Comparison on OpenStack and OpenNebula performance to improve multi-Cloud architecture on cosmological simulation use case

Eddy Caron, Lamiel Toch, Jonathan Rouzaud-Cornabas

Comparaison de performance entre OpenStack et OpenNebula et les architectures multi-Cloud: Application à la cosmologie.

E. Caron, L. Toch, J. Rouzaud-Cornabas

RESEARCH REPORT
N° 8421
December 2013
Project-Team Avalon

ISSN 0249-6399 ISRN INRIA/RR--8421--FR+ENG

Comparaison de performance entre OpenStack et OpenNebula et les architectures multi-Cloud: Application la cosmologie.

E. Caron[*†], L. Toch[‡], J. Rouzaud-Cornabas[‡]

Project-Team Avalon

Research Report n° 8421 — December 2013 — 20 pages

Abstract: With the increasing numbers of Cloud Service Providers and the migration of the Grids to the Cloud paradigm, it is necessary to be able to leverage these new resources. Moreover, a large class of High Performance Computing (HPC) applications can run these resources without (or with minor) modifications. But using these resources come with the cost of being able to interact with these new resource providers. In this paper we introduce the design of a HPC middleware that is able to use resources coming from an environment that compose of multiple Clouds as well as classical HPC resources. Using the DIET middleware, we are able to deploy a large-scale, distributed HPC platform that spans across a large pool of resources aggregated from different providers. Furthermore, we hide to the end users the difficulty and complexity of selecting and using these new resources even when new Cloud Service Providers are added to the pool. Finally, we validate the architecture concept through cosmological simulation RAMSES. Thus we give a comparison of 2 well-known Cloud Computing Software: OpenStack and OpenNebula.

Key-words: Cloud, IaaS, OpenNebula, Multi-Clouds, DIET, OpenStack, RAMSES, cosmology

[*] ENS de Lyon, France, Email: `FirstName.LastName@ens-lyon.fr`
[†] ENSI de Bourges, France, Email: `FirstName.LastName@ensi-bourges.fr`
[‡] INRIA, France, Email: `FirstName.LastName@inria.fr`

Comparison on OpenStack and OpenNebula performance to improve multi-Cloud architecture on cosmological simulation use case

Résumé : Avec l'augmentation du nombre de fournisseurs de service Cloud et la migration des applications depuis les grilles de calcul vers le Cloud, il est ncessaire de pouvoir tirer parti de ces nouvelles ressources. De plus, une large classe des applications de calcul haute performance peuvent s'excuter sur ces ressources sans modifications (ou avec des modifications mineures). Mais utiliser ces ressources vient avec le cot d'tre capable d'intragir avec des nouveaux fournisseurs de ressources. Dans ce papier, nous introduisons la conception d'un nouveau intergiciel HPC qui permet d'utiliser les ressources qui proviennent d'un environement compos de plusieurs Clouds comme des ressources classiques. En utilisant l'intergiciel DIET, nous sommes capable de dployer une plateforme HPC distribue et large chelle qui s'tend sur un large ensemble de ressources aggrges entre plusieurs fournisseurs Cloud. De plus, nous cachons l'utilisateur final la difficult et la complexit de slectionner et d'utiliser ces nouvelles ressources quand un nouveau fournisseur de service Cloud est ajout dans l'ensemble. Finalement, nous validons notre concept d'architecture via une application de simulation cosmologique RAMSES. Et nous fournissons une comparaison entre 2 intergiciels de Cloud: OpenStack et OpenNebula.

Mots-clés : Cloud, IaaS, OpenNebula, Multi-Clouds, DIET, OpenStack, RAMSES, cosmologie

1 Introduction

In recent years, distributed storage and computing have proved to be mandatory in IT. Internet Computing and Storage have considerably evolved, going from small isolated Physical Machines to large-scale cluster-like architectures driven by efficiency and scalability, one instance of these architectures is the "Clouds" [1, 2]. They aim being to be dynamically scalable and offer virtualized resources as a service over the Internet. Usually, most solutions deployed in Clouds focus on web applications and the load balanced is implicit or integrated. Nevertheless, Clouds can also be used in more computational-intensive domains as scalable computational resources. From a middleware point of view, Cloud infrastructures introduce new sets of resources with different features. Thus middleware environments should be extended to manage these platforms. Clouds is not yet an efficient solution for HPC but we can not neglected in future. In this paper, we propose a middleware to deal with a multiple Clouds (aka., *Sky* middleware), *i.e.,* a Cloud middleware for several Cloud platforms working together. We show how we can upgrade its design is based on an existing Grid middleware for managing virtual resources. Furthermore, we show that our proposal can be used to execute a complex scientific application for cosmological simulation, RAMSES [3], on different Cloud middleware without modifying it. We show that Cloud middleware must be taken into account when deploying a new private Cloud on user owned hardware. Indeed, even with similar virtualization facility, they do not expose the same performance. Moreover, installed Cloud middleware must also be taken into account when provisioning resources for the same reason as resource provisioning can be slower or faster depending of the Cloud middleware. Moreover a study of the scalability for this application on Cloud resources is provided.

Section 2 presents existing works on comparing Clouds and the application running on them. Then we present the existing Cloud APIs. We conclude the section by presenting multi-Clouds and brokering software. We use this state of the art to motivate our choice of software architecture. In Section 3, we present our extension of the DIET toolbox to support Cloud resources. First, we come back on the hierarchical architecture of DIET. Then we show how we have integrate a mechanism to support multi-Clouds environments. Furthermore, we explain how an application can leverage these mechanisms to transparently use resources coming from different Clouds. Moreover, the classical DIET scheduler is still used to efficiently schedule client requests to the best fitting instantiation of the requested application. In Section 5, we describe our experimental environment. We then highlight the performance variation between different Cloud middleware. We show the impact of this variability on our scientific application. Conclusion and future works are given in Section 6.

2 Related Work

In the first part of this section, we present the existing works on comparing Cloud software and their underlying components and especially the virtualization one. We show that all Cloud middleware and commercial Cloud Service Providers have their advantages and drawbacks. We motivate our new study focused on the comparison of OpenStack and OpenNebula to run cosmological simulation through RAMSES. The large number of Cloud platforms has resulted in a large number of different APIs. It is thus very difficult for the user and for the middleware integrator to choose which one to support. Indeed, we do not want to add new codes every time we add the support of a new Cloud middleware or a new public Cloud Service Provider.

Accordingly, in the Section 2.2, we show the different approaches to leverage the difficulty of managing different APIs to communicate with a set of Clouds.

In the last part of this section, we present the existing works on creating multi-Clouds.

Furthermore, we present a quick survey of Cloud brokering methods. Finally, we motivate our choice of integrating a Cloud brokering features in DIET.

2.1 Comparing Clouds

The common base components between all Clouds (Infrastructure as a Service layer) solutions is a virtualization layer or hypervisor. Numerous studies [4, 5] have compared the different hypervisors. The comparison of different hypervisors and Cloud middleware solutions is out of the scope of this paper. To avoid the noise due to different hypervisors when comparing Cloud middleware, we always use the same hypervisor, KVM.

At the Cloud middleware layer, several papers [6, 7, 8] have presented studies that compare the features of the different Cloud middleware. They only compares the functionalities and user communities. Other studies [9, 10] focus on the performance evaluation of scientific applications on a given Cloud.

Another group of papers presents benchmarks and benchmark tool suites for Clouds. They are used to compare different Cloud middleware and Cloud Service Providers. CloudCmp [11] proposes to compare different Cloud Service Providers. To do so, it measures the elastic computing, persistent storage, and networking services. C-Meter [12] allows to measure the time to acquire and release virtual computing resources. It also allows to compare different configurations of Virtual Machines. But, even on a single Cloud on a given template, it is requisite to closely study the performance given for the selected application. Indeed, variability is an important factor to study and can help to find the reason of it. Many papers [13, 14, 15, 16] study the performance variability of the same VM Template. This variability [17] is part due to what the other VMs are doing. Another factor of variability is the underlying hardware on which instances run [15]. These different hardware configurations can induce up to 60% performance variation.

Benchmarks are great to evaluate all the features of a Cloud and how it reacts with different workloads. Work on a real and targeted application is more complicated but more realistic. Thus we decided to focus on comparing Cloud middleware (OpenNebula and OpenStack) on a given hardware configuration and a cosmological simulation software called RAMSES. Accordingly we want to evaluate if on top of features, performance can be another factor to select a Cloud middleware.

2.2 Cloud APIs

There is 2 way to provide an abstraction to communicate with different Clouds providing different APIs. The Cloud adapters provide a library that easy the building of interfaces with different Clouds. The Cloud standardized APIs provide the scheme to respect in a Cloud library implementation.

Cloud Adapters Because of the fragmentation of Cloud APIs, software have been designed to cope with this issue: Simple Cloud API [18], Fog [19], Jclouds [20], Dasein Cloud [21], Apache Libcloud [22] and δ-Cloud [23]. We will focus on Libcloud and δ-Cloud that are the most active projects with a large set of features available.

[22] is a common library for controlling VMs in the Cloud. Libcloud currently supports a couple dozen Cloud providers. The principal drawback about LibCloud is its exclusive use of Python as the programming language to connect drivers to vendor APIs.

δ-Cloud offers a standardized API definition for Infrastructure as a Service (IaaS) Clouds with drivers for a range of different Clouds. It can be seen as an API for APIs. The δ-Cloud API is designed as a RESTful web service and comes with client libraries for all major programming languages. Red Hat's δ-Cloud, leverages a REST based API that offers more flexibility than the LibCloud library for the purpose of migrating software deployments from one Cloud infrastructure to another. Moreover, δ-Cloud API is compatible with Cloud-oriented storage systems such as S3 and Cloudfiles. The conception of the δ-Cloud API as a web service instead of a library makes it possible to operate the δ-Cloud server in one of 2 base configurations - close to the user, such as on a local computer/LAN or close to the Cloud provider's native API, which is particularly interesting for private Clouds. Of course, providers can also use δ-Cloud directly as their sole interface.

Cloud Standards Another type of API is provided through the standardization efforts of several standards [24, 25]. We introduce briefly 2 major standardizations: OCCI and the work from DMTF.

The Open Cloud Computing Interface (OCCI) began in March 2009 and comprises a set of open community-lead specifications delivered through the Open Grid Forum, which define how infrastructure service providers can deliver their Cloud resources through a standardized interface. The working group has a membership of over 250 members and includes numerous individuals, industry and academic parties. OCCI is a RESTful Protocol and API for all kinds of tasks management. OCCI was originally initiated to create a remote management API for IaaS model based Services, allowing for the development of interoperability tools for common tasks including deployment, autonomic scaling and monitoring. It has since evolved into an flexible API with a strong focus on interoperability while still offering a high degree of extensibility. OCCI is designed to meet the demand on interoperability, portability and integration (to note that it serves as an integration point for other standards efforts including DMTF, IETF, and SNIA). δ-Cloud implements this standard as the δ-Cloud driver for OpenNebula is an implementation based on the OGF OCCI API.

The DMTF (Distributed Management Task Force) is an industry organization that develops, maintains and promotes standards for systems management in enterprise IT environments. Many standardization efforts around the Cloud are done. DMTF's Cloud Management Working Group focuses on standardizing interactions between Cloud environments by developing Cloud management use cases, architectures and interactions. The Open Virtualization Format (OVF) has been designed by this organization. As well the Cloud Infrastructure Management Interface (CIMI) is a document to define a logical model for the management of resources within the Infrastructure as a Service domain. The standards produced are fruitful for IaaS developers.

Summary To conclude this study, it is very difficult to forecast which Cloud APIs will be the most commonly used. Thus we have choose δ-Cloud for the large scope of functionalities and the number of supported Cloud IaaS. Moreover δ-Cloud is not dedicated to a specific language. Finally through the support of the Red Hat company, this API should be sustainable.

2.3 Multi-Clouds middleware and Cloud Brokers

With the raising of customer's request of computational resources, the main problem that public Cloud providers have to address is the scalability. Now, one public Cloud for satisfying all demands are not enough, that is why in the literature, one spokes about *Cloud Federation* or *Federated Clouds*. In a Cloud Federation, small, medium and large Cloud providers put their Cloud together to gain in scalability. This federation of Clouds raises problem such as placement. Indeed, how to choose the Cloud among the federation of heterogeneous Clouds in terms of computational power, availability capacity.

Commercial brokering solutions exist: RightScale[1] or SpotCloud[2] For example, RightScale solution allows to manage a Cloud application that is spread between different Cloud Service Providers. It allows to automatically adapt the application based on the load on the applications. RightScale provides a single management interface that interacts with different Clouds such as Amazon Web Services, Rackspace, Windows Azure, CloudStack and OpenStack.

Open-source brokering solutions also exist. Aeolus[3] uses δ-Cloud to communicate with the Clouds. Its functionalities are less advanced than RightScale.

Most of the academic works on Cloud brokering is focused toward resources provisioning algorithms [26, 27, 28]. Nonetheless, RESERVOIR project [29] has proposed a multi-Cloud broker software. CLEVER project [30] has proposed a similar software.

We want to leverage all the multi-Cloud resources while hiding the complexity of managing them. So we choose to use the DIET open source project [31] to hide this complexity. We choose an approach similar to the Aeolus one as we also use δ-Cloud. Commercial, open-source or academic Cloud federation or brokering software would have added an additional layer without a gain of features. Nonetheless, DIET has been built so it is easy to change all the resources allocation algorithms. Accordingly, it is easy to test different Cloud resource provisioning algorithms with our proposal.

3 Diet: a Novel Cloud Toolbox

The DIET open source project [31] is focused on the development of a scalable middleware with initial efforts relying on distributing the scheduling problem across a hierarchy of agents. At the top of it sits the Master Agent (MA), with Service Daemon (SED) agents at the leaf level. Over the last few years, the Cloud phenomenon has been gaining more and more traction in the industry and in research communities because of its qualities. Of particular interest are its on-demand resource-provisioning model and its pay-as-you-go billing approach. We believe these features would make highly interesting additions to DIET.

3.1 The Diet Architecture

The DIET component architecture is hierarchically structured for improved scalability as illustrated in Fig. 1. The DIET toolkit is implemented in CORBA [32] and thus benefits from the many standardized, stable services provided by freely-available, high-performance CORBA implementations.

The DIET framework is composed of several elements. The first element is a **Client**, an application that uses the DIET infrastructure to solve problems using a GridRPC approach [33]. The second is the **SeD** (**Server Daemon**) which acts as the service provider, exposing functionalities through a standardized computational service interface; a single SED can offer any number of computational services. The third element of the DIET architecture, the **agents**, facilitates the service location and invocation interactions between clients and SEDs. Collectively, a hierarchy of agents provides higher-level services such as scheduling and data management. These services become scalable by distributing them across a hierarchy of agents composed of one or more **Master Agents (MA)** and several **Local Agents (LA)**.

The first step towards the Cloud was to enable DIET to benefit from on-demand resources. This should be done at the platform level and stay transparent to the user. In [34] we described

[1]`http://www.rightscale.com/`
[2]`http://www.spotCloud.com/`
[3]`http://www.aeolusproject.org/`

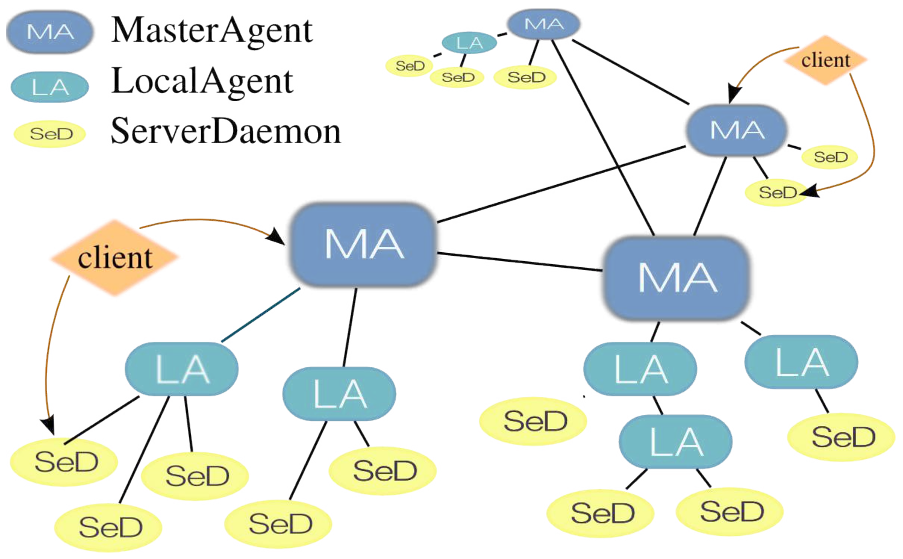

Figure 1: DIET multi-hierarchy.

how the Eucalyptus [35] Cloud platform can be used by DIET. Eucalyptus has the same management interface as Amazon EC2, but unlike EC2, it allows customized deployments on the user's hardware. There is a large number of ways in which the DIET platform can be connected to the Eucalyptus platform to give the required result. On one side, the DIET platform itself can be virtualized inside Eucalyptus Virtual Machines which would allow for dynamism at every level in the DIET platform. On the other side, the DIET platform sits completely outside of the Eucalyptus platform and treats the Cloud only as a provider of compute resources. We have implemented the latter as an extension of DIET that allows it to use on-demand resources when handling client service requests. This opened the path towards new researches around Grids and Cloud resource provisioning, data management over these platforms and hybrid scheduling in general. Thanks to this first proof of concept we have designed a new architecture described in the following sections.

3.2 The Diet Cloud Architecture

DIET already implements many prerequisites, such as service calls, scalable scheduling and data management. This allowed us to implement a Cloud middleware with minimal effort. In this section, we describe the architecture that upgrades DIET to a multi-Cloud middleware designed to interact multi-Cloud platforms. This architecture is shown in Fig. 2 where hierarchical agent connection is represented by the DIET logo. The aim of the SED Cloud is to provide a component that deals with a large number of Cloud middleware and Cloud Service Providers. Thus it hides the complexity and heterogeneity of the Cloud API layer (thanks to δ-Cloud [23]). Nevertheless the DIET SED Cloud could be interfaced with different APIs if it is required but we try to minimize this number of interface through a good choice of API. DIET can benefit from the IaaS capabilities and manage Virtual Machines. Nevertheless Virtual Machine management decisions will be taken according to the scheduler and the required SLA (Service Level Agreement). DIET has successfully interacted with Amazon EC2, thus the feasibly of this approach was validated in [34].

The DIET SED Cloud can bootstrap a Cloud instance, thus some Virtual Machines will be

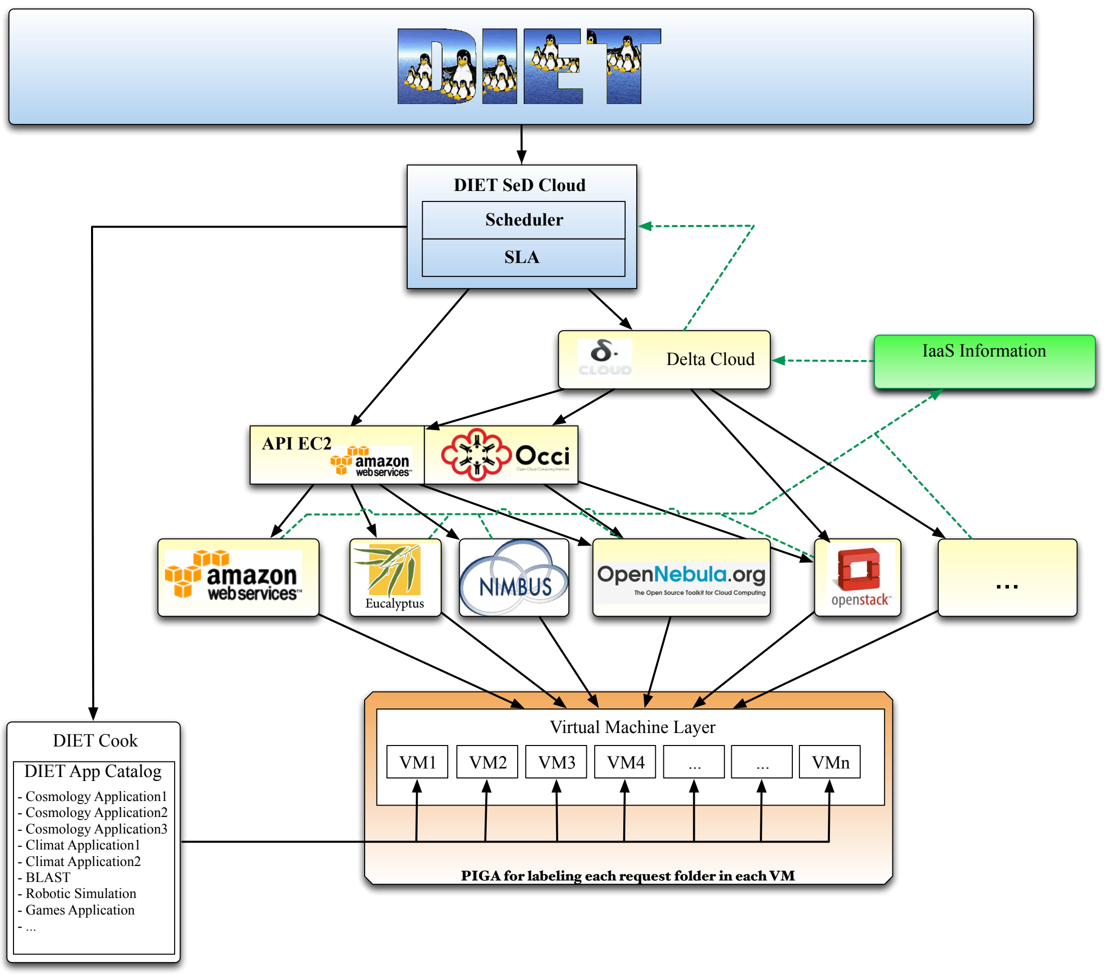

Figure 2: The Diet Cloud architecture.

available for computations. However applications must be deployed on these VM so that they can be added to the Diet platform. The Diet cook component enables the automatic installation of applications inside the provisioned VMs. When VMs are available, the Diet cook manager deploys a set of applications and launches the service registration into Diet. Many tools exist that ease automated application deployment such as Puppet [36] or chef [37]. After this step a classical Diet platform is available. Thank to the Diet SeD Cloud, a Diet architecture can expand or reduce the number of its compute resources.

4 Deploying ramses on multi-Clouds

Fig. 4 shows the classical and hierarchical architecture of Diet with **MA** and **LA** augmented with **SeDs Cloud** that offers Cloud services like instantiation of Virtual Machines end destruction of Virtual Machines.

4.1 ramses

RAMSES is a typical computational intensive application used by astrophysicists to study the formation of galaxies. It is used, among other things, to simulate the evolution through cosmic time of a collisionless, self-gravitating fluid called dark matter. Individual trajectories of macro-particles are integrated using a state-of-the-art N body solver, coupled to a finite volume Euler solver, based on the Adaptive Mesh Refinement techniques. The computational space is decomposed among the available processors using a mesh partitioning strategy based on the Peano-Hilbert cell ordering. RAMSES is a parallel program based on MPI (Message Passing Interface).

RAMSES reads the initial conditions from Fortran binary files, which are generated using a modified version of the GRAFIC2 code. This application generates Gaussian random fields at different resolution levels, consistent with current observational data obtained by the WMAP5 satellite observing the CMB radiation. The generated IC can be of 2 sorts. Either it contains a single level of resolution, i.e., the universe has a “homogeneous” distribution of dark matter, these IC are used to perform the initial low resolution simulation of a zoom re-simulation. Or, it can generate multiple levels of resolution, i.e., several nested “boxes”, like matryoshka dolls. These nested boxes add more particles, and thus more precision locally, on a point of interest in the universe. These are used in the zoom part of the simulation. The outputs of RAMSES are twofold. Periodically, RAMSES outputs backup files so as to be able to restart a simulation at a given time step. The second kind of output is of course the result of the simulation. The user defines time epochs at which she would like to obtain a snapshot of the simulation, i.e., a description of all particles (their position, mass, velocity, . . .), RAMSES will then output as many snapshots as requested.

The workflow is depicted in Fig. 3. It is divided into 2 main parts: the dark matter simulation (i.e., IC generation with GRAFIC2, and the simulation itself with RAMSES) and the “one-shot” post-process (i.e., HALOMAKER and TREEMAKER), followed by the parameter sweep part with GALAXYMAKER and MOMAF. Three DIET services are dedicated to executing this workflow. The first one deals with the dark matter simulation itself along with HALOMAKER and TREEMAKER post-processing, and can possibly run GALAXYMAKER, if no parameter sweep is requested. The second service executes an instance of GALAXYMAKER, thus, during the parameter sweep part, we have x—parameters— calls to the GALAXYMAKER service (x being the number of files that TREEMAKER created, and parameters the set of tested parameters). The same number of calls holds for the last service, which is in charge of running MOMAF. This workflow is submitted by the DIET to the Cloud infrastructure (the DIET agent in charge of managing and scheduling workflows), which in turn manages the execution of the different services.

4.2 VMs Provisioning

In the general case, the client connects itself to the **MA**, and make a request *req*: “I want m Virtual Machines with the hardware profile x”. As input parameters of the RAMSES workflow, we give the number of MPI processes to use, which is equal to the number of Virtual Machines, the path to the output directory on the NFS share and a set of specific parameters for the RAMSES application. A SED Cloud is selected by the VM provisioning algorithm. As other algorithms within the DIET toolbox, it is easy to add new ones thanks to a plugable approach. Then the selected SED Cloud requests m Virtual Machines with the hardware profile x to the corresponding Cloud platform. When the provisioning is completed, *i.e.* the VMs are running, the SED Cloud returns to the client a file containing all IP addresses of these Virtual Machines. After this step the client can use the Virtual Machines to install **SeDs** or any other applications.

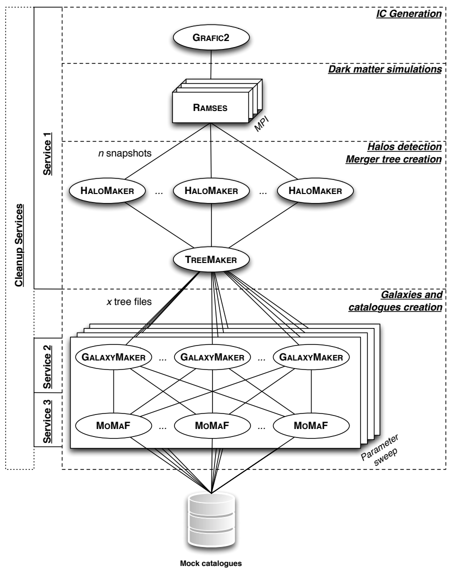

Figure 3: The RAMSES computational workflow

4.3 Application Deployment on VMs

To make easy the usage of their services and Cloud resources, **SeD** developers may design a xSeD[4] which directly uses the file containing IP addresses, connects itself to the Virtual Machines and leverages computational and networking resources. This DIET xSeD is a kind of wrapper to software preinstalled or automatically installed in the Virtual Machines through the DIET cook (see Fig. 2). Applying this approach, a xSeD can be launched on-demand, *i.e.* when a client who needs this particular xSED. Fig. 4 shows an example of a DIET Cloud hierarchy using 3 Cloud platforms, 2 based on the OpenStack middleware and 1 based on the OpenNebula middleware. Here, $x = ramses$ and it is the name of the cosmological application. Furthermore the xSeD is launched and connects to a Virtual Machine, it behaves as a classical SeD which raises information useful for task scheduling. So this architecture presents twofold advantages over a federation of Cloud: Virtual Machine provisioning and task scheduling.

[4] x denotes an arbitrary name of any software package.

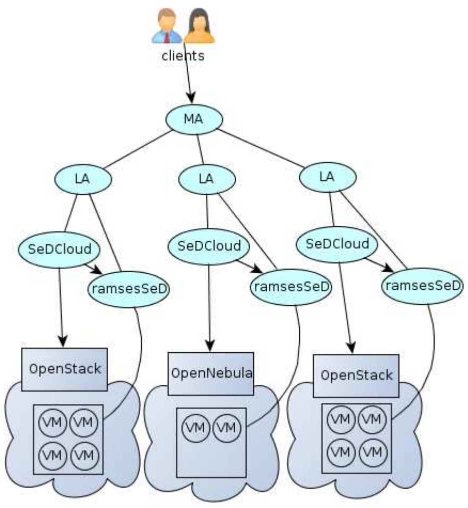

Figure 4: The DIET Cloud architecture using a cosmological application.

5 Experiments

Here, we study the behavior of 2 well-known Cloud middleware: OpenStack [38] Grizzly (2013.1.2) and OpenNebula [39] version 3.8.3. Experiments are carried out on the Grid'5000 platform [40].

5.1 Cloud Testbeds

We use 2 hardware configurations to compare both middleware.

5.1.1 Hardware configuration 1

We use 5 Physical Machines to build a Cloud infrastructure: 1 for the Cloud middleware and 4 for hosting VMs. Each Physical Machine has an Intel(R) Xeon(R) CPU X3440 2.53GHz with 4 cores and 16 GB of RAM. The 5 Physical Machines are connected to a switch with a bandwidth capacity of 1Gbit/s.

5.1.2 Hardware configuration 2

We use 4 Physical Machines to build a Cloud infrastructure: 1 for the Cloud middleware and 3 for hosting VMs. Each Physical Machine has 2 IntelXeonCPUs L5420 2.5GHz with 4 cores and 32 GB of RAM. The 4 Physical Machines are connected to a switch with a bandwidth capacity of 1Gbit/s.

5.1.3 Common configuration

We use only one 1 Physical Machine for the deployment of DIET. The DIET deployment is made of 1 MA and 1 SED with a service for Virtual Machine instantiations and 1 SED with a service for Virtual Machine destruction. All the Physical Machines are using Ubuntu 12.04 x64 as Operating System. For OpenStack, we use the default configuration. By default, OpenNebula is not configured to obtain good performance. We have configured it to use the virtio [41] module of Linux kernel in order to obtain almost-native network transfer between Virtual Machines and deactivate the disk cache of Virtual Machines.

5.2 Experimental parameters for ramses

Each RAMSES execution is using x VMs ($1 \leq x \leq 15$) for running the workflow and 1 VM for sharing files (NFS). The Virtual Machine image that contains the RAMSES software has a size of about 2 GB and the Virtual Machine image that contains the NFS server is about 1GB.

Our experimental campaign has 2 parameters: Cloud middleware used and number of VMs on which RAMSES runs. Each combination of parameters is executed 4 times and between each the whole Cloud testbed is destroyed and reinstalled from scratch. This destruction of the testbed is performed in order to avoid some residual image files of VMs in the Physical Hosts that could disturb the experiments. We have evaluate 2 different Cloud middleware: OpenStack and OpenNebula. Furthermore, we have evaluated RAMSES using from 1 to 15 VMs. Each experiment is composed of 2 steps: 1) instantiating and configuring VMs and 2) executing a RAMSES simulation. We measure the duration of these 2 steps. Indeed, we are interested in these 2 metrics since they are crucial to take an accurate choice between Cloud middleware in a HPC context.

5.2.1 Initial conditions for ramses simulations 1

First we want to compare the 2 Cloud middleware parameters for the cosmological simulations known as *initial conditions* while keeping the hardware 1. Then we want to compare the 2 Cloud middleware with these initial conditions on the hardware 2.

5.2.2 Initial conditions for ramses simulations 2

We also want to compare the 2 Cloud middleware with other parameters for the cosmological simulations known while keeping the hardware 1. These parameters involve more intensive computations.

The instantiating and configuring VMs step is composed of 4 sub-steps:

1. Instantiating and configuring a VM for the NFS server
2. Instantiating x VMs for RAMSES
3. Configuring the NFS share in each RAMSES VMs
4. Launching the RAMSES SeDs and link them to their corresponding RAMSES VM.

Between each experimentation, all the VMs are terminated. When the first step is completed, we launch the RAMSES workflow.

5.3 Experiment Results

Fig. 5 shows the time to instantiate Virtual Machines on OpenStack and OpenNebula. This time is measured between the date when the Cloud middleware starts to instantiate Virtual Machines and the date when they are ready to accept SSH connections. When instantiating between 1 and 4 VMs, OpenNebula is faster than OpenStack. But, when instantiating more than 5 VMs, OpenStack is a lot faster. Indeed, the instantiation time for OpenNebula is linear with the number of Virtual Machines whereas with OpenStack this time is not linear.

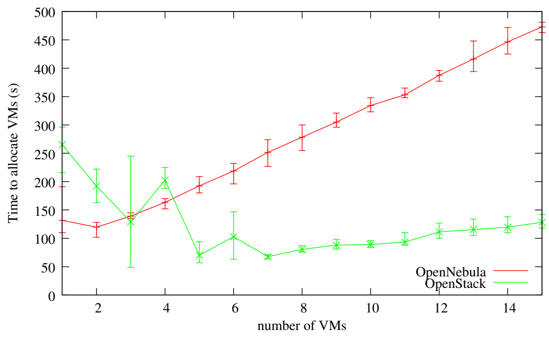

Figure 5: Time to allocate VMs with cosmological initial conditions 1 and hardware 1

These observations highlight the behaviors of both Cloud middleware. Indeed, with OpenStack, when n Virtual Machines have to be instantiated and scheduled on m Physical Machines, m copies of the Virtual Machine image are send to the m Physical Machines and they stay in cache on the Physical Machines. On the contrary, in this phase, OpenNebula sent n copies of the Virtual Machine image into the m Physical Machines. The problem is that, when Virtual Machines are destroyed, images in the cache of the Physical Machines are also destroyed. Accordingly when a Virtual Machine image needs to be deployed on a Physical Machine where it has already been deployed, the entire image will be transferred against from the controller Physical Machine to the Physical Machines that host VMs.

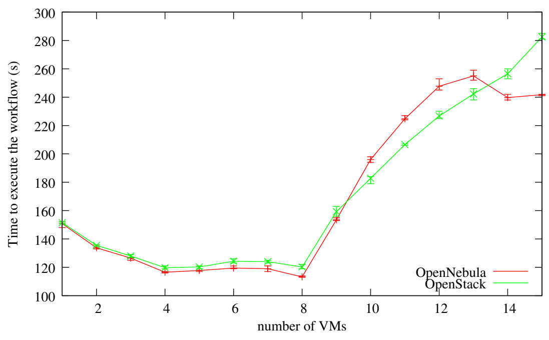

Figure 6: Time to run a RAMSES workflow with cosmological initial conditions 1 and hardware 1

We can notice that the curve for OpenStack presents some picks and larger error-bars than OpenNebula. They come from the fact that OpenStack checks whether a Virtual Machine image is present on the chosen Physical Machine or not. If an image is not present the middleware sends the image to the Physical Machine.

Fig. 6 shows the time to run a Ramses workflow on Virtual Machines instantiated with OpenStack and OpenNebula. This time is measured between the date when the computational workflow is submitted to DIET and the date when all computations are finished. We notice, that for any number of Virtual Machines, OpenStack and OpenNebula has the same performance. That seems to be coherent, since both Cloud middleware rely on the libvirt and KVM layers.

Fig. 7 shows the time to run an experiment. This time is measured between the date when the Cloud middleware starts to instantiate Virtual Machines and the date when all computations

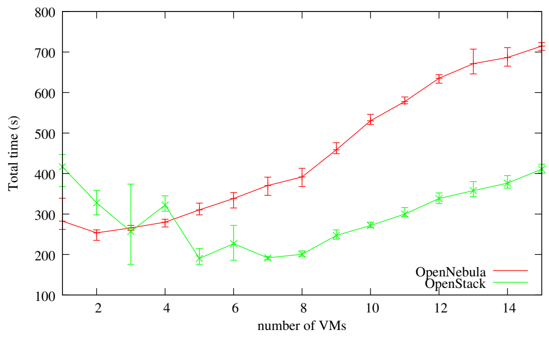

Figure 7: Time to allocate VMs and run a RAMSES workflow with cosmological initial conditions 1 and hardware 1

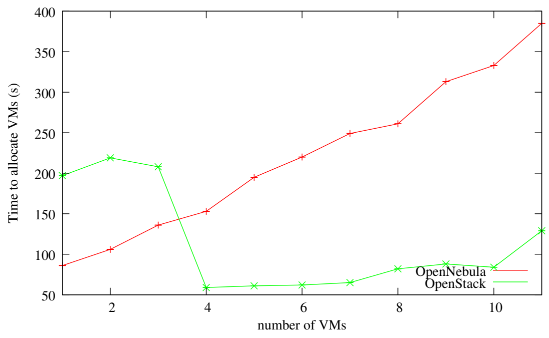

Figure 8: Time to allocate VMs with cosmological initial conditions 2 and hardware configuration 1

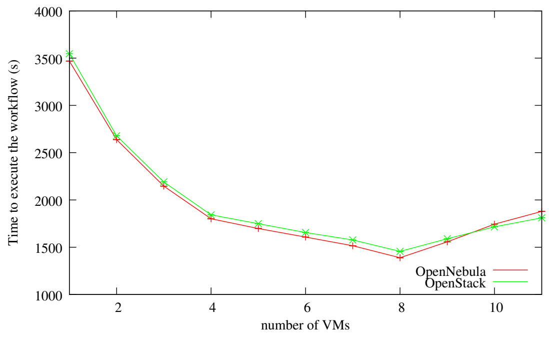

Figure 9: Time to run a RAMSES workflow with cosmological initial conditions 2 and hardware configuration 1

are finished. We can notice that OpenStack is more suitable for the HPC than OpenNebula due to a faster instantiation of large (bigger than 4) number of VMs.

Fig. 8 shows the time to allocate VMs. We notice that the behaviors of both Cloud middleware are not changed. As we expected the initial conditions have no impact on VM allocations.

Fig. 9 shows the time to run a workflow. It confirms that with these initial conditions the

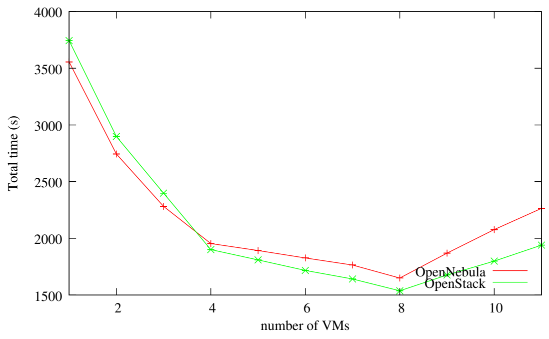

Figure 10: Time to allocate VMs and run a RAMSES workflow with cosmological initial conditions 2 and hardware configuration 1

runtime is more important than the simulations presented with the Fig. 6. A computation with 1 VM require almost 1 hour in this case against a hundred of seconds in the previous case. It also shows that the performance of the RAMSES executions similar for both Cloud middleware. Fig. 10 shows the sum of the allocation time and the runtime. With these new RAMSES parameters, the allocation time is negligible relatively to the runtime. Accordingly the slower allocation time of OpenNebula in comparison with OpenStack is almost negligible on the overall execution time. So in this case, we can say that for high computation intensive application the choice of a Cloud middleware does not matter a lot.

Fig. 11 shows the time to allocate VMs on the hardware configuration 2. We notice that the behaviors of both middleware are not changed even if we have used another hardware architecture. Fig. 12 shows the time to run a workflow. We notice, that for this hardware and any number of Virtual Machines, both OpenStack and OpenNebula have almost the same performance. Fig. 13 shows the sum of the allocation time and the runtime and for this case it shows that the best Cloud middleware to choose is still OpenStack when using a large number of VMs. So this set of experiments shows that the comparison between the 2 Cloud middleware is independent from the hardware architecture on which they are deployed.

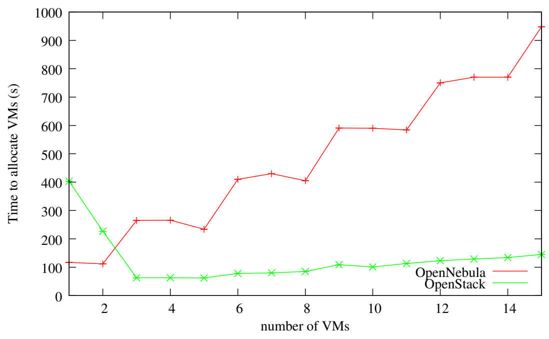

Figure 11: Time to allocate VMs with cosmological with initial conditions 1 and hardware configuration 2

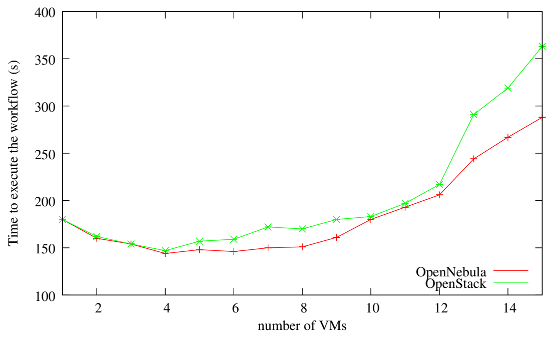

Figure 12: Time to run a RAMSES workflow with initial conditions 1 and hardware configuration 2

6 Conclusion

Our objective was to propose a multi-Cloud middleware that is able to deal with a set of Cloud middleware and Cloud Service Providers, thus seeing as a single pool of resources the multi-Clouds environment. In this paper, we have presented our open-source multi-Clouds middleware, DIET, and how it has been extended to access virtualized resources. Furthermore, we have also shown how a cosmological application, RAMSES, can easily take advantages of these new resources thank to DIET. Finally, we have compared the performance of 2 Cloud middleware and show that OpenStack is slightly better than OpenNebula due to a smaller instantiation time. This piece of information is important for further works, since it can be used to build a scheduler probe inside the SeD Cloud. The goal will be to choose the best Cloud platforms to instantiate Virtual Machines for the DIET services. Indeed, it motivates the need of providing monitoring information of the Cloud resources to the multi-Cloud middleware in order to take better choices when provisioning new Cloud resources. Our future works will be to work on algorithms for multi-Clouds that take into account these information and provides the adequate Cloud resources for the application based on users requirements and Cloud feedback.

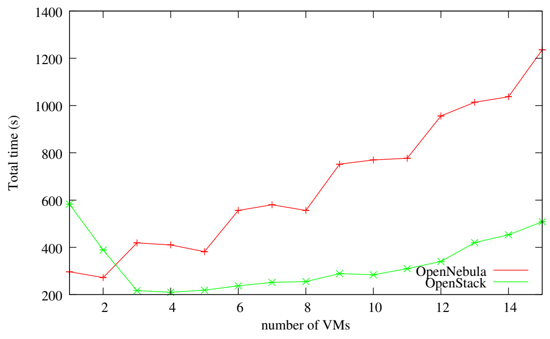

Figure 13: Time to allocate VMs and run a Ramses workflow with initial conditions 1 and hardware configuration 2

Acknowledgment

Part of this work was done thanks to the financial support of the Celtic+ project Seed4C (Eureka Celtic+ CPP2011/2-6).

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the Inria ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[2] L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Linder, "A Break in the Clouds: Towards a Cloud Definition," ACM SIGCOMM Computer Communication Review, vol. 39, no. 1, pp. 50–55, 2009.

[3] Y. Caniou, E. Caron, B. Depardon, H. Courtois, and R. Teyssier, "Cosmological simulations using grid middleware." in IPDPS. IEEE, 2007, pp. 1–8.

[4] J. Che, Q. He, Q. Gao, and D. Huang, "Performance measuring and comparing of virtual machine monitors," in Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on, vol. 2, 2008, pp. 381–386.

[5] K. Adams and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," SIGOPS Oper. Syst. Rev., vol. 40, no. 5, pp. 2–13, Oct. 2006. [Online]. Available: http://doi.acm.org/10.1145/1168917.1168860

[6] C. Baun, M. Kunze, J. Nimis, and S. Tai, "Open source cloud stack," in Cloud Computing. Springer Berlin Heidelberg, 2011, pp. 49–62. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-20917-86

[7] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: Openstack and opennebula," in Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, 2012, pp. 2457–2461.

[8] G. von Laszewski, J. Diaz, F. Wang, and G. Fox, "Comparison of multiple cloud frameworks," in Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, 2012, pp. 734–741.

[9] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in Proceedings of the 2008 ACM/IEEE conference on Supercomputing, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 50:1–50:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=1413370.1413421

[10] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," in Cloud Computing: First International Conference, CloudComp 2009, Munich, Germany, October 19-21, 2009, Revised Selected Papers, ser. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (LNICST), M. Diaz,

D. Avresky, A. Bode, C. Bruno, and E. Dekel, Eds., vol. 34. Springer, 2010, pp. 115–131. [Online]. Available: http://www.st.ewi.tudelft.nl/~iosup/ec2perf-sci-comp09cloudcomp.pdf

[11] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879143

[12] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, "C-meter: A framework for performance analysis of computing clouds," in Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 472–477. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2009.40

[13] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, "More for your money: exploiting performance heterogeneity in public clouds," in Proceedings of the Third ACM Symposium on Cloud Computing, ser. SoCC '12. New York, NY, USA: ACM, 2012, pp. 20:1–20:14. [Online]. Available: http://doi.acm.org/10.1145/2391229.2391249

[14] A. Iosup, N. Yigitbasi, and D. Epema, "On the Performance Variability of Production Cloud Services," in Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on, may 2011, pp. 104 –113.

[15] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui, "Exploiting Hardware Heterogeneity Within the Same Instance Type of Amazon EC2," in Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing, ser. HotCloud'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342763.2342767

[16] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance," Proc. VLDB Endow., vol. 3, no. 1-2, pp. 460–471, Sep. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1920841.1920902

[17] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift, "Resource-Freeing Attacks: Improve Your Cloud Performance (At Your Neighbor's Expense)," in Proceedings of the 2012 ACM conference on Computer and communications security, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 281–292. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382228

[18] Z. T. Inc., "Simple Cloud API. http://simplecloud.org/."

[19] Engine Yard, "Fog, the Ruby cloud service library. http://fog.io."

[20] "Apache jclouds," http://jclouds.incubator.apache.org/.

[21] "The dasein cloud api," http://dasein-cloud.sourceforge.net/.

[22] The Apache Software Foundation, "Apache Libcloud. http://libcloud.apache.org/."

[23] "Deltacloud api," http://deltacloud.apache.org/.

[24] P. Harsh, F. Dudouet, R. Cascella, Y. Jegou, and C. Morin, "Using open standards for interoperability issues, solutions, and challenges facing cloud computing," in Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm), 2012, pp. 435–440.

[25] M. Waschke, "Cloud-specific standards," in Cloud Standards. Apress, 2012, pp. 289–331.

[26] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, 2009, pp. 103–110.

[27] M. Andreolini, S. Casolari, M. Colajanni, and M. Messori, "Dynamic load management of virtual machines in cloud architectures," in Cloud Computing, ser. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, D. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, Eds. Springer Berlin Heidelberg, 2010, vol. 34, pp. 201–214. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12636-9_14

[28] E. Elmroth, F. Marquez, D. Henriksson, and D. Ferrera, "Accounting and billing for federated cloud infrastructures," in Grid and Cooperative Computing, 2009. GCC '09. Eighth International Conference on, 2009, pp. 268–275.

[29] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente, "From infrastructure delivery to service management in clouds," Future Gener. Comput. Syst., vol. 26, no. 8, pp. 1226–1240, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.02.013

[30] F. Tusa, A. Celesti, M. Paone, M. Villari, and A. Puliafito, "How clever-based clouds conceive horizontal and vertical federations," in Proceedings of the 2011 IEEE Symposium on Computers and Communications, ser. ISCC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 167–172. [Online]. Available: http://dx.doi.org/10.1109/ISCC.2011.5984011

[31] E. Caron and F. Desprez, "DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid," International Journal of High Performance Computing Applications, vol. 20, no. 3, pp. 335–352, 2006.

[32] O. M. Group, The Common Object Request Broker (CORBA): Architecture and Specification. Object Management Group, 1995.

[33] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova, "A GridRPC Model and API for End-User Applications," in GFD-R.052, GridRPC Working Group, Jun. 2007.

[34] E. Caron, F. Desprez, D. Loureiro, and A. Muresan, "Cloud Computing Resource Management through a Grid Middleware: A Case Study with DIET and Eucalyptus," in IEEE International Conference on Cloud Computing (CLOUD 2009). Bangalore, India: IEEE, Septembre 2009, published In the Work-in-Progress Track from the CLOUD-II 2009 Research Track.

[35] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Userful Systems," Computer Science Department, University of California, Santa Barbara, Tech. Rep. 2008-10, 2008.

[36] A. Bertolino, G. De Angelis, and A. Polini, "Automatic Generation of Test-Beds for Pre-Deployment QoS Evaluation of Web Services," in Proceedings of the 6th International Workshop on Software and Performance, WOSP 2007, Buenes Aires, Argentina, February 5-8, 2007, V. Cortellessa, S. Uchitel, and D. Yankelevich, Eds. ACM, 2007, pp. 137–140.

[37] P. Knoop, J. Hardin, T. Killeen, and D. Middleton, "The CompreHensive collaborativE Framework (chef)," in AGU Fall Meeting Abstracts, vol. 1, 2002, p. 13.

[38] "Openstack opensource cloud computing software," http://www.openstack.org/.

[39] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures," Computer, vol. 45, no. 12, pp. 65–72, 2012.

[40] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard, "Grid'5000: a large scale and highly reconfigurable grid experimental testbed," in Grid Computing, 2005. The 6th IEEE/ACM International Workshop on, 2005, pp. 8 pp.–.

[41] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," SIGOPS Oper. Syst. Rev., vol. 42, no. 5, pp. 95–103, Jul. 2008. [Online]. Available: http://doi.acm.org/10.1145/1400097.1400108

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399