarsub`. Si on ne précise pas l'heure et la date d'exécution, alors `oarsub` exécutera la tâche le plus tôt possible. Cet outil renvoie l'identifiant de la tâche. Il dispose de deux modes :

- **interactif** permettant d'obtenir un *shell* sur la ressource réservée ;
- **passif** permettant de lancer un script précisé en paramètre de la commande.

OAR permet aussi gérer l'état de disponibilité des noeuds :

- **free** signifie que le noeud est disponible ;
- **job** signifie que le noeud est occupé par une tâche ;
- **absent** est spécifié par l'administrateur ou automatiquement avant un redémarrage du noeud lors d'un déploiement ;
- **suspected** signifie que OAR a détecté un problème sur ce noeud ;
- **dead** est un état mis par l'administrateur et spécifiant que le noeud pose problème actuellement.

Il existe un grand nombre d'alternatives à OAR : PBS/OPENPBS, TORQUE, MAUI, etc. OAR est un clone de OPENPBS reprenant 80% des fonctionnalités de ses fonctionnalités pour 20% du nombre de lignes de code. OAR repose sur une conception beaucoup moins complexe que OPENPBS et est plus performant.

Actuellement, la version *1.x* de OAR est déployée sur l'ensemble de Grid'5000, mais la version *2.x* devrait l'être très bientôt et propose de nouvelles fonctionnalités.

En complément de OAR, il existe différents outils de visualisation permettant de connaître l'utilisation des clusters (figure 2.5) ainsi que l'ordonnancement des tâches exécutées à l'aide d'un diagramme de Gantt (figure 2.6).

Utilisation de OAR

Dans le cas le plus simple où l'on souhaiterait réserver un noeud afin d'exécuter intéractivement une commande, on effectuerait les étapees étapes suivantes :

- choix du cluster ;

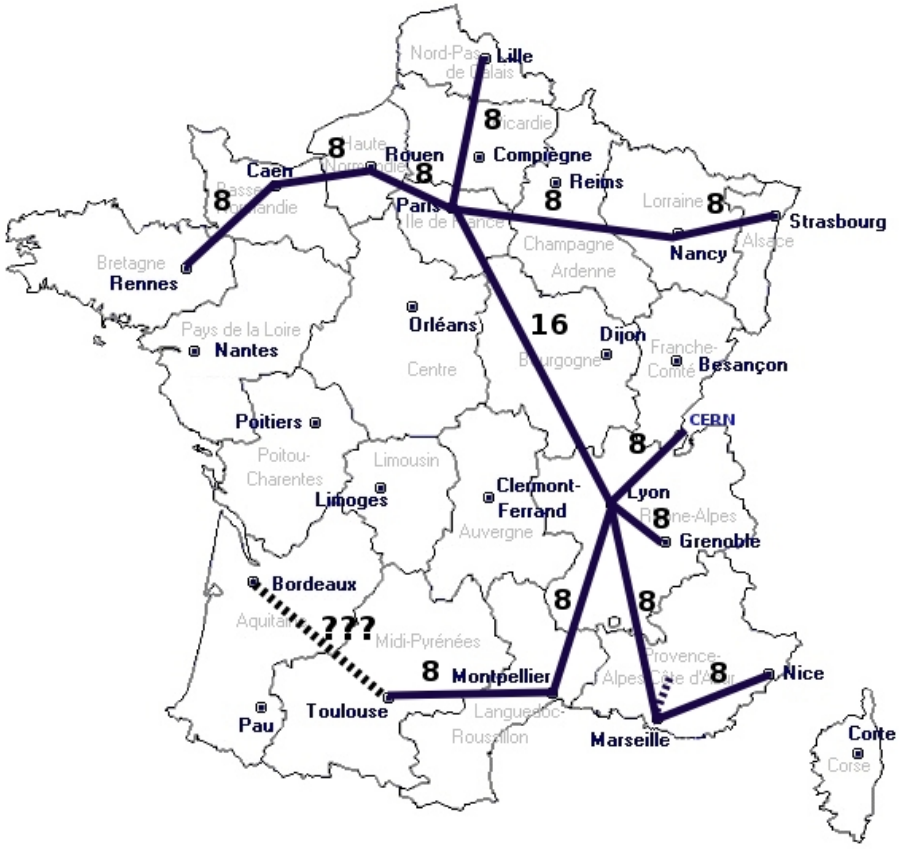

FIG. 2.4 – RENATER-4 (liens en 10Gb/s)

Cluster Name	azur	bordeaux	capricorne	gdx	grillon	helios	icluster2	idpot	paraquad	parasol	paravent	sagittaire	sol	tartopom	toulouse	all
Load																
Site	Sophia	Bordeaux	Lyon	Orsay	Nancy	Sophia	Grenoble	Grenoble	Rennes	Rennes	Rennes	Lyon	Sophia	Rennes	Toulouse	
Type	opteron	opteron	opteron	opteron	opteron	opteron	Itanium2	xeon	xeon	opteron	opteron	opteron	opteron	g5	opteron	
Free Nodes	1	31	0	113	109	50	0	3	25	6	54	1	5	30	0	**428**
Busy Nodes	71	65	32	224	56	6	85	17	39	56	42	66	45	0	56	**860**
All Nodes	74	99	56	342	166	56	103	25	64	64	99	69	50	32	57	**1356**

FIG. 2.5 – Statuts d'utilisation de Grid'5000 en utilisant l'outil MONIKA

- connexion par SSH au frontal[5] (il y a généralement un frontal par site) permettant d'utiliser OAR afin de réserver un ensemble de noeuds du cluster sélectionné ;
- utilisation de l'outil OARSUB afin de réserver un noeud via le mode interactif.

[5] c.-à-d. les machines des différents clusters sur lesquelles on peut se connecter pour réserver les noeuds.

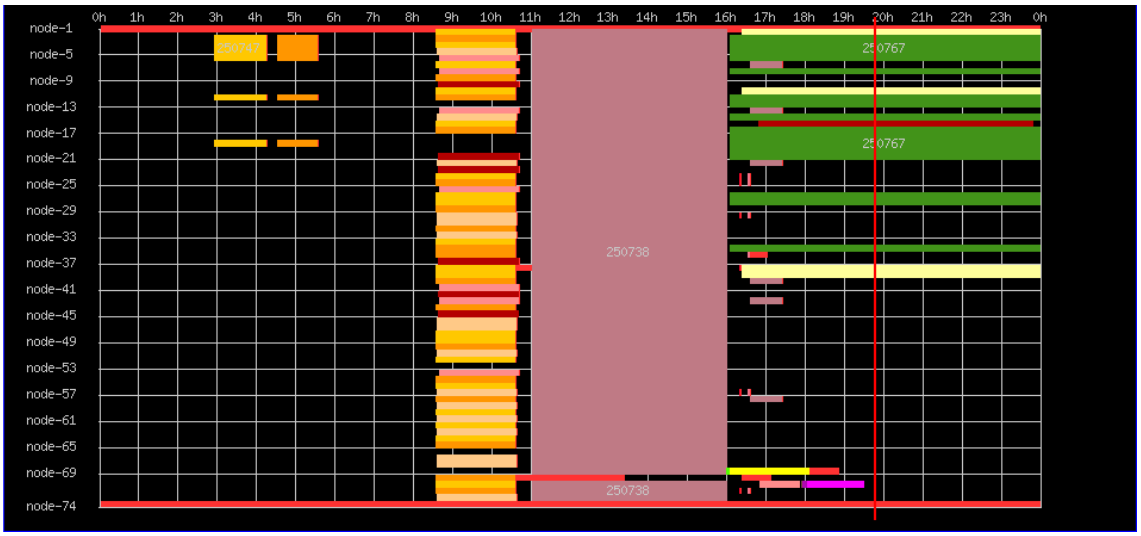

FIG. 2.6 – Ordonnancement des tâches d'un cluster de Sophia en utilisant l'outil DRAWGANTT

2.2.3 Déploiement d'environnements (KADEPLOY)

Les noeuds permettent d'exécuter une tâche et disposent d'un ensemble de logiciels pré-installés permettant d'effectuer la plupart des expérimentations. Cet environnement est appelé **environnement de référence**. Cependant, pour certaines expérimentations, il est indispensable de pouvoir disposer de logiciels supplémentaires ou d'avoir les droits administrateurs.

KADEPLOY est un outil en ligne de commande sous licence GNU/GPL développé par le laboratoire ID-IMAG. Il permet aux utilisateurs de Grid'5000 de déployer des environnements personnalisés afin d'effectuer des expérimentations nécessitant une configuration spécifique (installation et lancement de programmes nécessitant un accès administrateur, etc.) ou encore un système d'exploitation spécifique tels que *BSD, SOLARIS ou une distribution GNU/LINUX spécifique.

Le déploiement s'effectue en plusieurs étapes (figure 2.7) une fois les noeuds réservés via OAR. Chaque noeud dispose en plus de la partition pour l'environnement de référence d'une partition réservée aux déploiements et pour certains clusters d'une partition de taille plus grande et permettant d'y effectuer ce que l'on souhaite.

KADEPLOY dispose d'une base de données listant les environnements déployables. Ces environnements, généralement basés sur GNU/LINUX, permettent de disposer d'un système minimal à un système plus complet (NFS, LDAP, etc.). À partir de ce système, l'utilisateur a ensuite la possibilité de le configurer comme il le souhaite pour son expérimentation car il dispose d'un accès administrateur. Il est également possible de créer son propre environnement puis de l'enregistrer dans la base de données KADEPLOY afin de l'utiliser ensuite pour des déploiements.

Il existe d'autres outils que KADEPLOY permettant d'installer un système sur un ensemble de machines à la fois, par exemple FAI (Fully Automatic Installation). Cependant ce dernier dépend d'un serveur que chaque noeud va solliciter pour les déploiements, KADEPLOY est conçu pour permettre le déploiement d'un grand nombre de noeuds (figure 2.8). En effet, l'exécution des commandes s'effectue suivant un arbre de noeud permettant donc de réduire la complexité des déploiements. Ainsi, un déploiement sur l'ensemble des noeuds de `gdx`, soit 300 noeuds, prend environ 9 minutes en utilisant KADEPLOY.

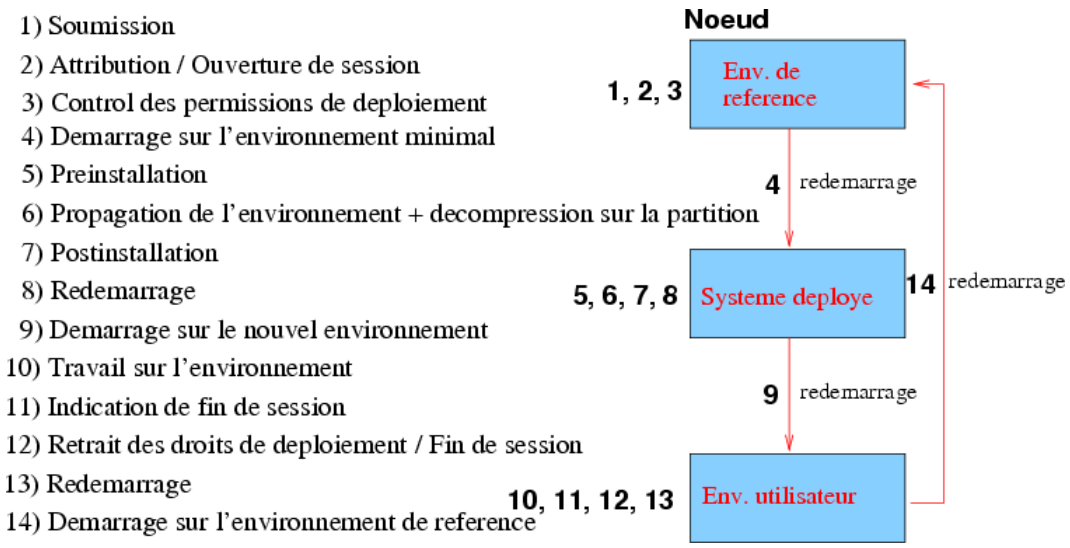

FIG. 2.7 – Étapes d'installation d'un noeud via KADEPLOY

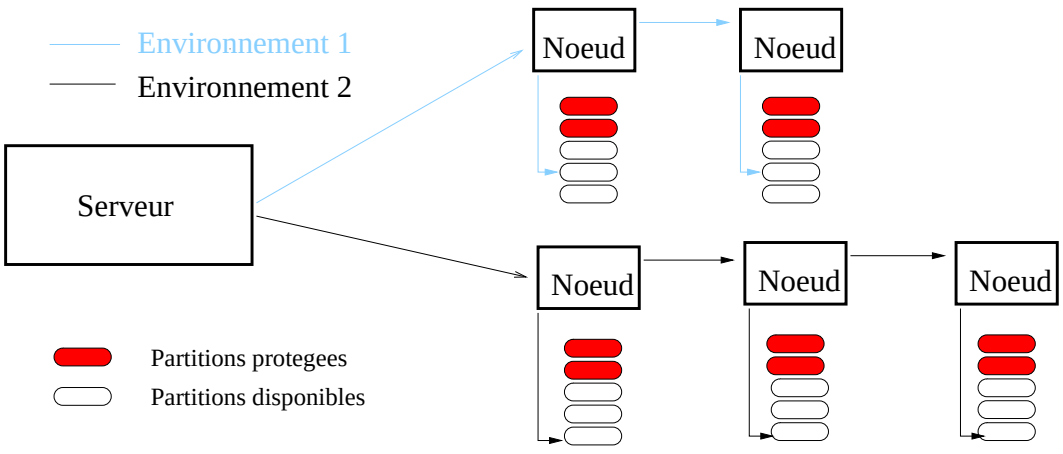

FIG. 2.8 – Déploiements concurrents avec KADEPLOY

Chapitre 3

Tests sur KADEPLOY

La première partie du stage a consisté en l'écriture d'un script destiné à tester KADEPLOY. Ce script a été écrit en utilisant le langage de programmation RUBY.

3.1 Objectifs

Tout d'abord, ce script a pour objectif de **détecter les problèmes inhérent à la nouvelle version de KADEPLOY**. En effet, il est prévu de déployer la nouvelle version de KADEPLOY très bientôt, et donc il faut s'assurer qu'il n'y a pas eu de régressions par rapport à la version actuellement installé sur la plupart des sites. Afin de pouvoir tester cette nouvelle version de KADEPLOY, le développeur de KADEPLOY a mis en place la version en développement sur le site d'Orsay.

Ce script a également comme objectif de **détecter les noeuds pouvant poser problème** tant au niveau matériel qu'au niveau des environnements de références.

Il est aussi destiné à **mettre en évidence les possibles problèmes de configurations de KADEPLOY sur les différents clusters**. En effet, KADEPLOY effectue des redémarrages des noeuds à plusieurs moments lors d'un déploiement (figure 2.7), il faut donc s'assurer que le délai de redémarrage n'est pas trop court, sinon le déploiement échoue, ni un délai trop long pour des raisons de performance.

3.2 Principe

Ce script va donc lancer un ensemble de déploiements rassemblés en tests afin de tester la fiabilité de KADEPLOY et des noeuds des différents clusters.

Afin de détecter les problèmes dans la section précédente, le script permet de réaliser plusieurs tests, chaque test comprenant par défaut, 1, 2, 4 et 8 déploiements concurrents lancés consécutivement sur l'ensemble des noeuds. Par exemples, si on lance le script sur 64 noeuds (numérotés de 1 à 64), on aura les déploiements suivants (identifiés par groupe de déploiements) :

1. un premier déploiement est effectué sur l'ensemble des 64 noeuds ;
2. une fois le précédent déploiement terminé, deux déploiements sont lancés en parallèle (un *thread* distinct par déploiement) comportant chacun la moitié des noeuds ;

3. quatre déploiements sont ensuite exécutés en parallèle (donc quatre *threads*) comportant chacun un quart des noeuds ;
4. enfin, huit déploiements concurrents (donc huit *threads*) sont effectués avec l'ensemble des noeuds.

3.3 Implémentation

3.3.1 Choix techniques

Ce script, KSTRESS, a été écrit en RUBY car c'est un langage de script orienté objet complet et utilisé pour la réalisation de programmes sur Grid'5000. KSTRESS repose également sur `cmdctrl` dépendant de `libtermios`. `cmdctrl`, développé par un doctorant du laboratoire, permet de récupérer les sorties standard (`stdout`) et d'erreur (`stderr`). En effet, KADEPLOY écrit sur ces deux sorties directement et n'a pas d'option pour spécifier un fichier de *log*.

3.3.2 Architecture générale

Il a été choisis de découper le script en quatre classes (diagramme de classe 3.1), une par fichier, permettant d'effectuer :

- un déploiement (classe `Deployment`) ;
- n déploiements concurrents (classe `ConcurrentDeployments`) ;
- un test qui comprend 1, 2, 4 puis 8 déploiements concurrents (classe `ConcurrentDeploymentsSet`) ;
- n tests, où n peut être spécifié par une option passée au script sinon un seul test est effectué, chaque test exécutant la classe ci-dessus (classe `StressTests`).

3.3.3 Format de sortie et exécution

Afin de pouvoir *parser* facilement les informations renvoyées par les différents tests pour pouvoir ensuite les afficher facilement sur une page web par exemple, il a fallu un format de sérialisation. YAML a donc été choisis pour sa lisibilité, simplicité et son intégration avec RUBY.

Par défaut, le script lance un seul test puis affiche une sortie détaillée des déploiements sur la sortie standard (`stdout`) et renvoie également des statistiques à propos des déploiements (temps moyen mis pour chaque groupe de déploiements concurrents, taux d'erreurs, etc.). L'annexe A.1 (page ix) comporte un exemple de sortie de la commande suivante :

```
oarsub -q deploy -l nodes=8 -p "netgdx='YES'"\
   "./bin/kstress-wrapper -e sid-x64-base-1.0 -p sda3"
```

On lance la commande `oarsub` en précisant que l'on utilise la file d'attente `deploy` (`deploy` est la file d'attente réservée aux déploiements), le cluster (NETGDX), 8 noeuds. On lance le *wrapper* de KSTRESS pour un déploiement sur la partition `sda3` avec l'environnement `sid-x64-base-1.0`. Le *wrapper* est un script *shell* permettant de lancer `kstress` en précisant l'emplacement des modules RUBY.

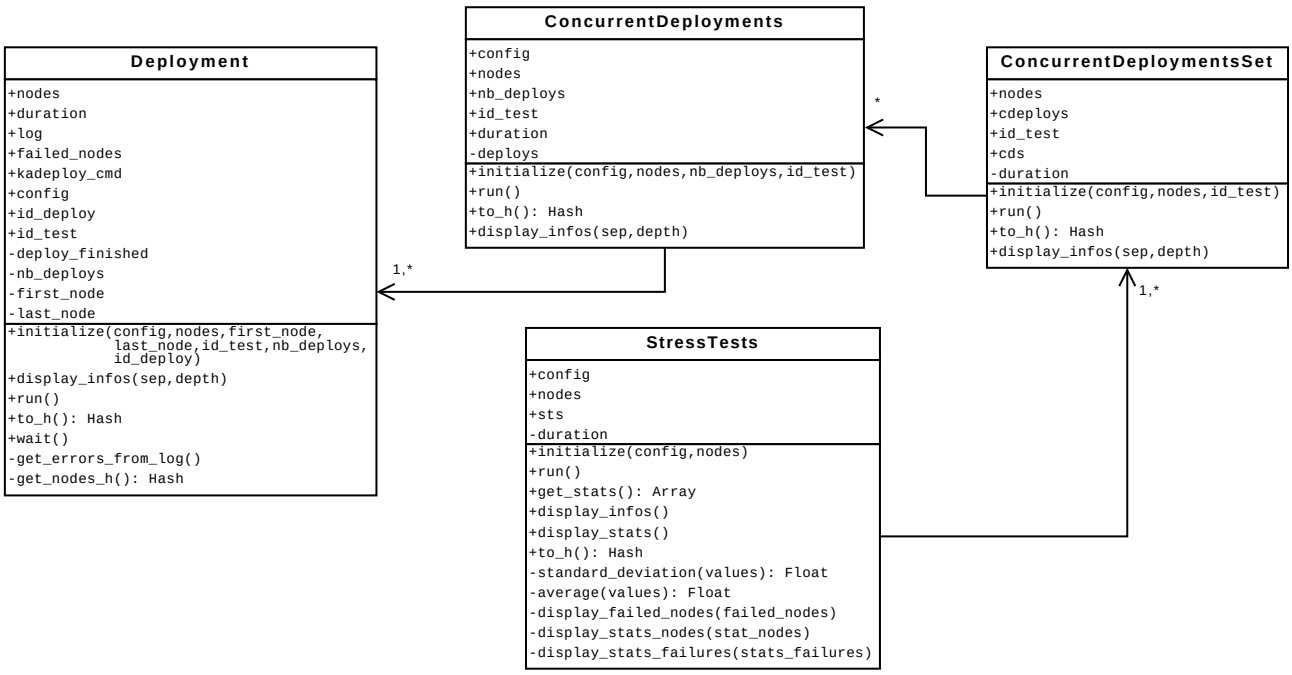

FIG. 3.1 – Diagramme de classe UML de KSTRESS

3.3.4 Exécution sur un site sans RUBY

Cet outil requiert une version récente de RUBY, ainsi que les modules `libtermios` et `cmdctrl`. Cependant, certains sites n'ont pas une version à jour de RUBY ou les modules nécessaires ne sont pas installés, il a été nécessaire de développer un *wrapper* en *shell* dont le fonctionnement est décrit ci-dessous.

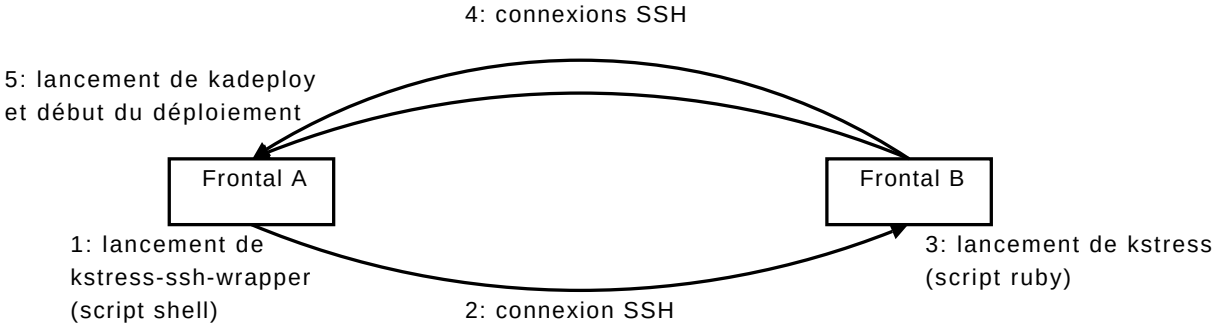

FIG. 3.2 – *Wrapper* SSH de KSTRESS si le frontal n'a pas RUBY

Le processus de déploiement via KSTRESS s'effectue ainsi (figure 3.2) :

1. on commence par lancer le *shell* sur le frontal A ;
2. celui-ci se connecte via SSH au frontal B (d'un site ou d'un cluster) sur lequel RUBY existe et/ou est à jour ;

3. le script principal, écrit en RUBY, est exécuté sur le frontal B ;
4. ce dernier va effectuer une connexion SSH par déploiement concurrent (on pourra ainsi avoir jusqu'à huit connexions simultanées) ;
5. KADEPLOY est exécuté sur le frontal B.

On peut ensuite récupérer les *logs* des déploiements sur le frontal B ainsi que le résumé sur le frontal A. Dans le cas où on a utilisé un frontal se trouvant sur un site différent, on récupérera l'ensemble en utilisant `scp` permettant d'effectuer une copie d'un fichier à distance. Dans le cas où on a utilisé un frontal se trouvant sur le même site, il est inutile d'utiliser `scp` car le répertoire utilisateur est partagé entre les différents clusters via NFS,

3.4 Problèmes rencontrés

Le premier problème rencontré a été au niveau des *threads*[1]. En effet, chaque déploiement est effectué dans un *thread* propre et il fallait donc s'assurer que l'ensemble des déploiements aient terminé leur exécution avant de pouvoir exécuter les suivants.

Le second problème rencontré est au niveau de KADEPLOY. Il a besoin d'être lancé depuis un terminal possédant des caractéristiques spéciales (`tty`) sinon la sortie standard (`stdout`) et la sortie d'erreur (`stderr`) ne peuvent pas être récupérées pour ensuite être *parser* afin de détecter les déploiements qui auraient pu échouer.

La première solution envisagée était d'utiliser la commande `script`, permettant de capturer les deux sortie de KADEPLOY, mais il a été finalement plus simple d'utiliser `cmdctrl`, décrit dans la section précédente. En effet, ce module résout les deux problèmes cités ci-dessus car il permet de lancer un programme dans un *thread* propre. Cependant, il a fallu corriger un bogue dans ce module afin de pouvoir sauvegarder la sortie de KADEPLOY une fois qu'un déploiement est terminé, donc une fois le processus KADEPLOY terminé.

3.5 Résultats

3.5.1 Découverte d'un bogue sur KADEPLOY

Le script a d'abord été lancé sur le cluster `gdx`. Cela a permis de découvrir un bogue au niveau de la génération des identifiants des déploiements. En effet, lorsque les déploiements sont lancés en même temps, leurs identifiants pouvaient être identiques, engendrant ainsi un soucis de cohérence dans les résultats car KADEPLOY affichait l'ensemble des noeuds des n déploiements concurrents pour chaque déploiement. Ce bogue a désormais été corrigé par le développeur de KADEPLOY.

3.5.2 Problèmes spécifiques aux clusters

Le script a ensuite été lancé sur la majorité des clusters en utilisant tous les noeuds disponibles. Six tests ont été lancés exécutant jusqu'à huit déploiements concurrents. Quelques problèmes ont été rencontrés et sont examinés ci-dessous (pour l'ensemble des résultats obtenus, se reporter à l'annexe A.2, page xiii).

[1] Un thread est une entité faisant partie d'un processus mais partageant le même espace mémoire avec les autres *threads* du même processus.

Problème rencontré sur le cluster PARASOL (RENNES)

La seule erreur rencontrée sur le cluster PARASOL de Rennes (figure 3.3) est `not there on check !` et semble provenir d'une mauvaise configuration du *timeout* de KADEPLOY, les machines mettant plus de temps à redémarrer que celui configuré dans KADEPLOY. Des tests sont effectués actuellement afin de vérifier que le nouveau *timeout* fonctionne correctement.

Deux noeuds rencontraient des problèmes, après vérification par l'administrateur du site, il est apparu que l'ordre de *boot* était modifié. Ainsi, le système amorcé n'est pas correct et en conséquence les déploiements échouent.

6 tests jusqu'à 8 déploiements concurrents (par nombre de déploiements concurrents)									
	Durées de déploiement (s)				Erreurs				
depl. conc.	min.	moy.	max.	écart-type	min.	moy.	**max.**	écart-type	noeuds (%)
1 (63n * 1)	542	608	695	162.94	0	0.67	**2**	1.83	1.06%
2 (31n * 2)	314	475	686	434.96	0	0.17	**1**	1.29	0.55%
4 (15n * 4)	306	442	680	560.10	0	0.08	**1**	1.35	0.53%
8 (7n * 8)	302	406	620	680.21	0	0.00	**0**	0.00	0.00%

FIG. 3.3 – KSTRESS sur RENNES (cluster PARASOL)

Problèmes rencontrés sur les cluster NETGDX et GDX (ORSAY)

Les clusters NETGDX et GDX sont administrés par le développeur de KADEPLOY et disposent donc de la version en développement.

Lors du déploiement de noeuds sur le cluster NETGDX (tableau 3.4), on remarque un taux d'erreur important. Selon l'administrateur : « *la seconde carte réseau (qui peut changer suivant les noeuds) pose également quelques problèmes, d'où des statistiques différentes* ». Ce problème n'a pas vraiment de solution actuellement.

6 tests jusqu'à 8 déploiements concurrents (par nombre de déploiements concurrents)									
	Durées de déploiement (s)				Erreurs				
depl. conc.	min.	moy.	max.	écart-type	min.	moy.	**max.**	écart-type	noeuds (%)
1 (30n * 1)	471	484	502	32.08	1	2.17	**5**	3.58	7.23%
2 (15n * 2)	352	402	478	163.91	0	0.92	**4**	4.35	6.13%
4 (7n * 4)	353	394	472	207.83	0	0.71	**2**	3.31	10.14%
8 (3n * 8)	331	363	502	275.68	0	0.15	**1**	2.45	5.00%

FIG. 3.4 – KSTRESS sur ORSAY (cluster NETGDX)

Le déploiement de noeuds sur le cluster GDX (tableau 3.5) admet un taux d'erreur très important. En effet, toujours selon l'administrateur, lors du déploiement sur un ensemble de noeuds très important (308 sur GDX, ce qui en fait donc le plus gros cluster de Grid'5000), le premier déploiement perd plus de noeuds que les suivants. On rencontrerait des comportements limites sur GDX dû au nombre important de noeuds qui seraient difficiles à repérer et donc à corriger. Une solution pour repérer les problèmes de GDX seraient de lancer plusieurs fois le script sur un même ensemble de noeuds pas trop importants.

6 tests jusqu'à 8 déploiements concurrents (par nombre de déploiements concurrents)									
	Durées de déploiement (s)				Erreurs				
depl. conc.	min.	moy.	max.	écart-type	min.	moy.	**max.**	écart-type	noeuds (%)
1 (308n * 1)	521	553	701	162.19	1	5.17	**10**	8.18	1.68%
2 (154n * 2)	408	459	701	256.66	0	3.67	**10**	13.66	2.38%
4 (77n * 4)	337	374	413	103.77	0	1.71	**6**	7.81	2.22%
8 (38n * 8)	37	333	511	373.68	0	0.94	**5**	8.17	2.47%

FIG. 3.5 – KSTRESS sur ORSAY (cluster GDX)

Chapitre 4

Tests des environnements de référence

La seconde partie du stage a consisté en l'écriture d'un script destiné à vérifier que les environnements de référence des noeuds et des frontaux disposent d'un ensemble minimal d'applications nécessaires aux expérimentations. Ce script est basé sur un script *shell* développé dans le cadre du projet, il a été réécrit en RUBY avec l'objectif d'être plus modulaire et plus facilement extensible.

Ce script n'a pas encore été utilisé en production car le comité technique de Grid'5000 doit déterminer les programmes indispensables sur les frontaux et noeuds ainsi que leurs versions.

4.1 Objectifs

Actuellement, il existe de nombreuses disparités au niveau des logiciels et des versions installés sur les frontaux et noeuds des différents sites et clusters. Cela peut poser des problèmes pour la réalisation d'expérimentations ayant besoin de certains logiciels et ne pouvant s'exécuter avec des versions trop anciennes. Par exemple, le lancement du *wrapper* SSH de KSTRESS a posé des problèmes dûs à une version de SSH trop ancienne.

Ainsi, ce script permet de **vérifier la présence de programmes et leurs versions** et **permettre l'exécution de scripts de test** à partir d'un ou plusieurs fichiers de spécification écrits en YAML.

4.2 Implémentation

4.2.1 Choix techniques

Ce script, REFENV, a été écrit en RUBY pour les même raisons que KSTRESS (section 3.3, 16) mais ne requiert aucun module supplémentaire.

4.2.2 Architecture générale

Il a été choisis de découper le script en quatre classes principales (diagramme de classe 4.1), une par fichier, permettant d'effectuer :

– un test général, dont hériterons les deux classes ci-dessous (classe `EnvironmentTest`) ;

– un test sur une commande (`CommandTest`) ;
– un test sur l'exécution d'un script qui doit renvoyer 0 comme code de retour, sinon cela signifie que le programme a rencontré une erreur (classe `ScriptTest`) ;
– un test sur l'ensemble des scripts d'un répertoire (classe `DirectoryTest`).

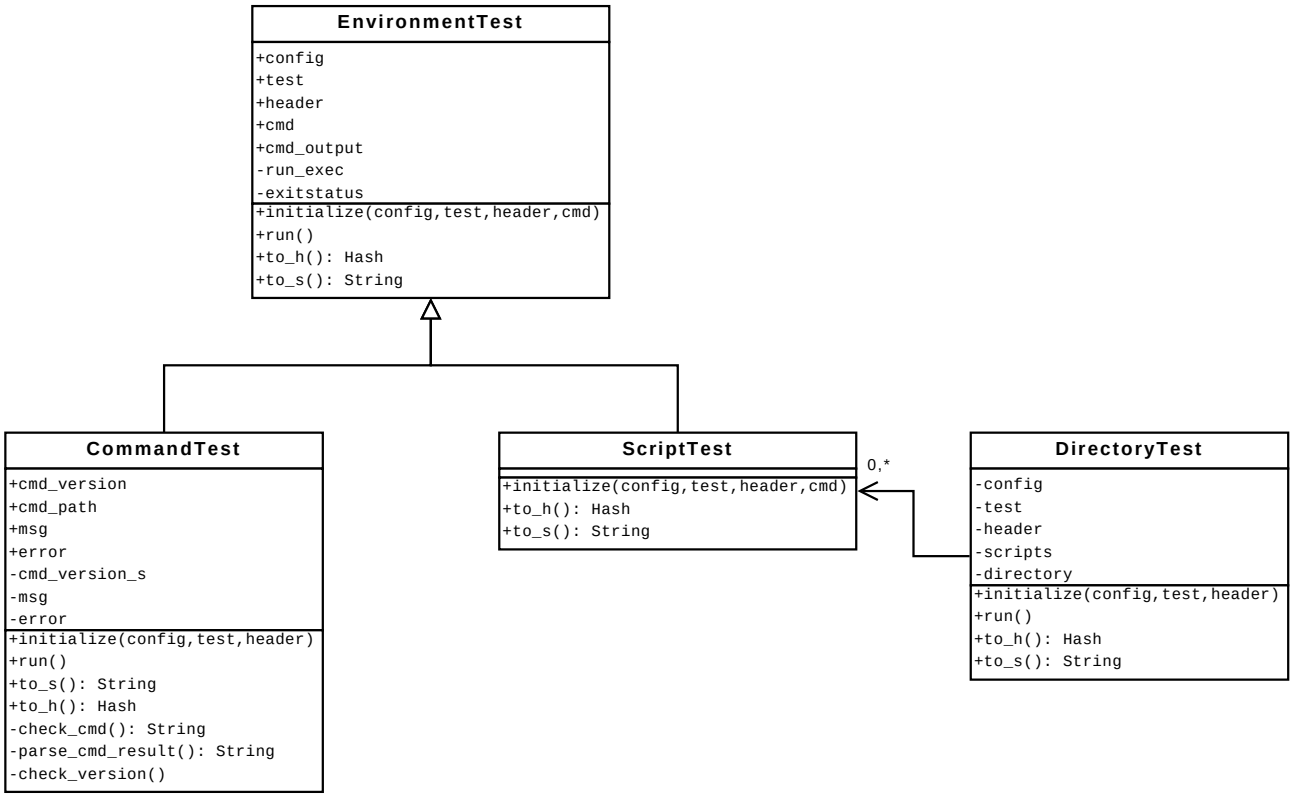

FIG. 4.1 – Diagramme de classe UML de REFENV

En plus de ces classes, il existe une classe (`CompareVersions`) permettant de comparer les versions d'un programme, il supporte simplement les deux opérateurs pris en compte dans le fichier de spécification (>= et <).

Ce script est censé être lancé manuellement sur le frontal ou sur un noeud une fois que ce dernier a été réservé en mode interactif.

4.2.3 Format de sortie

Par défaut, REFENV génère un rapport succinct affiché sur la sortie standard (`stdout`) et un rapport plus détaillé dans un fichier `report.log`. Ces deux rapports sont simplement du texte formaté comme nous l'illustrerons dans la section suivante.

Il est également possible de générer le rapport en YAML afin de pouvoir *parser* facilement la sortie. À l'instar de KSTRESS, cette sortie en YAML permettrait de générer une page web indiquant l'état des différents environnements de référence des différents sites.

4.3 Exemple d'utilisation

La figure 4.2 est un exemple de fichier de spécification permettant de valider les points suivants :

- la version de RUBY installée sur les frontaux est supérieure ou égale à 1.8.4 et strictement inférieure à 1.9 ;
- la version de PYTHON installée sur les frontaux et les noeuds (`all`) est au moins 2.4 ;
- les scripts se trouvant dans le répertoire `examples/scripts` renvoient 0 comme code de retour.

```
title: Fichier de spécification test1
description: description du fichier de spécification
version: 0.1
target: all
---
target: frontend
cmd: ruby
cmd-version: ruby -v
version-min: 1.8.4
version-max: 1.9
version-re: !ruby/regexp "/^ruby (.*) \(.*/"
---
target: all
cmd: python
cmd-version: python -V
version-min: 2.4.6
version-re: !ruby/regexp "/^Python (.*)/"
---
target: all
directory: examples/scripts
```

FIG. 4.2 – Fichier de spécification REFENV en YAML

Il a été décidé de générer un rapport succinct (figure 4.3) sur la sortie standard de la commande et un autre plus détaillé dans un fichier `.log` (figure 4.4). Ces deux rapports sont issus de la commande suivante :

```
./bin/refenv-wrapper --frontend=y examples/example.yaml
```

On utilise un *wrapper* en *shell* afin de pouvoir configurer l'emplacement des modules du script.

```
OK        ruby                          found /usr/bin/ruby (1.8.4 <= 1.8.6 < 1.9)
FAILED    python                        Installed version is 2.4.4, whereas it requires at least 2.4.6
OK        examples/scripts/example1.sh  status code 0
FAILED    examples/scripts/example2.sh  status code 1
```

FIG. 4.3 – Résumé du rapport de REFENV

```
#################### Test1 v0.1 (examples/example.yaml) ####################
==================================== ruby ====================================
OK: found /usr/bin/ruby (1.8.4 <= 1.8.6 < 1.9)
OUTPUT: ruby 1.8.6 (2007-03-13 patchlevel 0) [i486-linux]
================================== python ==================================
FAILED: Installed version is 2.4.4, whereas it requires at least 2.4.6
OUTPUT: Python 2.4.4
======================== directory: examples/scripts =========================
----------------------- examples/scripts/example1.sh -----------------------
OK: status code 0
OUTPUT: foo
----------------------- examples/scripts/example2.sh -----------------------
FAILED: status code 1
OUTPUT: bar
```

FIG. 4.4 – Rapport complet de REFENV

Conclusion

Les grilles de calcul représentent actuellement un avenir prometteur permettant de disposer d'une puissance de calcul importante. Cette dernière est particulièrement nécessaire dans le domaine de la recherche scientifique. Ces grilles de calcul sont hétérogènes et constituées d'un nombre important de machines, constituant ainsi un environnement complexe.

Cette complexité devient encore plus importante dans les grilles de calcul expérimentales. En effet, les outils utilisés sont eux aussi expérimentaux mais surtout il n'est pas toujours évident de détecter et corriger les problèmes de ces différents outils lorsqu'on teste ces outils à grande échelle. Grid'5000 est donc une grille de calcul expérimentale où sont développés des outils déjà utilisés en production. Un autre problème rencontré dans ce type de grille de calcul est le manque de personnels s'en occupant. C'est pourquoi il est nécessaire de tester ces différents outils automatiquement. Tout d'abord afin d'éviter des déploiements d'outils en versions instables.

Le premier script développé, KSTRESS, a permis de détecter des problèmes sur certains clusters qui sont en train d'être examinés afin de réduire le taux d'erreur mais aussi améliorer les temps de déploiements. De plus, ce script pourra être utilisé afin de tester les prochaines versions majeures de KADEPLOY.

Le second script, REFENV, est plus modulaire et extensible que la version écrite en *shell*. Lorsqu'une liste des logiciels des environnements de référence aura été établie, ce script permettra de tester la présence et la version des différents logiciels installés.

Ce stage m'a apporté énormément au niveau programmation parallèle et m'a permis d'apprendre également des fonctionnalités avancées de SSH. De plus, j'ai pu découvrir un nouveau langage de programmation qu'est le RUBY. Enfin ce stage m'a permis de découvrir le monde du travail mais aussi d'acquérir une première expérience professionnelle. Il m'a surtout permis de découvrir l'environnement de la recherche ainsi que les environnements distribués tant au niveau réseau qu'au niveau système. J'ai donc énormément enrichi mes connaissances au niveau programmation et système grâce à cet environnement et surtout grâce aux personnes que j'ai rencontré.

Glossaire

Cluster ou grappe – regroupe un ensemble d'ordinateurs homogène se situant au même endroit (localisé).

Frontal – machines des différents clusters sur lesquelles on peut se connecter pour réserver les noeuds.

Intergiciel ou middleware – désigne les logiciels servant d'intermédiaire entre d'autres logiciels. On utilise généralement du *middleware* comme intermédiaire de communication entre des applications complexes, distribuées sur un réseau informatique.

Kadeploy – outil permettant d'utiliser un environnement personnalisé, à la différence de l'environnement de référence (c.-à-d. l'environnement installé par défaut sur les noeuds).

LDAP (Lightweight Directory Access Protocol) – protocole permettant l'interrogation et la modification des services d'annuaire (généralement gestion de comptes utilisateur).

NFS (Network File System) – système de fichiers en réseau permettant, au niveau de Grid'5000, de rendre disponible les répertoires utilisateurs sur les différents clusters d'un même site.

OAR – gestionnaire de ressources utilisé sur Grid'5000 mais aussi sur des grilles de calcul en production. Cet outil permet de réserver des machines afin d'effectuer des expérimentations.

Ordonnanceur de type batch – gestion des tâches organisée en file d'attente, chaque tâche doit se terminer avant qu'une autre débute.

Parser – consiste à analyser syntaxiquement un fichier afin de récupérer des informations. Il est logiquement plus simple d'utiliser un format existant auquel correspondra généralement un module s'occupant de cette tâche.

Partition – une partition permet de découper logiquement un disque dur.

Processeur multi-coeur – permet de combiner un ou plusieurs processeurs en une seule puce permettant d'augmenter la puissance de calcul sans augmenter la fréquence du processeur et permet de réduire la dissipation thermique.

Shell – interface utilisateur pouvant être aussi bien graphique que texte. Dans ce rapport, c'est une interface en mode texte permettant de d'entrer des commandes.

SSH (Secure SHell) – programme informatique et protocole de communication sécurisé, particulièrement utilisés pour l'administration de machines à distance de manière sécurisée.

Sérialisation – consiste à sauvegarder un objet sous forme de chaîne qui permettra de le restaurer lors de la prochaine exécution du programme.

Thread – comparable à un processus si ce n'est qu'un thread appartient à un processus. Les différents threads d'un processus partagent cet espace mémoire.

Wrapper – code permettant de lancer un autre script avec différents paramètres ou variables prédéfinies (par exemple pour configurer le chemin des modules RUBY).

YAML – langage de sérialisation de données.

Table des figures

Bibliographie

[1] GRID'5000
http://grid5000.fr

[2] WIKIPEDIA
Gestionnaire de tâches
http://fr.wikipedia.org/wiki/Gestionnaire_de_tâches

[3] *Ka Clustering Tools*
http://ka-tools.sourceforge.net/

[4] YIANNIS GEORGIOU, JULIEN LEDUC, BRICE VIDEAU, JOHANN PEYRARD et OLIVIER RICHARD
A Tool for Environment Deployment in Clusters and light Grids
http://www2.lifl.fr/MAP/negst/firstWorkshop/slides/Kadeploy2_SMTPS.pdf, 2006

[5] OAR *Documentation*
http://oar.imag.fr/docs/manual.html, 2007

[6] N. CAPIT, G. DA COSTA, G. HUARD, C. MARTIN, G. MOUNIÉ, P. NEYRON, et O. RICHARD
Expériences autour d'une nouvelle approche de conception d'un gestionnaire de travaux pour grappe
http://oar.imag.fr/papers/oar_cfse03.pdf, Octobre 2003

[7] YAML4R PROJECT *Yaml Cookbook* http://yaml4r.sourceforge.net/cookbook/,

Planning

Semaine 1	Lecture de documentations sur Grid'5000 et RUBY mais aussi familiarisation avec Grid'5000
Semaine 2	Écriture de KSTRESS
Semaine 3	Écriture de KSTRESS
Semaine 4	Corrections sur KSTRESS et écriture de REFENV
Semaine 5	Ajout de fonctionnalités sur REFENV et lancement de KSTRESS sur tous les clusters de Grid'5000
Semaine 6	Lancement de KSTRESS sur les sites posant problème précédemment et analyse des résultats
Semaine 7	Rédaction du rapport
Semaine 8	Préparation de la présentation pour le comité technique
Semaine 9	Présentation du travail effectué lors du comité technique du projet Grid'5000 à Sophia Antipolis
Semaine 10	Tests sur MPI et préparation de la soutenance

Annexe A

Annexes

A.1 Rapport complet de KSTRESS

Exemple de rapport de l'outil KSTRESS lancé sur 20 noeuds pour un test exécutant jusqu'à 8 déploiements concurrents :

```
Ran 1 test(s) (1991s)
  Test #1 (1991s)
    Ran 1 simultaneous deployment(s) (259s)
      Deployment #1 (259s)
        Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-1d-1to20
        Node(s) involved:
          > gdx0304.orsay.grid5000.fr
          > gdx0294.orsay.grid5000.fr
          > gdx0275.orsay.grid5000.fr
          > gdx0184.orsay.grid5000.fr
          > gdx0041.orsay.grid5000.fr
          > gdx0222.orsay.grid5000.fr
          > gdx0032.orsay.grid5000.fr  (Failed: Preinstall failed on node)
          > gdx0070.orsay.grid5000.fr
          > gdx0242.orsay.grid5000.fr
          > gdx0114.orsay.grid5000.fr
          > gdx0077.orsay.grid5000.fr
          > gdx0060.orsay.grid5000.fr
          > gdx0277.orsay.grid5000.fr
          > gdx0233.orsay.grid5000.fr
          > gdx0227.orsay.grid5000.fr
          > gdx0280.orsay.grid5000.fr
          > gdx0247.orsay.grid5000.fr
          > gdx0160.orsay.grid5000.fr
          > gdx0259.orsay.grid5000.fr
          > gdx0183.orsay.grid5000.fr
    Ran 2 simultaneous deployment(s) (259s)
      Deployment #1 (259s)
        Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-2d-1to10
        Node(s) involved:
          > gdx0184.orsay.grid5000.fr  (Failed: Preinstall failed on node)
          > gdx0041.orsay.grid5000.fr
          > gdx0222.orsay.grid5000.fr
          > gdx0032.orsay.grid5000.fr  (Failed: Preinstall failed on node)
          > gdx0070.orsay.grid5000.fr
          > gdx0114.orsay.grid5000.fr
```

```
      > gdx0077.orsay.grid5000.fr
      > gdx0060.orsay.grid5000.fr
      > gdx0160.orsay.grid5000.fr
      > gdx0183.orsay.grid5000.fr
  Deployment #2 (247s)
    Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-2d-11to20
    Node(s) involved:
      > gdx0304.orsay.grid5000.fr
      > gdx0294.orsay.grid5000.fr
      > gdx0275.orsay.grid5000.fr
      > gdx0242.orsay.grid5000.fr
      > gdx0277.orsay.grid5000.fr
      > gdx0233.orsay.grid5000.fr
      > gdx0227.orsay.grid5000.fr
      > gdx0280.orsay.grid5000.fr
      > gdx0247.orsay.grid5000.fr
      > gdx0259.orsay.grid5000.fr
Ran 4 simultaneous deployment(s) (753s)
  Deployment #1 (720s)
    Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-4d-1to5
    Node(s) involved:
      > gdx0041.orsay.grid5000.fr
      > gdx0032.orsay.grid5000.fr  (Failed: Preinstall failed on node)
      > gdx0070.orsay.grid5000.fr
      > gdx0077.orsay.grid5000.fr
      > gdx0060.orsay.grid5000.fr
  Deployment #2 (753s)
    Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-4d-6to10
    Node(s) involved:
      > gdx0184.orsay.grid5000.fr  (Failed: Preinstall failed on node)
      > gdx0222.orsay.grid5000.fr
      > gdx0114.orsay.grid5000.fr
      > gdx0160.orsay.grid5000.fr
      > gdx0183.orsay.grid5000.fr
  Deployment #3 (498s)
    Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-4d-11to15
    Node(s) involved:
      > gdx0242.orsay.grid5000.fr
      > gdx0233.orsay.grid5000.fr
      > gdx0227.orsay.grid5000.fr
      > gdx0247.orsay.grid5000.fr
      > gdx0259.orsay.grid5000.fr
  Deployment #4 (737s)
    Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-4d-16to20
    Node(s) involved:
      > gdx0304.orsay.grid5000.fr
      > gdx0294.orsay.grid5000.fr  (Failed: Preinstall failed on node)
      > gdx0275.orsay.grid5000.fr
      > gdx0277.orsay.grid5000.fr
      > gdx0280.orsay.grid5000.fr
Ran 8 simultaneous deployment(s) (719s)
  Deployment #1 (719s)
    Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-8d-1to2
    Node(s) involved:
      > gdx0041.orsay.grid5000.fr
      > gdx0032.orsay.grid5000.fr
  Deployment #2 (467s)
    Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-8d-3to4
    Node(s) involved:
      > gdx0070.orsay.grid5000.fr
      > gdx0060.orsay.grid5000.fr
  Deployment #3 (467s)
    Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-8d-5to6
    Node(s) involved:
      > gdx0114.orsay.grid5000.fr
```

```
          > gdx0077.orsay.grid5000.fr
      Deployment #4 (705s)
        Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-8d-7to8
        Node(s) involved:
          > gdx0160.orsay.grid5000.fr
          > gdx0183.orsay.grid5000.fr
      Deployment #5 (466s)
        Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-8d-9to10
        Node(s) involved:
          > gdx0184.orsay.grid5000.fr
          > gdx0222.orsay.grid5000.fr
      Deployment #6 (466s)
        Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-8d-11to12
        Node(s) involved:
          > gdx0233.orsay.grid5000.fr
          > gdx0227.orsay.grid5000.fr
      Deployment #7 (466s)
        Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-8d-13to14
        Node(s) involved:
          > gdx0242.orsay.grid5000.fr
          > gdx0247.orsay.grid5000.fr
      Deployment #8 (466s)
        Logs basename: kadeploy-devgdx002.orsay.grid5000.fr-20070515140257-t1-8d-15to16
        Node(s) involved:
          > gdx0275.orsay.grid5000.fr
          > gdx0259.orsay.grid5000.fr
********************
Summary (ran 1):
* The following nodes failed
+ gdx0032.orsay.grid5000.fr
  - Preinstall failed on node (test #1, 1 deployment(s), id: #1)
  - Preinstall failed on node (test #1, 2 deployment(s), id: #1)
  - Preinstall failed on node (test #1, 4 deployment(s), id: #1)

+ gdx0184.orsay.grid5000.fr
  - Preinstall failed on node (test #1, 2 deployment(s), id: #1)
  - Preinstall failed on node (test #1, 4 deployment(s), id: #2)

+ gdx0294.orsay.grid5000.fr
  - Preinstall failed on node (test #1, 4 deployment(s), id: #4)

* Deployments duration per number of concurrents deployments (seconds)
conc. depl.    min       avg       max       stddev
1 (20n * 1)    259       259       259       0.00
2 (10n * 2)    247       253       259       8.49
4 (5n * 4)     498       677       753       208.00
8 (2n * 8)     466       527       719       301.04

* Failures per number of concurrents deployments
conc. depl.    min       avg       max       stddev    nodes
1 (20n * 1)    1         1.00      1         0.00      5.00%
2 (10n * 2)    0         1.00      2         1.41      10.00%
4 (5n * 4)     0         0.75      1         0.87      15.00%
8 (2n * 8)     0         0.00      0         0.00      0.00%
```

A.2 Rapports de KSTRESS pour tous les clusters

Le tableau suivant représente les résultats des tests effectués avec KSTRESS sur l'ensemble des clusters :

6 tests jusqu'à 8 déploiements concurrents (par nombre de déploiements concurrents)										
		Durées de déploiement (s)				Erreurs				
cluster	depl. conc.	min.	moy.	max.	écart-type	min.	moy.	max.	écart-type	noeuds (%)
Capricorne (Lyon)	1 (29n * 1)	651	658	665	11.78	0	0.00	0	0.00	0.00%
	2 (14n * 2)	266	529	654	533.21	0	0.00	0	0.00	0.00%
	4 (7n * 4)	232	456	949	1011.85	0	0.00	0	0.00	0.00%
	8 (3n * 8)	226	273	641	734.21	0	0.00	0	0.00	0.00%
Sagittaire (Lyon)	1 (67n * 1)	293	395	670	341.81	0	0.00	0	0.00	0.00%
	2 (33n * 2)	283	340	507	294.26	0	0.00	0	0.00	0.00%
	4 (16n * 4)	247	311	484	349.28	0	0.00	0	0.00	0.00%
	8 (8n * 8)	240	285	504	458.90	0	0.00	0	0.00	0.00%
Netgdx (Orsay)	1 (30n * 1)	471	484	502	32.08	1	2.17	5	3.58	7.23%
	2 (15n * 2)	352	402	478	163.91	0	0.92	4	4.35	6.13%
	4 (7n * 4)	353	394	472	207.83	0	0.71	2	3.31	10.14%
	8 (3n * 8)	331	363	502	275.68	0	0.15	1	2.45	5.00%
	Erreurs rencontrées : *Preinstall failed on node*, *not there on last check*.									
Gdx (Orsay)	1 (308n * 1)	521	553	701	162.19	1	5.17	10	8.18	1.68%
	2 (154n * 2)	408	459	701	256.66	0	3.67	10	13.66	2.38%
	4 (77n * 4)	337	374	413	103.77	0	1.71	6	7.81	2.22%
	8 (38n * 8)	37	333	511	373.68	0	0.94	5	8.17	2.47%
	Erreurs rencontrées : *Preinstall failed on node*, *mount of /dev/sda3 failed on node*, *not there on first check*, *not there on last check*, *simple remote execution failed on node*.									
Grelon (Nancy)	1 (120n * 1)	365	440	546	180.86	0	0.00	0	0.00	0.00%
	2 (60n * 2)	349	390	519	193.74	0	0.00	0	0.00	0.00%
	4 (30n * 4)	299	353	498	235.52	0	0.00	0	0.00	0.00%
	8 (15n * 8)	276	325	539	346.45	0	0.00	0	0.00	0.00%
Grillon (Nancy)	1 (44n * 1)	312	333	341	24.14	0	0.00	0	0.00	0.00%
	2 (22n * 2)	266	308	343	97.93	0	0.00	0	0.00	0.00%
	4 (11n * 4)	245	289	351	130.73	0	0.00	0	0.00	0.00%
	8 (5n * 8)	239	274	343	201.72	0	0.00	0	0.00	0.00%

		Durées de déploiement (s)				Erreurs				
cluster	depl. conc.	min.	moy.	max.	écart-type	min.	moy.	max.	écart-type	noeuds (%)
Infiniband (Bordeaux)	1 (48n * 1)	710	862	918	187.82	0	0.00	0	0.00	0.00%
	2 (24n * 2)	708	823	903	225.19	0	0.00	0	0.00	0.00%
	4 (12n * 4)	379	774	915	667.35	0	0.00	0	0.00	0.00%
	8 (6n * 8)	321	702	910	1107.08	0	0.00	0	0.00	0.00%
Paraquad (Rennes)	1 (64n * 1)	319	338	351	30.33	0	0.00	0	0.00	0.00%
	2 (32n * 2)	302	328	363	72.24	0	0.00	0	0.00	0.00%
	4 (16n * 4)	301	329	367	80.97	0	0.00	0	0.00	0.00%
	8 (8n * 8)	292	344	402	236.21	0	0.00	0	0.00	0.00%
Parasol (Rennes)	1 (63n * 1)	542	608	695	162.94	0	0.67	2	1.83	1.06%
	2 (31n * 2)	314	475	686	434.96	0	0.17	1	1.29	0.55%
	4 (15n * 4)	306	442	680	560.10	0	0.08	1	1.35	0.53%
	8 (7n * 8)	302	406	620	680.21	0	0.00	0	0.00	0.00%
	Erreurs rencontrées : *not there on last check*, *not there on check*.									
Sol (Sophia)	1 (50n * 1)	178	190	201	21.67	0	0.00	0	0.00	0.00%
	2 (25n * 2)	162	206	369	249.82	0	0.00	0	0.00	0.00%
	4 (12n * 4)	156	192	362	252.67	0	0.00	0	0.00	0.00%
	8 (6n * 8)	137	196	345	281.30	0	0.00	0	0.00	0.00%
Sol (Sophia)	1 (71n * 1)	534	634	699	174.92	0	0.00	0	0.00	0.00%
	2 (35n * 2)	511	617	691	243.15	0	0.00	0	0.00	0.00%
	4 (17n * 4)	244	501	698	595.41	0	0.00	0	0.00	0.00%
	8 (8n * 8)	235	399	724	805.87	0	0.00	0	0.00	0.00%

Résumé

Les grilles de calcul sont des environnements logiciels et matériels complexes à cause de leur hétérogénéité et de leur taille. Leur bon fonctionnement est difficile à assurer. Ce stage a consisté en l'écriture de deux outils permettant de vérifier le bon fonctionnement de certains aspects de Grid'5000. Le premier outil, KSTRESS, permet de détecter les problèmes au niveau de l'outil responsable de l'installation d'environnements de travail personnalisés. Le second outil devrait permettre prochainement de vérifier les logiciels installés ainsi que leurs versions sur les environnements de travail de Grid'5000.

Mots-clés : grille de calcul, gestion de la qualité, cluster, environnements distribués

Abstract

As computing grids involve sharing heterogeneous resources, software and hardware environments are quite complex. Thus, it is complicated to ensure that everything works properly. This internship was about writing two tools for checking if the tools within the computing grid works correctly. The first tool, KSTRESS, aims to find bugs in the software which sets up customized environments. The second tool will check for missing or out-to-date softwares in the Grid'5000 work environments.

Keywords : computing grid, quality assurance, cluster, distributed environments