A Ring to Rule Them All - Revising OpenStack Internals to Operate Massively Distributed Clouds

A. Lebre, J. Pastor, Frederic Desprez

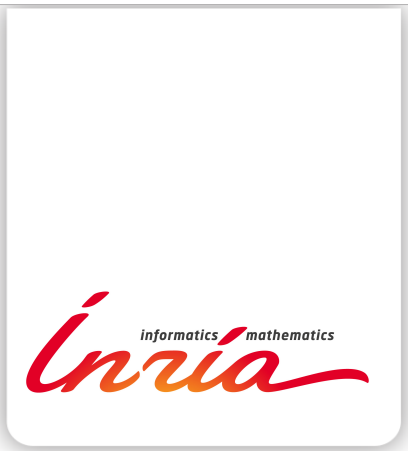
TECHNICAL REPORT

N° 480

February 2016

Project-Team DISCOVERY IPL

ISSN 0249-0803 ISRN INRIA/RT--480--FR+ENG



A Ring to Rule Them All - Revising OpenStack Internals to Operate Massively Distributed Clouds

A. Lebre*†, J. Pastor*†, Frederic Desprez*

Project-Team DISCOVERY IPL

Technical Report n° 480 — February 2016 — 21 pages

Abstract:
The deployment of micro/nano data-centers in network point of presence offers an opportunity to deliver a more sustainable and efficient infrastructure for Cloud Computing. Among the different challenges we need to address to favor the adoption of such a model, the development of a system in charge of turning such a complex and diverse network of resources into a collection of abstracted computing facilities that are convenient to administrate and use is critical.
In this report, we introduce the premises of such a system. The novelty of our work is that instead of developing a system from scratch, we revised the OpenStack solution in order to operate such an infrastructure in a distributed manner leveraging P2P mechanisms. More precisely, we describe how we revised the Nova service by leveraging a distributed key/value store instead of the centralized SQL backend. We present experiments that validated the correct behavior of our prototype, while having promising performance using several clusters composed of servers of the Grid'5000 testbed. We believe that such a strategy is promising and paves the way to a first large-scale and WAN-wide IaaS manager.

Key-words: Fog, Edge Computing, Peer To Peer, Self-*, Sustainability, Efficiency, OpenStack, Future Internet.

* Inria, France, Email: FirstName.LastName@inria.fr
† Mines Nantes/LINA (UMR 6241), France.

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Révisiter les mécanismes internes du système OpenStack en vue d'opérer des infrastructures de type nuage massivement distribuées

Résumé :

La tendance actuelle pour supporter la demande croissante d'informatique utilitaire consiste à construire des centres de données de plus en plus grands, dans un nombre limité de lieux stratégiques. Cette approche permet sans aucun doute de satisfaire la demande actuelle tout en conservant une approche centralisée de la gestion de ces ressources, mais elle reste loin de pouvoir fournir des infrastructures répondant aux contraintes actuelles et futures en termes d'efficacité, de juridiction ou encore de durabilité. L'objectif de l'initiative DISCOVERY[3] est de concevoir le *LUC OS*, un système de gestion distribuée des ressources qui permettra de tirer parti de n'importe quel nœud réseau constituant la dorsale d'Internet afin de fournir une nouvelle génération d'informatique utilitaire, plus apte à prendre en compte la dispersion géographique des utilisateurs et leur demande toujours croissante.

Après avoir rappelé les objectifs de l'initiative DISCOVERY et expliqué pourquoi les approches type fédération ne sont pas adaptées pour opérer une infrastructure d'informatique utilitaire intégrée au réseau, nous présentons les prémisses de notre système. Nous expliquerons notamment pourquoi et comment nous avons choisi de démarrer des travaux visant à revisiter la conception de la solution Openstack. De notre point de vue, choisir d'appuyer nos travaux sur cette solution est une stratégie judicieuse à la vue de la complexité des systèmes de gestion des plateformes IaaS et de la vélocité des solutions open-source.

Mots-clés : Calcul utilitaire basé sur la localité, systèmes pair-à-pair, self-*, durabilité,OpenStack Internet du futur

[3] http://beyondtheclouds.github.io

1 Introduction

To satisfy the escalating demand for Cloud Computing (CC) resources while realizing an economy of scale, the production of computing resources is concentrated in mega data centers (DCs) of ever-increasing size, where the number of physical resources that one DC can host is limited by the capacity of its energy supply and its cooling system. To meet these critical needs in terms of energy supply and cooling, the current trend is toward building DCs in regions with abundant and affordable electricity supplies or taking advantage of free cooling techniques available in regions close to the polar circle [13].

However, concentrating Mega-DCs in only few attractive places implies different issues. First, a disaster[1] in these areas would be dramatic for IT services the DCs host as the connectivity to CC resources would not be guaranteed. Second, in addition to jurisdiction concerns, hosting computing resources in a few locations leads to useless network overheads to reach each DC. Such overheads can prevent the adoption of Cloud Computing by several kind of applications such as mobile computing or Big Data ones.

The concept of micro/nano DCs at the edge of the backbone [14] is a promising solution for the aforementioned concerns. However, operating multiple small DCs breaks somehow the idea of mutualization in terms of physical resources and administration simplicity, making this approach questionable. One way to enhance mutualization is to leverage existing network centers, starting from the core nodes of the network backbone to the different network access points (*a.k.a.*. PoPs – Points of Presence) in charge of interconnecting public and private institutions. By hosting micro/nano DCs in PoPs, it becomes possible to mutualize resources that are mandatory to operate network/data centers while delivering widely distributed CC platforms better suited to cope with disasters and to match the geographical dispersal of users. A preliminary study has established the fundamentals of such an *in-network distributed cloud* referred by the authors as the *Locality-Based Utility Computing* (LUC) concept [3]. However, the question of how operating such an infrastructure still remains. Indeed, at this level of distribution, latency and fault tolerance become primary concerns, and collaboration between components that are hosted on different location must be organized wisely.

In this report, we propose to discuss some key-elements that motivate our choices to design and implement the *LUC Operating System* (LUC-OS), a system in charge of turning a LUC infrastructure into a collection of abstracted computing facilities that are as convenient to administrate and that can be used in the same way as existing Infrastructure-as-a-Service (IaaS) managers [9, 22, 23]. We explain, in particular, why federated approaches [4] are not satisfactory enough to operate a LUC infrastructure and why designing a fully distributed system makes sense. Moreover, we describe the fundamental capabilities the LUC OS should deliver. Because they are similar to those provided by existing IaaS managers and because technically speaking it would be a non-sense to develop the system from scratch, we chose to instanciate the LUC OS concept on top of the OpenStack solution [23].

The main contribution is a proof of concept of the *Nova* service (the Open-

[1] On March 2014, a large crack has been found in the Wanapum Dame leading to emmergency procedures. This hydrolic plan supports the utility power supply to major data centers in central Washington.

Stack compute element) that has been distributed on top of a decentralized key/value store (KVS). By such a mean, it becomes possible to efficiently operate several geographical sites by a single OpenStack system. The correct functioning of this proof of concept has been validated via several experiments performed on top of Grid'5000 [1]. In addition to tackling both the scalability and distribution issues of the SQL database, our KVS proposal leads to promising performance. More than 80% of the API requests are performed faster than with the SQL backend without doing any modification in the *Nova* code.

The remaining of the report is as follows. Section 2 explains our design choices. Section 3 describes OpenStack and how we revised it. The validation of our prototype focusing on the Nova service is presented in Section 4. In Section 5, we present the ongoing works towards a complete LUC OS leveraging the OpenStack ecosystem. Finally Section 7 concludes and discusses future research and development actions.

2 Design Considerations

The massively distributed cloud we target is an infrastructure that is composed of up to hundreds of micro DCs, which are themselves composed of up to tens of servers. While brokering and federated approaches are generally the solutions that have been investigated to operate such infrastructures, we explain in this section why revising OpenStack with P2P mechanisms is an interesting opportunity.

2.1 From Centralized to Distributed

Federations of clouds are the first approaches that are considered when it comes to operate and use distinct clouds. Each micro DC hosts and supervises its own CC infrastructure and a brokering service is in charge of provisioning resources by picking them on each cloud. While federated approaches with a simple centralized broker can be acceptable for basic use cases, advanced brokering services become mandatory to meet requirements of production environments (monitoring, scheduling, automated provisioning, SLAs enforcements ...). In addition to dealing with scalability and single point of failure (SPOF) issues, brokering services become more and more complex to finally integrate most of the mechanisms that are already implemented by IaaS managers [5, 16]. Consequently, the development of a brokering solution is as difficult as the development of an IaaS manager but with the complexity of relying only on the least common denominator APIs. While few standards such as OCCI [20] start to be adopted, they do not allow developers to manipulate low-level capabilities of each system, which is generally mandatory to finely administrate resources. In other words, building mechanisms on top of existing ones, as it is the case of federated systems, prevents them from going beyond the provided APIs (or require intrusive mechanisms that must be adapted to the different systems). The second way to operate a distributed cloud infrastructure is to design and build a dedicated system, *i.e.*, an *Operating System*, which will define and leverage its own software interface, thus extending capacities of traditional Clouds with its API and a set of dedicated tools. Designing a specific system offers an opportunity to go beyond classical federations of Clouds by addressing all crosscutting concerns

of a software stack as complex as an IaaS manager.

The following question is to analyze whether collaborations between mechanisms of the system should be structured either in hierarchical or in flat way via a P2P scheme. During the last years few hierarchical solutions have been proposed in industry [6, 7] and academia [11, 12]. Although they may look easier at first sight than Peer-to-Peer structures, hierarchical approaches require additional maintenance costs and complex operations in case of failure. Moreover, mapping and maintaining a relevant tree architecture on top of a network backbone is not meaningful (static partitioning of resources is usually performed). As a consequence, hierarchical approaches do not look to be satisfactory to operate a massively distributed IaaS infrastructure such as the one we target. On the other side, P2P file sharing systems are a good example of software that works well at large scale in a context where Computing/Storage resources are geographically spread. While P2P/decentralized mechanisms have been underused for building operating system mechanisms, they have showed the potential handle the intrinsic distribution of LUC infrastructures as well as the scalability required to manage them [10].

To summarize, we advocate the development of a dedicated system, *i.e.*, the *LUC Operating system* that will interact with low level mechanisms on each physical server and leverage advanced P2P mechanisms. In the following section, we describe the expected LUC OS capabilities.

2.2 Cloud Capabilities

The LUC OS should deliver a set of high level mechanisms whose assembly results in a system capable of operating an IaaS infrastructure.

Recent studies have showed that state of the art IaaS managers [24] were constructed over the same concepts and that a reference architecture for IaaS managers can be defined [21].

This architecture covers primary services that are needed for building the LUC OS:

- The **virtual machines manager** is in charge of managing VMs' cycle of life (configuration, scheduling, deployment, suspend/resume and shut down).
- The **Image manager** is in charge of VM' template files (*a.k.a.* VM images).
- The **Network manager** provides connectivity to the infrastructure: virtual networks for VMs and external access for users.
- The **Storage manager** provides persistent storage facilities to VMs.
- The **Administrative tools** provide user interfaces to operate and use the infrastructure.
- Finally, the **Information manager** monitors data of the infrastructure for the auditing/accounting.

Thus the challenge is to guarantee for each of the aforementioned services, its decentralized functionning in a fully distributed way. However, as designing

and developing the LUC OS from scratch would be an herculean work, we propose to minimize both design and implementation efforts by reusing as much as possible succesful mechanisms, and more concretly by investigating whether a revised version of the OpenStack [23] could fulfill requirements to operate a LUC infrastructure. In other words, we propose to determine which parts of OpenStack can be directly used and which ones must be revised with P2P approaches. This strategy enables us to focus the effort on key issues such as the distributed functioning and the organization of efficient collaborations between software components composing our revised version of OpenStack, *a.k.a.* the LUC OS.

3 Revising OpenStack

OpenStack [23] is an open-source project that aims at developing a complete CC management system. Its architecture is comparable with the reference architecture previously described (see Figure 1). Two kinds of nodes compose an OpenStack infrastructure: compute and controller nodes. The former are dedicated to the hosting of VMs while the latter are in charge of executing the OpenStack services.

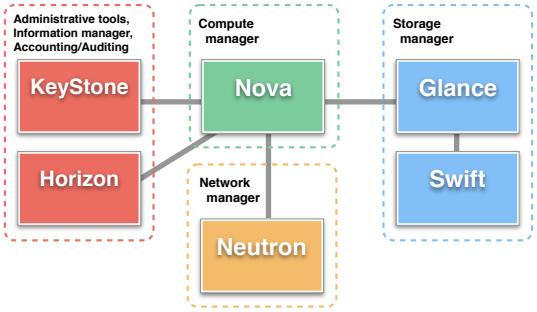


Figure 1: Core-Services of OpenStack.

The OpenStack services are organized following the *Shared Nothing* principle. Each instance of a service (*i.e.*, service worker) is exposed through an API accessible through a *Remote Procedure Call* (RPC) system implemented on top of a messaging queue or via web services (REST). This enables a weak coupling between services. During their life-cycle, services create and manipulate logical objects that are persisted in shared databases, thus enabling service workers to easily collaborate while keeping the compatibility with the *Shared Nothing* principle.

However, even if this organisation of services respects the *Shared Nothing principle*, the message bus and the fact that objects are persisted in shared databases limit the scalabilty of the system, as stated in its documentation:

> *OpenStack services support massive horizontal scale. Be aware that this is not the case for the entire supporting infrastructure. This is particularly a problem for the database management systems and message queues that OpenStack services use for data storage and remote procedure call communications.*

As a consequence, the process of revising OpenStack towards a more decentralized and an advanced distributed functioning, should be carried out in two ways: the distribution of the messaging queue and the distribution of the shared relational databases.

3.1 Decentralizing the AMPQ Bus

As indicated, services composing OpenStack collaborate mainly through a RPC system built on top of an AMQP bus. The AMQP implementation used by OpenStack is *RabbitMQ*. While this solution is generally articulated around the concept of a centralized master broker, it provides by default a cluster mode that can be configured to work in a highly available mode. Several machines, each hosting a RabbitMQ instance, work together in an Active/Active functioning where each queue is mirrored on all nodes. While it has the advantage of being simple, it has the drawback of being very sensible to network latency, and thus it is not relevant for multi-site configurations at large scale. This limitation is well known from the distributed messaging queue community. Few workarounds have been proposed for solutions such as RabbitMQ and more recently, P2P-like systems such as ActiveMQ [25] or ZeroMQ [15] have been released. Such broker-less solutions satisfy the LUC requirements in terms of scalability and because an action already showed that it is feasible to replace RabbitMq with ZeroMQ[2], we chose to focus our efforts on the DB challenge.

3.2 Decentralizing the Databases

From today's perspective, most of the OpenStack deployments are involving few compute nodes and do not require more than a single database (DB) node in terms of scalability. Each instance of every service composing the OpenStack infrastructure can collaborate with remote instances by sharing logical objects (inner-states) that are usually persisted in a single DB node. The use of a second DB is generally considered to satisfy the high availability constraint that is mandatory in production infrastructures. In such a context, the OpenStack community recommends the use of at least an *active/passive* replication strategy. A second DB acts as a failover of the master instance. However, when the infrastructure becomes larger or includes distinct locations, it becomes mandatory to distribute the existing relational DBs over several servers.Two approaches are proposed by the OpenStack community. The first one consists in partitioning the infrastructure into group called *cells* configured as a tree. The top-cell is generally composed of one or two nodes (*i.e.*, the top-cell does not include compute nodes) and is in charge of redistributing requests to the child cells. Each child cell can be seen as an independent OpenStack deployment with its own DB server and message queue broker. In addition to facing hierarchical approach issues we previously discussed (see Section 2.1), we highlight that additional mechanisms are mandatory on top of the vanilla OpenStack code in order to make collaboration between cells possible. Consequently, this approach looks closer to a brokering solution than a native collaboration of a IaaS management system as we target. The second approach consists in federating several OpenStack deployments throughout an *active/active* replication mechanism [18]

[2] https://wiki.openstack.org/wiki/ZeroMQ (valid on Dec 2015)

provided by the *Galera* solution. By such a mean, when an instance of a service processes a request and performs some actions on one site, changes in the inner-states stored in the DB are also propagated to all the other DBs of the infrastructure. From a certain point of view, it gives the illusion that there is only one unique DB shared by all OpenStack deployments. Although the described technique has been used in production systems, most of them only involve a limited number of geographical sites. Indeed, active replication mechanisms imply important overheads that limit the size of infrastructures. To sum up neither the hierarchical approach nor the active replication solution are suited to deal with a massively distributed infrastructure as the one we target.

While not yet explored for the main OpenStack components, NoSQL databases seem to have more suitable properties for highly distributed context, providing a better scalability and built-in replication mechanisms. Distributed Hash Tables (DHTs) and more recently key/value stores (KVSes) built on top of the DHT concept such as *Dynamo* [10] have demonstrated their efficiency in terms of scalability and fault tolerance properties. The challenge consists in analyzing how OpenStack internals can be revised to be able to manipulate inner-states through a KVS instead of a classical SQL system. In the next section we present how we performed such a change for the Nova component.

3.3 From MySQL to REDIS, The Nova POC

As illustrated in Figure 2, the architecture of Nova has been organized in a way which ensures that each of its sub-services does not directly manipulate the DB. Instead it calls API functions proposed by a service called "nova-conductor". This service forwards API calls to the **"db.api"** component that proposes one implementation per database type. Currently, there is only one implementation that works on relational DBs. This implementation relies on the *SQLAlchemy* object-relational-mapping (ORM) that enables the manipulation of a relational database via object oriented code.

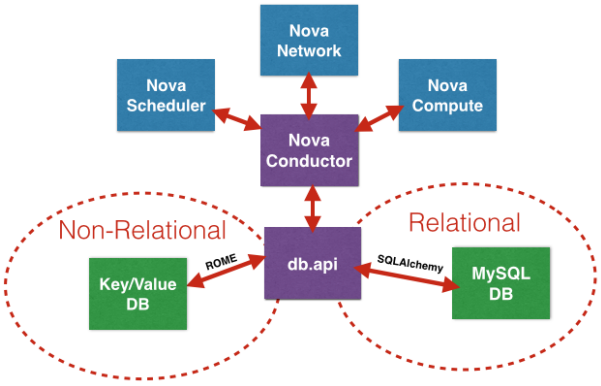


Figure 2: Nova - Software Architecture and DB dependencies.

Thanks to the usage of this ORM, the given implementation is not tightly-coupled with the relational model. Such a feature enabled us to develop *ROME*, a library that exposes the same functions as *SQLAlchemy* and performs the same actions but on non-relational DBs. As *ROME* and *SQLAlchemy* are very

similar, we have been able to copy the existing implementation of **"db.api"**, and to replace every use of *SQLAlchemy* by a call to *ROME*, enabling Nova's services to work with a KVS, while limiting the number of changes in the original source code.

Thanks to this modification, it is possible to deploy an OpenStack infrastructure that works with a natively distributed database, which gives a first glimpse of large multi-site deployments. Figure 3 depicts such a deployment: each geographical site hosts at least one controller node and at least a part of the NoSQL DB, *a.k.a.* the KVS Controller nodes collaborate in a flat way thanks to the shared KVS and the shared AMQP bus. The number of controller nodes on each site can vary according to the expected demand created by end-users. Finally, a controller node can be deployed either on a dedicated node or be mutalized with a compute node as illustrated for Site 3. We higlight that any controller node can provision VMs by orchestrating services on the whole infrastructure and not only on the site where it is deployed. Concerning the choice of the NoSQL database, we chose to use REDIS in our prototype because of its deployment/usage simplicity. However we are aware that databases that focus on high-availability and partition tolerance criteria, such as Cassandra [19], could be a good fit as they have already been deployed on production environments.

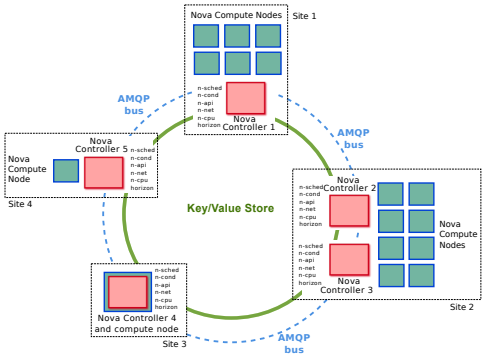
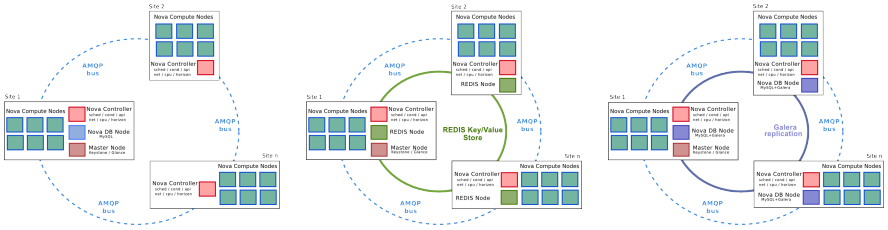


Figure 3: Nova controllers (in light-red) are connected through a shared key/-value backend and the AMQP bus. Each controller runs all nova services and can provision VMs on any compute node (in light blue).

4 Experimental Validation

The validation of our proof-of-concept has been done via three sets of experiments. The first one aimed at measuring the impact of the use of the REDIS NoSQL solution instead of the MySQL system in a single site deployment. The second set focused on multi-site scenarios by comparing the impact of the latency on our distributed Nova service with respect to an active/active Galera deployment. Finally, the last experiment showed that higher level OpenStack mechanisms are not impacted by the use of the REDIS KVS.

All Experiments have been performed on Grid'5000 [1], a large-scale and versatile experimental testbed that enables researchers to get an access to a large amount of computing resources with a very fine control of the experimental con-



(a) Single centralized DB (b) REDIS Key/Value Store (c) Galera

Figure 4: Investigated deployments on top of G5K and role of each server node.

ditions. We deployed and configured each node involved in the experiment with a customized software stack (Ubuntu 14.04, a modified version of OpenStack "Devstack", and REDIS v3) using Python scripts and the Execo toolbox [17]. We underline that we used the legacy network mechanisms integrated in Nova (*i.e.*,, without Neutron) and deployed other components that were mandatory to perform our experiment (in particular Keystone and Glance) on a dedicated node belonging to the first site (entitled master node on Figure 4).

4.1 Impact of REDIS w.r.t MySQL

4.1.1 Time penalties

Changes made over Nova's source code to support a NoSQL database as REDIS is likely to affect its reactivity. The first reason is that a KVS does not provide a support of operations like joining, and thus the code we developed to provide such operations, creates a computation overhead. The second reason is related to networking. Unlike a single MySQL node, in a REDIS system data is spread over several nodes. Thus, a request can lead to several network exchanges. Finally, REDIS provides a replication strategy to deal with fault tolerant aspects, leading also to possible overheads.

Table 1: Average response time to API requests for a mono-site deployment (in ms).

Backend configuration	REDIS	MySQL
1 node	83	37
4 nodes	82	-
4 nodes + repl	91	-

Table 2: Time used to create 500 VMs on a single cluster configuration (in sec.)

Backend configuration	REDIS	MySQL
1 node	322	298
4 nodes	327	-
4 nodes + repl	413	-

Table 1 compares average response times used to satisfy API requests made during the creation of 500 VMs on an infrastructure deployed over one clus-

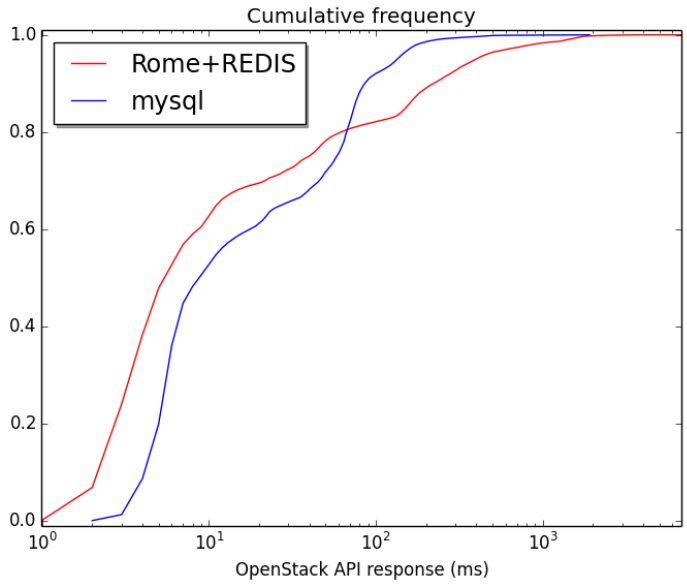
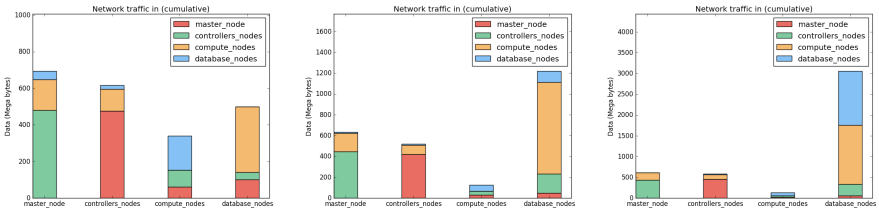


Figure 5: Statistical distribution of Nova API response time (in ms.).



(a) Single centralized DB

(b) 4 nodes REDIS cluster without replication

(c) 4 nodes REDIS cluster with replication (1 replica)

Figure 6: Amount of data exchanged per type of nodes, varying the DB configuration (MySQL or REDIS).

ter (containing 1 controller node and 6 compute nodes), using either REDIS or the MySQL backend under the three aforementioned scenarios. While the distribution of REDIS between several nodes and the use of the replication feature do not significantly increase the response time (first column), the difference between the average API response time of our KVS approach and the vanilla MySQL code may look critical at first sight (124% higher). However, it must be mitigated with Figure 5 and Table 2. Figure 5 depicts the statistical distribution of the response time of each API call that has been made during the creation of the 500 VMs. It is noticeable that for a large part of them (around 80%), our Rome/REDIS solution delivers better performance than the SQLAlchemy/MySQL backend. On the other side, the 10% of the slowest API calls are above 222 ms with our proposal while they are are around 86 ms when using MySQL. Such a difference explains the averages recorded in Table 1 and

we need to conduct deeper investigations to identify the kind of requests and how they can be handled in a more effective way. Overall, we can see that even with these slow-requests, the completion time for the creation of 500 VMs is competitive as illustrated by Table 2. In other words, some API functions have a more significant impact than others on the VM creation time.

4.1.2 Networking penalties

As we target the deployment of an OpenStack infrastructure over a large number of geographical sites linked together through the Internet backbone, the quantity of data exchanged is an important criterion for the evalution of our solution. In particular, as data is stored in the KVS with an object structure, it requires a serialization/deserialization phase when objects are stored/queried. To enable this serialization, the addition of some metadata is required, which leads to a larger data footprint and thus a larger amount of data exchanged between database nodes and OpenStack nodes.

To determine wether the level of network-overhead is acceptable or not, networking data has been collected during the previous experiments. Table 3 compares the total amount of data exchanged over network depending of the database configuration that has been used. As MySQL does not store serialized objects, *i.e.*, objects are serialized at the client-side by the ORM, thus only raw data is exchanged over the network, we consider the single node MySQL as the optimal solution, which has been measured at 1794MB. Our solution deployed over a single REDIS node consumes 2190MB, which means that the networking overhead related to the combination of ROME and REDIS is estimated to be around 22%. Doing the same experiment with a 4 nodes REDIS cluster without data replication leads to a 33% networking overhead compared to a single MySQL node. Finally, when the data replication is enabled with one replica, the amount of data exchanged over the network is 76% higher than without replication, which is intuitive.

However, the infrastructures that have been deployed contain different kind of nodes (controllers, compute nodes, database nodes, ...) and the aforementioned data takes everything into account and thus it does not enable us to get a precise view of the origin of the networking overhead.

Table 3: Amount of data exchanged over the network (in MBytes)

Backend configuration	REDIS	MySQL
1 node	2190	1794
4 nodes	2382	-
4 nodes + repl (1 replica)	4186	-

By using the **iftop** [3] tool, it has been possible to gather information about the origin and destination of TCP/IP messages exchanged during the creation of VMs, and thus to get a precise idea of the origin of any networking overhead. Figure 6 depicts the data exchanges in function of the origin (horizontal axis) and the destination (vertical bars), varying the database configuration. It is noticeable that there is no significant difference in terms of exchange pattern

[3] http://www.ex-parrot.com/pdw/iftop/

for OpenStack nodes, regardless the kind of nodes and regardless the DB configuration that has been used (either MySQL or REDIS). We can notice slightly different patterns for DB Nodes: a comparison of Figure 6(a) and 6(b) confirms that the networking overhead depicted in Table 3 comes from the DB nodes. Finally, Figure 6(c) confirms that most of the overhead observed when enabling data replication is also caused by additional data exchanged between database nodes.

4.2 Multi-site Scenarios

The second experiment we performed consisted in evaluating a single OpenStack deployment over several locations. Our goal was to compare the behaviour of a single MySQL OpenStack with the advised Galera solution and our Rome+REDIS proposal. Figure 4 depicts how the nodes have been configured for each scenario. While the deployment of a single MySQL node is a non sense in a production infrastructure as discussed before, evaluating such a scenario enabled us to to get an indication regarding the maximum performance we can expect. Indeed in such a scenario, the DB is deployed on a single server located in one of the locations, without any synchronization mechanism and consequently no overhead related to communications with remote DB nodes on the contrary to a clustered Redis or an infrastructure composed of several MySQL DBs that are synchronized with the Galera mechanism. Moreover, conducting such an experiment at large scale enabled us to see the limit of such a centralized approach.

Regarding the experimental methodology, all executions have been conducted on servers of the same site (Rennes) in order to ensure reproducibility: *distinct locations* (*i.e.*, clusters) have been emulated by adding latency between group of servers thanks to the `TC` unix tool. Each *cluster* was containing 1 controller node, 6 compute nodes, and one DB node when needed. Scenarios including 2, 4, 6, and 8 clusters have been evaluated, leading to infrastructures composed of up to 8 controllers and 48 compute nodes overall. The latency between each cluster has been set to 10 ms and then 50 ms. Finally, in order to evaluate the capability of such infrastructures to distribute the workload on several controllers, and to detect concurrency problems inherent in using a non relational DB backend, the creation of the 500 VMs has been fairly distributed among the available controllers in parallel.

Table 4: Time to create 500 VMs with a 10ms inter-site latency (in sec.).

Nb of locations	REDIS	MySQL	Galera
2 clusters	271	209	2199
4 clusters	263	139	2011
6 clusters	229	123	1811
8 clusters	223	422	1988

Table 4 and Table 5 present the time to create the 500 VMs. As expected, increasing the number of clusters leads to a decrease of the completion time. This is explained by the fact that a larger number of clusters means a larger number of controllers and compute nodes to handle the workload.

The results measured for a 10ms latency, show that our approach takes a

Table 5: Time to create 500 VMs with a 50ms inter-site latency (in sec.).

Nb of locations	REDIS	MySQL	Galera
2 clusters	723	268	*
4 clusters	427	203	*
6 clusters	341	184	-
8 clusters	302	759	-

rather constant time to create 500 VMs, which stabilizes around 220 seconds. While a single MySQL node has better results until 6 clusters, one can see the limitations of a single server with 8 clusters. In such a case, the single MySQL performs 89% slower than our approach, while the advised Galera solution is 891% slower than our approach.

With a 50ms inter-cluster latency, the difference between REDIS and MySQL is accentuated in the 8 clusters configuration, as MySQL is 151% slower than our REDIS approach.

Regarding Galera, it is noteworthy that important issues related to concurrent modifications of the databases appear with a 50 ms latency, preventing many of the 500 VMs to be created (*i.e.*, several bugs occur leading Nova to consider many VMs as crashed). Such pathological behaviours are due to both the important latency between clusters and the burst mode we used to create the 500 VMs (for information, we succeeded to create 500 VMs but in a sequential manner for 2 and 4 clusters).

To summarize, in addition to tackling the distribution issue, the couple Rome+REDIS enables OpenStack to be scalable: the more controllers are taking part to the deployment, the better the performance is.

4.3 Compatibility with Advanced Features

The third experiment aimed at validating the correct behaviour of existing OpenStack mechanisms while using our Rome+REDIS solution. Indeed, in order to minimize the intrusion in the OpenStack source code, modifications have been limited to the **nova.db.api** component. This component can be considered as the part of Nova that has the most direct interaction with the DB. Limiting the modification to the source code of this component should enable us to preserve compatibility with existing mechanisms at higher level of the stack. To empirically validate such an assumption, we conducted experiments involving multi-site and the usage of host-aggregate/availability-zone mechanism (one advanced mechanism of OpenStack that enables the segregation of the infrastructure). As with aforementioned experiments, our scenario involved the creation of 500 VMs in parallel on a multi-sites OpenStack infrastructure deployed on top of Rome+REDIS with data replication activated (the replica factor was set to 1). Two sets of experiments were conducted: a set where each node of a same geographical site was member of a same host aggregate (that is the same availability zone) and a second set of experiments involving flat multi-site OpenStack (*i.e.*, without defining any availability zone).

Experimental results show that the host-aggregate/availability-zone mechanism behaves correctly on top of our proposal. While VMs are correctly balanced according to the locations where they have been started, the flat multi-site de-

ployment led to a non uniform distribution with respectively 26%, 20%, 22%, 32% of the created VMs for a 4 clusters experiments.

5 On-going Work

While the Nova revision we presented in the previous sections is a promising proof-of-concept toward widely distributed OpenStack infrastructures, there are remaining challenges that need to be tackle.

In this section, we present two on-going actions that aim at deepening the relevance of our approach. First, we show that existing seggregation mechanisms provided by OpenStack are not satisfactory when it comes to reducing inter-site communications. In response, two actions could be made: on one hand introducing networking locality in the shared databases and the shared messaging, on the other hand distributing remaining services of OpenStack.

Second, we discuss the preliminary study we made on the Glance image service to investigate whether it is possible to apply similar modifications to the ones we performed on Nova. Such a validation is critical as Glance is a key element for operating a production-ready IaaS infrastructure.

These two actions clearly demonstrate that our approach is promising enough to favor the adoption of the distributed cloud model supervised by a single system.

5.1 Locality Challenges / μcro DCs Segregation

Deploying a massively multi-site Cloud Computing infrastructure operated by OpenStack is challenging as communication between nodes of different geographical clusters can be subject to an important network latency, which can be a source of disturbances for OpenStack. Experimental results presented in the Table 5 of Section 4.2 clearly showed that an OpenStack distributed on top of our Rome+REDIS solution can already operate over an ISP network with a high inter-site latency (50 ms). While this result is positive and can indicate that such a configuration is appropriated for operating a distributed CC infrastructure involving tens of geographical site, it is important to understand the nature of network traffic. Table 6 shows the total traffic *vs.* the traffic between the remote sites using a 4 sites OpenStack leveraging our Rome+REDIS proposal and with the host-aggregate feature. The first line clearly shows that even with the host-aggregate feature enabled, there is a dramatic amount of communications (87.7%) made between nodes located in distinct geographical sites.

Table 6: Quantity of data exchanged over network (in MBytes)

	Total	Inter-site	Proportion
4 clusters	5326	4672	87.7%

A quarter of these inter-site communications are caused by the isolation of Nova from other OpenStack services (*i.e.*, Keystone and Glance) which were deployed on a dedicated master node in our experiments. Indeed, operations like serving VM images were naturally a source of artificial inter-site communications. This situation clearly advocates in favor of massively distributing the

remaining services, as we did with Nova. Finally, as instances of OpenStack services collaborate via a shared messaging bus and via a shared database, unless these two elements will be able to avoid the broadcasting of information by taking advantage of network locality, the level of inter-site communication will remain large. We are investigating two directions. First, we are studying whether the use of a P2P bus such as ZeroMQ[15] can reduce such a network overhead and second whether the service catalog of Keystone can become locality-aware in order to "hide" redundant services that are located remotely.

5.2 Revising Glance: The OpenStack Image Manager

Similarly to the Nova component (see Section 3.3), only the inner states of Glance are stored in a MySQL DB, the VM images are already stored in a fully distributed way (leveraging either SWIFT or Ceph/Rados solution [26]). Therefore, our preliminary study aimed at determining whether it was possible or not to reuse the ROME library to switch between the SQL and NoSQL backends. As depicted by Figure 7, the Glance code from the software engineering point of view is rather close to the Nova one. As a consequence, replacing the MySQL DB by a KVS system did not lead to specific issues. We underline that the replacement of MySQL with REDIS was even more straightforward than for Nova as Glance enables the configuration of specific API for accessing persistent data (`data_api` in the Glance configuration file). We are currently validating that each request is correctly handled by Rome. Preliminary performance experiments are planned for the beginning of 2016.

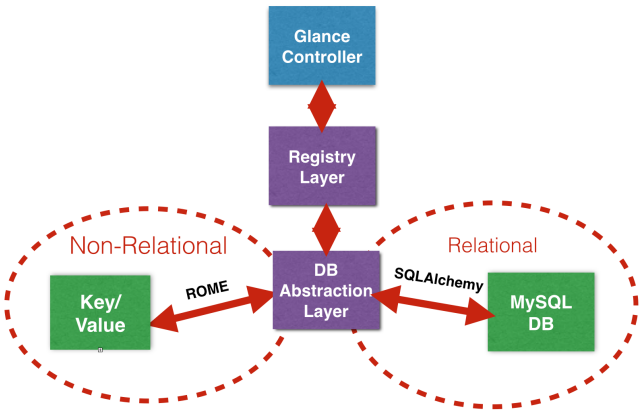


Figure 7: Glance - Software Architecture and DB dependencies.

6 OpenStack for Operating Massively Distributed Clouds

While advantages and opportunities of massively distributed clouds have been emphasized several years ago [8, 14], delivering an OpenStack that can natively be extended to distinct sites will create new opportunities. We present in this section the most important ones and also raise associated challenges (beyond the

technical issues we should resolve to finalize our LUC OS) that our community should tackle to be able to fully use the technical capabilities offered by LUC infrastructures.

From the infrastructure point of view, a positive side-effect of our revised version of OpenStack is that it will natively allow the extension of a private cloud deployment with remote physical resources. Such a scenario is a strong advantage in comparison to the current hybrid offers as it does not break the notion of a single deployment operated by the same tenant. Each time a company will face a peak of activity, it will be possible to provision dedicated servers and attach them to the initial deployment. Those servers can either be provided by dedicated hosting services that have DCs close to the institution or by directly deploying transportable and containerized server rooms close to the private resources. This notion of *WANwide elasticity* can be generalized as it will be possible to deploy such containerized server rooms whenever and wherever they will be mandatory. As examples, we can envision to temporarily deploy IT resources for sport events such as olympic games or for public safety purposes in case of disasters. Network/Telecom operators will also be able to deploy IT resources on their radio base stations they operate in order to deliver Fog/Edge computing solutions. The common thread in these use-cases is the possibility of extending an infrastructure wherever needed with additional resources, the only constraint being to be able to plug the different locations with a backbone that offers enough bandwidth and quality of service to satisfy network requirements. The major advantage is that such an extension is completely transparent for the administrators/users of the IaaS solution because they continue to supervise/use the infrastructure as they are used to. The associated challenge our community should shortly address to deliver such a *Wanwide elasticity* is related to the automatic installation/upgrade of the LUC OS (*i.e.*, our revised OpenStack software) throughout different locations. In such a context, scalability and geo-distribution make the installation/upgrade process more difficult as it would probably require to relocate computations/data between locations in order to be able to update physical servers without impacting the execution of hosted applications. Another issue to investigate is whether it makes sense to extend a deployment between several network operators and how such extensions can be handled. We underline that even for such an extension (the term federation is probably more appropriated in this situation), the two OpenStack systems will join each other to form a single system. Of course security issues should also be addressed, but they are beyond the scope of this paper.

From the software point of view, developers will be able to design new applications but also revise major cloud services in order to deliver more locality aware management of data, computation, and network resources. For instance, it will be possible to deploy on-demand Content Delivery Network solutions according to specific requirements. Cloud storage services could be revised to mitigate the overheads of transferring data from sources to the different locations where there are needed. New strategies can favor for instance a pulling mode instead of a pushing one. Nowadays data is mostly uploaded to the remote clouds without considering whether such data movements are effectively solicited or not. We expect that LUC infrastructures will enable data to stay as close as possible to the source that generates them and be transferred on the other side only when it will be solicited. Such strategies will mitigate the cost of transferring data in all social networks for instance. Similarly, developers

will be able to deliver Hadoop-like strategies where computations are launched close to data sources. Such mechanisms will be shortly mandatory to handle the huge amount of data that Internet of Things will generate. However, delivering the LUC OS will not be sufficient to allow developers to implement such new services. Our community should start as soon as possible to revise and extend current interfaces (*a.k.a.* Application Programming Interfaces). In particular, the new abstractions should allow applications to deal with geo-distribution opportunities and contextual information by using them to specify deployment/reconfigurations constraints or to develop advanced adaptation scenarios in order to satisfy for instance SLAs.

Finally, the last opportunity we envision is related to the use of renewable energies to partially power each PoP of a LUC infrastructure. Similarly to follow-the-moon/follow-the sun approach, the use of several sites spread across a large territory will offer opportunities to optimize the use of distinct energy sources (solar panels, wind turbines). While such an opportunity has been already underlined [2], the main advantage is once again related to the native capability of our revised OpenStack to federate distinct sites, allowing users to use such a widely distributed infrastructure in a transparent way while enabling administrators to balance resources in order to benefit from green energy sources when available (since the system is supervised by a single system we expect that the development of advanced load-balancing strategies throughout the different servers composing the infrastructure would be simplified).

7 Conclusion

Distributing the management of Clouds is a solution to favor the adoption of the distributed cloud model. In this paper, we presented our view of how such distribution can be achieved by presenting the premises of the LUC Operating System. We chose to develop it by leveraging the OpenStack solution. This choice presents two advantages. It minimizes the development efforts and maximizes the chance of being reused by a large community. As a proof-of-concept we presented a revised version of the Nova service that uses a NoSQL backend. We discussed few experiments validating the correct behavior and showed promising performance over 8 clusters.

Our ongoing activities focus on two aspects. First, we expect to finalize the same modifications on the Glance image service soon and start to investigate also whether such a DB replacement can be achieved for Neutron. We highlight that we chose to concentrate our effort on Glance as it is a key element to operate an OpenStack IaaS platform. Indeed, while Neutron is becoming more and more important, the historical network mechanisms integrated in Nova are still available and intensively used. The second activity studies how it can be possible to restrain the visibility of some objects manipulated by the different controllers that have been deployed throughout the LUC infrastructure: our POC manipulates objects that might be used by any instance of a service, no matter where it is deployed. If a user has build an OpenStack project (tenant) that is based on few sites, appart from data-replication, there is no need for storing objects related to this project on external sites. Restraining the storage of such objects according to visibility rules would save network bandwidth and reduce overheads.

Although delivering an efficient distributed version of OpenStack is a challenging task, we believe that addressing it is the key to go beyond classical brokering/federated approaches and to promote a new generation of cloud computing more sustainable and efficient. We are in touch with large groups such as Orange Labs and are currently discussing with the OpenStack foundation to propose the Rome/REDIS Library as an alternative to the SQLAlchemy/MySQL couple. Most of the materials presented in this article such as our prototype are available on the Discovery initiative website.

Acknowledgments

Since July 2015, the Discovery initiative is mainly supported through the Inria Project Labs program and the I/O labs, a joint lab between Inria and Orange Labs. Further information at http://www.inria.fr/en/research/research-fields/inria-project-labs

References

[1] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lebre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec. Adding Virtualization Capabilities to the Grid'5000 Testbed. In I. Ivanov, M. Sinderen, F. Leymann, and T. Shan, editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[2] J. L. Berral, I. n. Goiri, T. D. Nguyen, R. Gavaldá, J. Torres, and R. Bianchini. Building green cloud services at low cost. In *Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems*, ICDCS '14, pages 449–460, Washington, DC, USA, 2014. IEEE Computer Society.

[3] M. Bertier, F. Desprez, G. Fedak, A. Lebre, A.-C. Orgerie, J. Pastor, F. Quesnel, J. Rouzaud-Cornabas, and C. Tedeschi. Beyond the Clouds: How Should Next Generation Utility Computing Infrastructures Be Designed? In Z. Mahmood, editor, *Cloud Computing*, Computer Communications and Networks, pages 325–345. Springer International Publishing, 2014.

[4] R. Buyya, R. Ranjan, and R. N. Calheiros. InterCloud: Utility-oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *10th Int. Conf. on Algorithms and Architectures for Parallel Processing - Vol. Part I*, ICA3PP'10, pages 13–31, 2010.

[5] R. Buyya, R. Ranjan, and R. N. Calheiros. Intercloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *Algorithms and architectures for parallel processing*, pages 13–31. Springer, 2010.

[6] Cascading OpenStack. https://wiki.openstack.org/wiki/OpenStack_cascading_solution.

[7] Scaling solutions for OpenStack. http://docs.openstack.org/openstack-ops/content/scaling.html.

[8] K. Church, A. G. Greenberg, and J. R. Hamilton. On delivering embarrassingly distributed cloud services. In *HotNets, Usenix*, pages 55–60. Citeseer, 2008.

[9] CloudStack, Open Source Cloud Computing. http://cloudstack.apache.org.

[10] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-Value Store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[11] F. Farahnakian, P. Liljeberg, T. Pahikkala, J. Plosila, and H. Tenhunen. Hierarchical VM Management Architecture for Cloud Data Centers. In *6th International Conf. on Cloud Computing Technology and Science (CloudCom)*, pages 306–311, Dec 2014.

[12] E. Feller, L. Rilling, and C. Morin. Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds. In *12th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, pages 482–489, 2012.

[13] J. V. H. Gary Cook. How Dirty is Your Data ? Greenpeace International Report, 2013.

[14] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73, 2008.

[15] P. Hintjens. *ZeroMQ: Messaging for Many Applications.* " O'Reilly Media, Inc.", 2013.

[16] I. Houidi, M. Mechtri, W. Louati, and D. Zeghlache. Cloud Service Delivery Across Multiple Cloud Platforms. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 741–742. IEEE, 2011.

[17] M. Imbert, L. Pouilloux, J. Rouzaud-Cornabas, A. Lèbre, and T. Hirofuchi. Using the EXECO Toolbox to Perform Automatic and Reproducible Cloud Experiments. In *1st Int. Workshop on UsiNg and building ClOud Testbeds (UNICO, collocated with IEEE CloudCom*, Dec. 2013.

[18] B. Kemme and G. Alonso. Database Replication: A Tale of Research Across Communities. *Proc. VLDB Endow.*, 3(1-2):5–12, Sept. 2010.

[19] A. Lakshman and P. Malik. Cassandra: A Decentralized Structured Storage System. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[20] N. Loutas, V. Peristeras, T. Bouras, E. Kamateri, D. Zeginis, and K. Tarabanis. Towards a Reference Architecture for Semantically Interoperable Clouds. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 143–150. IEEE, 2010.

[21] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer*, 45(12):65–72, 2012.

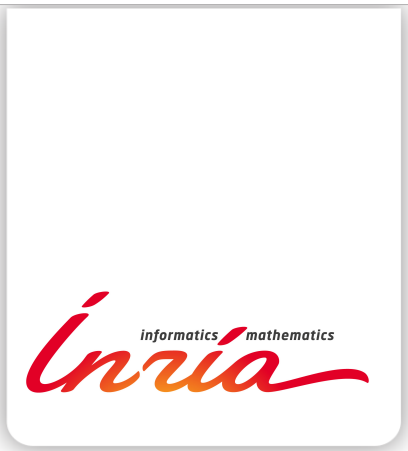
[22] Open Source Data Center Virtualization. http://www.opennebula.org.

[23] The Open Source, Open Standards Cloud. http://www.openstack.org.

[24] J. Peng, X. Zhang, Z. Lei, B. Zhang, W. Zhang, and Q. Li. Comparison of Several Cloud Computing Platforms. In *2nd Int. Symp. on Information Science and Engineering (ISISE)*, pages 23–27, 2009.

[25] B. Snyder, D. Bosnanac, and R. Davies. *ActiveMQ in Action*. Manning, 2011.

[26] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association, 2006.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

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

ISSN 0249-0803