Réalisation d'expériences avec Grid'5000

Simon Delamare
LIP – Laboratoire de l'informatique du parallélisme
Université de Lyon (CNRS, ENS de Lyon, Inria, UCBL)

Pascal Morillon
IRISA – Institut de Recherche en Informatique et Systèmes Aléatoires
Université de Rennes

Lucas Nussbaum
LORIA – Laboratoire Lorrain de Recherche en Informatique et ses Applications
Université de Lorraine

Résumé

Grid'5000 est une infrastructure pour la recherche et l'expérimentation des systèmes informatiques. Son principal objectif est de proposer une plate-forme pour réaliser des expériences, pouvant être de natures très diverses : par exemple, valider le bon comportement d'un logiciel, d'un matériel ou d'une configuration.

Son utilisation est particulièrement adaptée au déploiement de systèmes large échelle et complexes (une infrastructure OpenStack complète, une topologie réseau de grande envergure, etc.), souvent difficilement réalisable avec les ressources et outils disponibles en interne.

Pour cela, Grid'5000 met à disposition des ressources matérielles abondantes et variées, entièrement réservables et reconfigurables par ses utilisateurs, ainsi qu'une suite logicielle pour aider à la réalisation des expériences.

Après avoir introduit le fonctionnement et les principaux services rendus par la plate-forme, notre présentation montrera de manière pratique comment réaliser une expérience à l'aide de Grid'5000.

Pour cela, nous nous appuierons sur un scénario d'exemple, qui consiste à évaluer les ressources matérielles nécessaires au déploiement d'une application Web devant satisfaire un grand nombre de requêtes clientes.

Les différentes étapes nécessaires à la mise en place de ce scénario seront décrites : réservation des ressources matérielles, déploiement des systèmes d'exploitation, mise en place de l'application Web au sein d'instances Docker répliquées, génération des requêtes clientes et utilisation de l'API Grid'5000 pour collecter les mesures de performance. Nous expliquerons également comment cette expérience peut être scriptée dans un langage de haut niveau tel que Python.

Mots clefs

Expérimentation, Plate-forme, Infrastructure, Bare metal, Docker, NGINX

1 Introduction

Une des activités incontournables des métiers liés aux systèmes et réseaux informatiques est l'expérimentation : elle est nécessaire pour valider une solution, par exemple vérifier le bon comportement d'un logiciel,

d'un matériel ou d'une configuration. Lorsque l'objet étudié est de grande taille et avec des contraintes de performance, par exemple lorsqu'il met en jeu de nombreuses machines ou une architecture réseau complexe, les solutions basées sur la virtualisation (Vagrant) ou les conteneurs (Docker) ne sont plus suffisantes, à cause des biais et des limitations qu'elles induisent.

Grid'5000 tente de répondre à cette problématique en proposant une plate-forme pour la recherche et l'expérimentation en informatique distribuée (incluant notamment les domaines des systèmes distribués, du Cloud, des réseaux, du calcul haute performance, du Big Data, etc.) [1].

En mettant à disposition des ressources matérielles abondantes et variées, entièrement réservables et reconfigurables par les utilisateurs, ainsi qu'une suite logicielle complète pour la réalisation d'expériences, Grid'5000 est devenue une plate-forme de référence et a servi de support à de nombreux travaux de recherche et de développement[1].

Notre proposition d'article et de présentation vise à exposer comment les ingénieurs systèmes et réseaux de la communauté enseignement et recherche pourraient utiliser Grid'5000 pour mesurer des performances, tester et évaluer différentes solutions logicielles, étudier un déploiement de préproduction, etc.

La suite de ce document abordera les différents points que nous proposons de présenter : le fonctionnement de Grid'5000, les principaux services rendus par la plate-forme et enfin le détail de la mise en place d'une expérience à l'aide d'un scénario d'exemple.

2 Présentation de Grid'5000

2.1 Fonctionnement

Grid'5000 est organisé en un Groupement d'Intérêt Scientifique auquel participent les principaux acteurs de l'informatique dans l'enseignement supérieur et la recherche : l'Inria, le CNRS, la Conférence des Présidents d'Université, etc.

Depuis son ouverture en 2005, Grid'5000 accueille environ 500 utilisateurs chaque année. En 2016, 40 millions de coeurs.heures y ont été utilisés. La communauté des utilisateurs est structurée autour du site Web, qui propose nouvelles et documentation, de la mailing list et des formations et écoles organisées régulièrement.

2.2 Ressources matérielles et principaux outils

La plate-forme Grid'5000 est répartie en 8 sites (7 en France et 1 au Luxembourg). Ces sites sont reliés entre eux par un réseau 10 Gbit/s fourni par Renater (figure 1), et dédié à la plate-forme (isolé du réseau de production de Renater, pour permettre de garantir les performances sur le réseau de Grid'5000).

Grid'5000 met à disposition de ses utilisateurs plus de 800 machines ou *nœuds*, qui sont répartis sur les différents sites et regroupés au sein de clusters de configuration homogène.

Le matériel disponible dans ces nœuds est varié : on y trouve différentes générations de processeurs Intel (et AMD), de technologies réseaux (Ethernet 10 Gbit/s, Infiniband, Omni-Path), de dispositifs de stockage (disques SSD, disques de grand volume) et de cartes accélératrices (GPU, Xeon Phi).

Grâce au gestionnaire de ressources OAR, les utilisateurs ont la possibilité de réserver un ou plusieurs nœuds durant une durée limitée. OAR permet ensuite aux utilisateurs des machines de s'y connecter interactivement, de leur soumettre un programme à exécuter, ou simplement de les réserver avant de les reconfigurer.

1. Le site Web de Grid'5000 est https://www.grid5000.fr

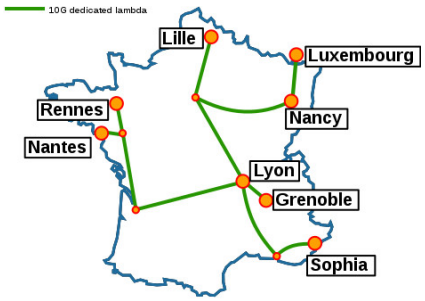

Figure 1 - L'infrastructure Grid'5000 répartie en différents sites

La reconfiguration des machines Grid'5000 est possible à plusieurs niveaux. L'outil Kadeploy est disponible pour redéployer le système d'exploitation de son choix sur les machines réservées, au niveau *bare metal*. De manière plus facile, il est possible de devenir *root* en utilisant la commande `sudo-g5k` afin d'effectuer les opérations privilégiées classiques (installation de logiciels, modification des configurations, etc). Pour restaurer l'environnement d'origine, le nœud est ensuite réinstallé par Kadeploy à la fin de la réservation, de manière transparente. L'outil Kavlan permet de basculer les nœuds dans un VLAN isolé, dédié à l'expérience de l'utilisateur. Ces VLANs peuvent aussi être propagés entre les sites, pour isoler une *tranche* multi-sites de la plate-forme pour la durée d'une expérience. Enfin, certains paramètres CPU (P-states, C-states) peuvent aussi être modifiés.

Il est important de noter que Grid'5000 suit un modèle *Hardware as a Service* : les utilisateurs ont accès directement aux ressources matérielles. La reconfiguration s'effectue donc au niveau *bare metal*, ce qui permet par exemple aux utilisateurs de comparer différentes solutions de virtualisation.

Grid'5000 propose également différents outils pour faciliter la réalisation et l'interprétation des expériences. Il est par exemple possible de collecter des informations sur les performances des machines réservées : utilisation CPU, mémoire et réseau, mais également consommation électrique mesurée à la prise.

De plus, Grid'5000 offre une API de type REST pour accéder à ses différentes fonctionnalités (réservation, reconfiguration, monitoring, etc.). Afin de permettre le *scripting* des expériences dans des langages de programmation de haut niveau (Python, Ruby), différentes bibliothèques qui utilisent cette API sont disponibles.

Enfin, différents scripts, dont certains ont été proposés par la communauté des utilisateurs, sont disponibles pour mettre en place automatiquement des infrastructures complexes sur les ressources Grid'5000, par exemple basées sur OpenStack, Ceph, ou Hadoop.

3 Exemple de scénario d'expérience

Voici un exemple d'expérience illustrant certaines possibilités de la plate-forme. Considérons une application Web. Nous cherchons à savoir le nombre de serveurs nécessaires à l'hébergement de celle-ci pour que le service soit correctement rendu, en fonction d'un certain volume de requêtes clientes.

L'application considérée sera *Jupyter NBViewer*[2] qui permet d'interpréter dynamiquement un *notebook* Jupyter et d'afficher son rendu au travers d'un serveur Web. Plusieurs instances de cette application seront déployées par Docker sur un ensemble de serveurs physiques (*backend nodes*). Afin de répartir les requêtes des clients entre les différentes instances, un équilibrage de charge sera réalisé par le logiciel *NGINX*, qui sera déployé sur une machine physique dédiée (*load balancer*). Enfin, une dernière machine physique

2. https://github.com/jupyter/nbviewer

(*request generator*) sera utilisée pour générer les requêtes des clients qui consisteront à demander à l'application NBViewer d'interpréter un notebook de résolution d'équation de la chaleur par la méthode de Crank Nicholson [3] et de la retourner sous forme de page HTML. L'architecture de cette expérience est illustrée dans la figure 2.

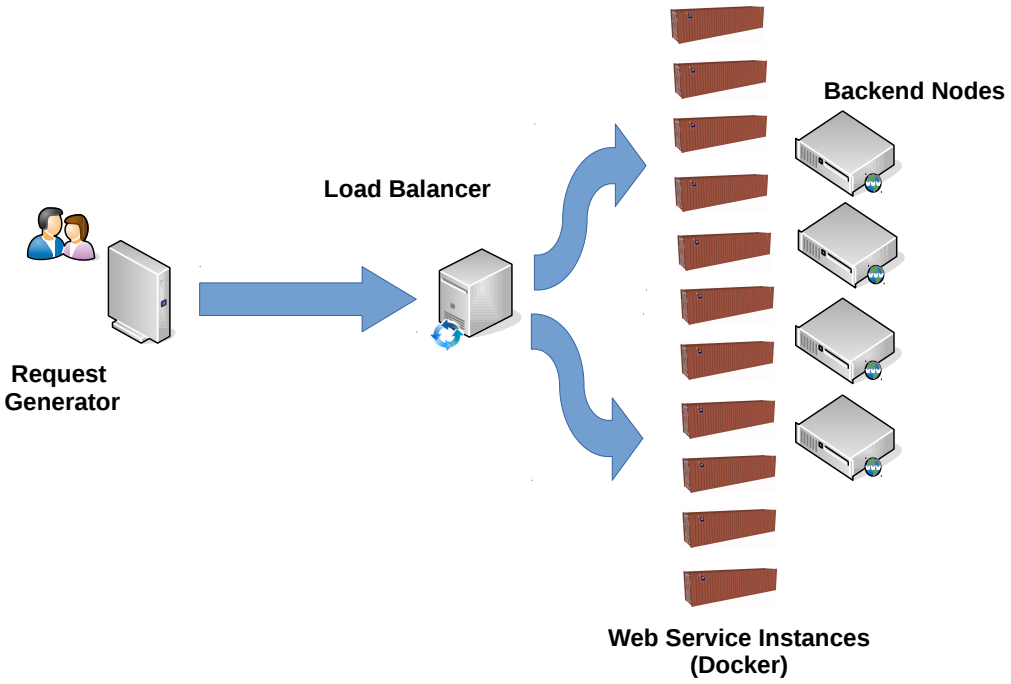

Figure 2 - Architecture du scénario d'expérience : test de charge d'un service Web répliqué

3.1 Mise en place de l'expérience dans Grid'5000

La mise en place d'une telle expérience dans Grid'5000 se réalise en plusieurs étapes.

3.1.1 Choix et réservation des ressources matérielles

En fonction des caractéristiques matérielles du serveur qu'il souhaite étudier, l'utilisateur va choisir quelques nœuds d'un cluster parmi ceux disponibles dans Grid'5000.

L'ensemble des clusters disponibles et leurs caractéristiques sont décrits dans la page *Hardware* du site Grid'5000, ainsi que dans l'API Grid'5000, accessible en HTTP au format JSON. Un extrait de la sortie de cette API est proposé dans l'extrait 1.

L'utilisateur doit ensuite obtenir un accès aux ressources choisies pour pouvoir les utiliser. OAR [4] est le système de gestion de ressources utilisé dans Grid'5000. Il permet l'ordonnancement des *jobs* utilisateurs

3. Voir http://georg.io/2013/12/Crank_Nicolson et http://nbviewer.jupyter.org/github/waltherg/notebooks/blob/master/2013-12-03-Crank_Nicolson.ipynb pour le code source du notebook
4. https://oar.imag.fr/

```
#Output of:
# curl -k https://api.grid5000.fr/stable/sites/lyon/clusters/nova/nodes/nova-1?pretty
{
  "architecture": {
    "nb_cores": 16,
    "nb_procs": 2,
    "nb_threads": 32,
    "platform_type": "x86_64"
  },
  "bios": {
    "configuration": {
      "cstate_clc": true,
      "cstate_enabled": true,
      "ht_enabled": true,
      "turboboost_enabled": true
    },
    "release_date": "09/08/2016",
    "vendor": "Dell Inc.",
    "version": "2.2.5"
  },
  "chassis": {
    "manufacturer": "Dell Inc.",
    "name": "PowerEdge R430",
(...)
```

Extrait 1: Exemple de sortie JSON de l'API de description du nœud Grid'5000 *nova-1.lyon.grid5000.fr*

sur les nœuds disponibles. Dans Grid'5000, ces jobs correspondent généralement à l'exécution d'une expérience, bien qu'il soit aussi possible d'accéder aux ressources interactivement ou de les réserver à l'avance.

Pour réaliser notre expérience, l'utilisateur va réserver, à la date du 14 novembre, 10 nœuds du cluster appelé *nova* pour une durée de 2 heures afin d'y exécuter un script `experiment.sh`. Pour cela, il utilise la commande suivante :

```
oarsub -t deploy -l nodes=10,walltime=2: -p "cluster='nova'" \
       -r "2017-11-14 09:00" "./experiment.sh"
```

3.1.2 Installation et configuration des ressources

À l'aide de l'outil Kadeploy[5], Grid'5000 laisse la possibilité à ses utilisateurs de déployer le système d'exploitation de leur choix sur les ressources qu'ils ont réservés.

L'équipe Grid'5000 fournit un ensemble d'environnements prêts à être déployés, mais les utilisateurs ont la possibilité de construire les leurs. Le déploiement du système d'exploitation s'effectue lorsque les ressources réservées sont disponibles pour l'utilisateur (le job OAR est en cours d'exécution). C'est dans ce cas la première chose réalisée dans un script d'expérience.

Pour l'expérience à réaliser, l'utilisateur va déployer un environnement Debian nommé `jessie-x64-base` sur l'ensemble des nœuds réservés, appelés *nova-1* à *nova-10* :

```
kadeploy3 -k -m nova-[1-10] -e jessie-x64-base
```

5. http://kadeploy3.gforge.inria.fr/

L'utilisateur est ensuite en mesure de se connecter avec SSH sur chaque nœud et ainsi d'accéder, en tant qu'utilisateur *root*, au système déployé. Il lui est ainsi possible d'installer les logiciels et d'appliquer les configurations qu'il désire.

Dans l'expérience, l'utilisateur décide d'utiliser le nœud *nova-1* comme nœud *request generator*. Il s'y connecte, y installe l'outil *Locust*. Le nœud *nova-2* est utilisé comme répartiteur de charge et l'utilisateur installe *nginx* sur cette machine. Les nœuds *nova-3* à *nova-10* restants sont utilisés comme *backend nodes*, l'utilisateur va y installer Docker et sur chaque nœud, instancier l'image *jupyter/nbviewer* mise à disposition par les auteurs de l'application NBViewer sur Docker Hub. L'image est instanciée autant de fois qu'il y a de cœurs disponibles sur la machine. Des redirections de port sont mises en place dans Docker pour rendre accessible l'application NBViewer depuis le réseau Grid'5000. Sur *nova-2*, *NGinx* est enfin configuré pour répartir les requêtes HTTP entrantes vers les conteneurs instanciés.

3.1.3 Exécution de l'expérience

Une fois les ressources matérielles correctement configurées, l'expérience peut être démarrée. L'utilisateur se connecte sur la machine *request generator* et y exécute Locust pour générer des requêtes HTTP telles que les enverraient un groupe de 100 clients durant 1 minute.

Les requêtes émises par Locust sont envoyées au *load balancer* qui les répartit vers les différentes instances Docker. Pour chacune d'entre elles, l'application NBViewer convertit le notebook en page HTML et retourne le résultat aux clients. Une fois l'exécution de Locust terminée, ce dernier retourne des statistiques sur les requêtes réalisées : moyenne et distribution des temps de réponse, taux d'échec, etc.

3.1.4 Collecte des résultats

La dernière étape de l'exécution d'un scénario d'expérience est la collecte des résultats. Les métriques concernées dépendent de l'expérience réalisée, mais Grid'5000 fournit aussi un ensemble d'information de performance pour tous les nœuds disponibles, accessibles au travers de son API.

Des informations, telles que la consommation énergétique d'un nœud, son utilisation réseau, ou encore sa charge CPU, ou son taux d'occupation mémoire sont ainsi mises à disposition et restent accessibles après l'expérience. Ces informations sont disponibles via l'API au format JSON, mais peuvent aussi être visualisées via des pages Web dédiées, tel qu'illustré dans la figure 3.

Dans l'expérience étudiée, l'utilisateur va collecter les résultats de Locust en copiant les fichiers de sortie générés dans son répertoire personnel. Il va également interroger l'API Grid'5000 pour obtenir les informations de consommation énergétique et de charge CPU observées sur les nœuds *load balancer* et *backend* durant l'exécution de l'expérience, par exemple en utilisant la commande suivante :

```
curl https://api.grid5000.fr/3.0/sites/lyon/metrics/power\
  /timeseries?only=nova-3,nova-4&from=1506345459&to=1506345489&resolution=1
```

Ainsi, l'utilisateur peut savoir si le nombre et la capacité des serveurs utilisés étaient suffisants pour traiter les requêtes des clients et estimer la consommation énergétique induise.

3.2 Mise en place d'une campagne d'expérimentation

L'exécution unique d'un scénario d'expérience n'est souvent pas suffisante à l'obtention des résultats désirés. Par exemple, il peut être nécessaire de répéter l'exécution d'un scénario pour obtenir des résultats statistiquement satisfaisants, de faire varier les conditions expérimentales pour comprendre leurs influences, etc. La répétition de ces expériences, c'est-à-dire la réalisation d'une campagne d'expérimentation, incite à la mise en place de certaines bonnes pratiques exposées ici.

Figure 3 - Interface du logiciel Kwapi , qui permet de visualiser en temps réel la consommation énergétique des nœuds lors d'une expérience

3.2.1 Script d'exécution

Lorsque l'expérience doit être exécutée de nombreuses fois, il n'est pas envisageable de réaliser ses différentes étapes manuellement ou de manière interactive. L'utilisateur devra écrire un script qui automatisera l'exécution de l'expérience sur Grid'5000. C'est aussi une bonne manière de documenter précisément ce qui a été fait, dans une démarche de *recherche reproductible*.

L'utilisation d'un langage de shell, bien que possible, montre souvent ses limites en ce qui concerne la gestion des erreurs, la maintenabilité du code, la manipulation des données de résultat. L'utilisation d'un langage de programmation de plus haut niveau est donc suggérée pour la réalisation de ces scripts.

Différentes bibliothèques ont été proposées pour interfacer l'utilisation de Grid'5000 avec un langage de programmation de haut niveau, notamment Execo [6] pour Python et Ruby-Cute [7] pour Ruby. La réalisation de telles bibliothèques est facilitée par l'API REST disponible sur Grid'5000, qui permet d'accéder à l'ensemble de ses fonctionnalités.

L'utilisateur va donc écrire les différentes étapes du scénario d'exemple dans un programme Python en s'appuyant sur la bibliothèque Execo. Il pourra ainsi découper les différentes étapes de l'expérience en différentes fonctions, comme montré dans les extraits 2 et 3.

6. http://execo.gforge.inria.fr/doc/latest-stable/index.html
7. https://github.com/ruby-cute/ruby-cute

```
if __name__ == "__main__":
    (client, lb, backends) = reserve_resources()
    install_resources(client, lb, backends)
    configure_scenario(client, lb, backends)
    start_scenario(client, lb, backends)
    collect_results(client, lb, backends)
    clean_scenario(client, lb, backends)
```

Extrait 2: Fonction principale du programme avec les différentes étapes du déroulement du scénario

```
def reserve_resources():
    logger.info("Submitting OAR job...")
    jobid, site = oarsub([
        (OarSubmission(resources="nodes=10",
                       walltime="1:00:00",
                       job_type="deploy",
                       sql_properties="cluster='nova'"
                ), 'lyon')
        ], abort_on_error=True)[0]

    logger.info("Getting OAR node list from %s@%s..." % (jobid, site))
    nodes = sorted(get_oar_job_nodes(jobid, site, timeout=180),
                   key=lambda h: str(h))

    logger.info("Node list is %s" % nodes)
    return nodes[0], nodes[1], nodes[2:]
```

Extrait 3: Fonction de réservation des ressources dans le script d'exécution du scénario

3.2.2 Paramètres de l'expérience

Les résultats d'une expérience dépendent de son environnement d'exécution. Afin d'étudier l'influence de cet environnement sur les résultats obtenus, on définit généralement différents paramètres qui sont susceptibles d'influer sur les résultats. On réalise l'expérience avec différentes combinaisons de ces paramètres et on compare les résultats obtenus.

Les paramètres à étudier peuvent être de natures variées. Dans notre scénario, nous étudierons l'influence du matériel utilisé (c'est-à-dire le *cluster* Grid'5000), le nombre de nœuds de *backend* utilisés, ainsi que le nombre de clients.

L'utilisation d'un langage de programmation de haut niveau, présenté dans le chapitre précédent, va grandement faciliter l'exécution répétée des scénarios associés à différentes combinaisons de paramètres. Les extraits 4 et 5 montrent ainsi les programmes Python précédemment exposés, modifiés pour exécuter l'expérience de manière répétée avec différentes combinaisons de paramètres.

```
if __name__ == "__main__":

    num_clients_p = [10, 100, 1000, 10000]
    num_backends_p = [1, 3, 5, 10]

    (client, lb, backends) = reserve_resources(num_backends=max(num_backends_p))
    install_resources(client, lb, backends)

    for num_clients, num_backends in product(num_clients_p, num_backends_p):
        configure_scenario(client, lb, backends[:num_backends])
        start_scenario(client, lb, backends[:num_backends], num_clients)
        collect_results(client, lb, backends[:num_backends],
            result_dir="./jres_xp/c%d-b%d" % (num_clients, num_backends))
        clean_scenario(client, lb, backends[:num_backends])
```

Extrait 4: Fonction principale du programme modifiée pour exécuter l'expérience plusieurs fois selon différents paramètres

```
def reserve_resources(cluster='nova', num_backends=3):
    logger.info("Submitting OAR job...")
    jobid, site = oarsub([
        (OarSubmission(resources="nodes=%d" % int(num_backends+2),
                       walltime="1:00:00",
                       job_type="deploy",
                       sql_properties="cluster='%s'" % cluster
                ), get_cluster_site(cluster))
        ], abort_on_error=True)[0]

    logger.info("Getting OAR node list from %s@%s..." % (jobid, site))
    nodes = sorted(get_oar_job_nodes(jobid, site, timeout=180),
                   key=lambda h: str(h))
    logger.info("Node list is %s" % nodes)
    return nodes[0], nodes[1], nodes[2:]
```

Extrait 5: La fonction de réservation des ressources modifiée pour prendre en compte les paramètres de l'expérience

3.2.3 Résultats

Une fois l'expérience exécutée un nombre de fois suffisant, avec les combinaisons de paramètres souhaitées, les résultats collectés peuvent être agrégés et interprétés.

La figure 4 montre ainsi les temps de réponse moyens obtenus par les clients et le temps de réponse maximal obtenu par 95% de ceux-ci, en fonction de leur nombre et du nombre de nœuds *backend* dans notre scénario. Comme attendu, le temps de réponse croît avec le nombre de clients, mais diminue si l'on augmente le nombre de *backend*. On constate qu'il est suffisant d'utiliser 3 *backends* pour obtenir des performances satisfaisantes lors de la présence de 1000 clients.

La figure 5 montre les consommations électriques instantanées moyennes cumulées pour le *load balancer* et les nœuds de *backends*, en fonction de leur nombre et du nombre de clients. Ces résultats permettent d'observer que l'utilisation du *load balancer* et de 3 serveurs de *backend* du même type que les noeuds du cluster *nova* entrainerait une consommation moyenne d'environ 600 Watts lorsque le nombre de clients ne dépasse pas la centaine et de 800 Watts lorsque celui-ci est de l'ordre du millier.

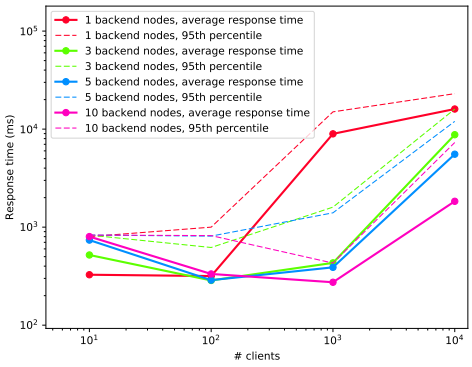

Figure 4 - Temps de réponse moyen et du dernier décile nécessaire à l'application NBViewer pour servir un notebook Jupyter à ses clients

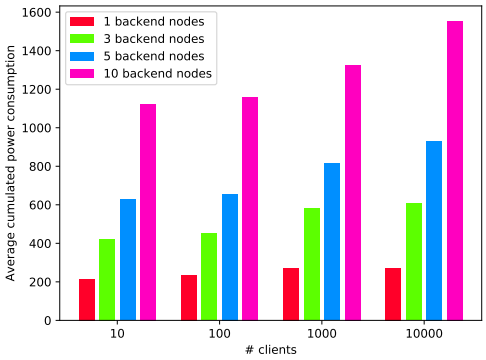

Figure 5 - Consommation instantanée moyenne cumulée par le load balancer et les nœuds backend

Le script d'exécution de l'expérience complet est disponible sur Internet [8].

4 Conclusion

Grid'5000 propose une plate-forme pour la réalisation d'expériences à grande échelle. Nous avons proposé de présenter les ressources matérielles et logicielles ainsi offertes aux utilisateurs. Nous avons notamment illustré la mise en œuvre concrète d'une expérience au travers d'un scénario d'exemple.

Bien que n'ayant couvert qu'une partie des possibilités offertes par la plate-forme, ce scénario d'exemple illustre de manière pratique la mise en place d'une expérience sur Grid'5000.

Nous espérons que les possibilités de la plate-forme présentées aux JRES susciteront l'intérêt et pourront répondre aux besoins de certains participants. Un programme *Open Access* [9] permet aux membres de la communauté JRES souhaitant réaliser leurs propres expériences sur Grid'5000 d'obtenir facilement et rapidement un compte sur l'infrastructure.

Nous souhaitons remercier les différents organismes qui soutiennent Grid'5000, ainsi que les membres des équipes techniques et scientifiques qui rendent possible l'existence et le fonctionnement de la plate-forme.

Bibliographie

[1] Bal H., Breton V., Brinkmann A., Danelutto M., Feng W., Keahey K., Kuonen P. et Parashar M.. Grid'5000 science advisory board report, 2014. https://www.grid5000.fr/ScientificCommittee/SAB%20report%20final%20short.pdf.

8. https://gitlab.inria.fr/delamare/jres17_experiment
9. https://www.grid5000.fr/open-access