Revising OpenStack to Operate Fog/Edge Computing infrastructures

Adrien Lebre, Jonathan Pastor, Anthony Simonet, Frédéric Desprez

To cite this version:

Adrien Lebre, Jonathan Pastor, Anthony Simonet, Frédéric Desprez. Revising OpenStack to Operate Fog/Edge Computing infrastructures. IC2E 2017 : IEEE International Conference on Cloud Engineering, Apr 2017, Vancouver, Canada. 10.1109/IC2E.2017.35 . hal-01273427

HAL Id: hal-01273427

https://hal.inria.fr/hal-01273427

Submitted on 16 Nov 2016

Revising OpenStack to Operate Fog/Edge Computing infrastructures

Adrien Lebre, Jonathan Pastor, Anthony Simonet
Inria, Mines de Nantes, LINA
Nantes, France
Email: firstname.lastname@inria.fr

Frédéric Desprez
Inria, LIG
Grenoble, France
Email: firstname.lastname@inria.fr

***Abstract*—Academic and industry experts are now advocating for going from large-centralized Cloud Computing infrastructures to smaller ones massively distributed at the edge of the network. Among the obstacles to the adoption of this model is the development of a convenient and powerful IaaS system capable of managing a significant number of remote data-centers in a unified way.**

In this paper, we introduce the premises of such a system by revising the OpenStack software, a leading IaaS manager in the industry. The novelty of our solution is to operate such an Internet-scale IaaS platform in a fully decentralized manner, using P2P mechanisms to achieve high flexibility and avoid single points of failure. More precisely, we describe how we revised the OpenStack Nova service by leveraging a distributed key/value store instead of the centralized SQL backend. We present experiments that validate the correct behavior and gives performance trends of our prototype through an emulation of several data-centers using Grid'5000 testbed. In addition to paving the way to the first large-scale and Internet-wide IaaS manager, we expect this work will attract a community of specialists from both distributed system and network areas to address the Fog/Edge Computing challenges within the OpenStack ecosystem.

I. Introduction

To satisfy the escalating demand for Cloud Computing (CC) resources while realizing economies 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 trend has been 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 a 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 prevent the adoption of the Cloud Computing paradigm by several kind of applications such as mobile computing ones [24].

[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.

The concept of micro/nano DCs deployed at the edge of the backbone has been proposed as a promising solution for the aforementioned concerns [14]. Although it has been proposed as soon as 2008, the model has been debated for a couple of years because it was believed that there would be no way to operate multiple small DCs without increasing initial and exploitation expenditures. A recent study demonstrated that a model leveraging existing network facilities (*a.k.a.* network point of presences) can deliver competitive solutions from the economic viewpoint in comparison to current Amazon offers [25]. Conducting such studies is important as the advent of Internet of Things applications motivates even more "in-network" Cloud Computing platforms [28], *a.k.a.* "fog/edge computing" infrastructures [3], [27].

While the question of whether Fog/Edge computing platforms will be deployed is not being debated anymore, the question of how operating such a widely geo-distributed 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 locations must be organized wisely.

This paper proposes several contributions to make progress on the aformentioned question.

First, we discuss some key elements that motivate the choice of designing a system capable of supervising a set of remote DCs in a unified way. We explain why federated approaches [4] are not the best approaches to operate Fog/Edge infrastructures and why designing a fully distributed system makes sense.

Second, we present and evaluate a proof of concept of a first-class Internet-scale IaaS manager. Because fundamental capabilities of this system are similar to those provided by existing IaaS managers and because technically speaking it would be a non-sense to develop

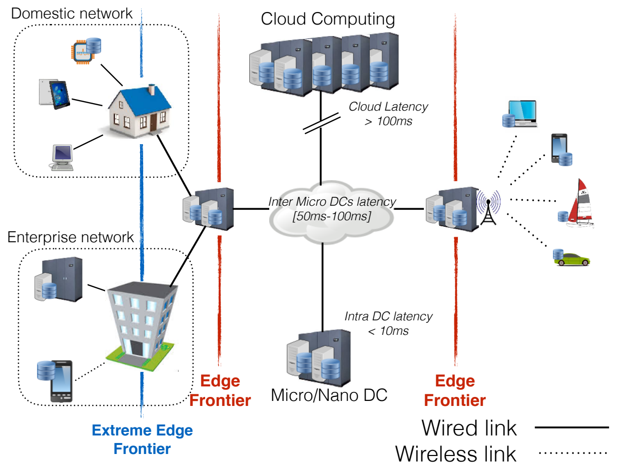


Fig. 1. Fog/Edge Computing Architecture Model

the system from scratch, we chose to build our prototype on top of the OpenStack solution [22]. This will allow us to propose a system that is as convenient to administrate and as easy to use as existing IaaS managers. Specifically, we describe how we revised the *Nova* service (the OpenStack compute element) with a decentralized key/value store in place of the centralized SQL database. This revision enables us to distribute Nova over several geographical sites. 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 prototype 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 last contribution underlines a set of challenges and opportunities that are unique to Fog/Edge computing infrastructures. These challenges go well beyond the scope of this paper and require a large community, with academic and industry specialists from many different fields such as operating systems, networks, scheduling and programming models. With this work, we hope to bring attention to the matter and start building a large community around an OpenStack-based Internet-scale IaaS Manager, like it was once done for Linux and HPC.

The remaining of the paper is as follows. Section II explains our design choices. Section III describes OpenStack and how we revised it. The validation of our prototype focusing on the Nova service is presented in Section IV. In Section V, we present challenges related to the design and development of an Internet-scale IaaS manager. Finally Section VI concludes and discusses future research and development actions.

II. 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 one hundred servers (up to two racks). Figure 1 gives an overview of a Fog/Edge architecture. with several micro/nano DCs and the expected latency between each element. The network links can be either wired or wireless (represented by plain and dashed lines on the figure). Finally, it might be possible to consider additional DCs deployed at the Extreme Edge, within a private institution.

In this section, we discuss design considerations that motivate our implementation choices.

A. *Broker vs Cooperative Systems*

Cloud Federations based on brokering approaches are the first solutions that are considered when it comes to 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 cloud infrastructure that is spread across distinct sites is to design and build a dedicated system, *i.e.*, an Internet-scale IaaS manager. Such a manager 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.

Moreover, a IaaS manager for Fog/Edge platforms 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. For instance on Figure 1, one can consider to extend the resource available in a company (left side on the figure) with resources provided by the first micro DCs present in the network. 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, it can be possible 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. This kind of features cannot be achieved with a brokering approach, unless the brokering middleware is already present and always the cloud entry point.

B. From Centralized to Distributed

Considering the advantage of a Internet-scale IaaS manager with respect to brokering proposals, the next 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 P2P 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). Finally, a hierarchical structure means that there is a global entry point that does not enable to address the latency issue (every cloud request going through the global entry point before being served by one DC). 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 IaaS system mechanisms, they have showed potential for handling the intrinsic distribution of Fog/Edge infrastructures in a scalable manner [9].

C. The Choice of OpenStack

The Internet-scale manager we target 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 [23] 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 each of those systems 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 [22] could fulfill requirements to operate a Fog/Edge 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 of OpenStack.

III. Revising OpenStack

OpenStack [22] is an open-source project that aims to develop a complete Cloud Computing software stack.

Figures 2 presents the general vision of OpenStack with the three expected capabilities of IaaS platforms: Compute Network and Storage. Applications at the top can request through a high level API compute, network and storage resources. OpenStack bricks in the middle communicate through shared services. From the technical point of view, OpenStack is composed of two kinds of nodes: on the first side, the compute/storage/network nodes are dedicated to deliver the XaaS capabilities such as hosting VMs for the compute; on the other side the controller nodes are in charge of executing the OpenStack services.

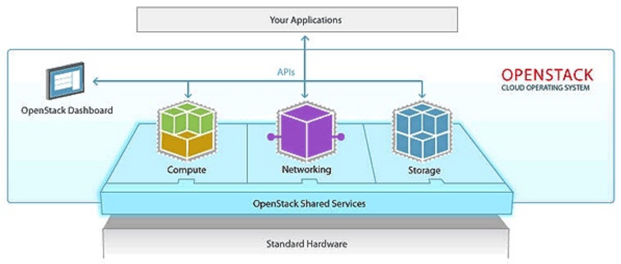


Fig. 2. OpenStack Overview

Figure 3 shows the core-services of OpenStack. This architecture is comparable with the reference one described in the previous section.

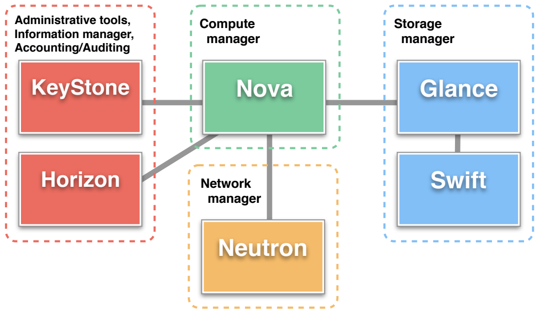


Fig. 3. Core-Services of OpenStack.

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. 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 the OpenStack 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.

To conclude, revising OpenStack to make it fully decentralized should be carried out in two ways: distributing the messaging queue as well as the shared relational databases.

A. Distributing 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 also provides a cluster mode that can be configured to work in a highly available manner. 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 sensitive to network latency, and thus it is not relevant for multi-site configurations. This limitation is well known from the distributed messaging queue community. Few workarounds have been proposed for systems such as RabbitMQ and more recently, P2P-like solutions such as ActiveMQ [26] or ZeroMQ [15] have been released. Thoses broker-less approaches satisfy our requirements in terms of scalability. Considering that there is already an action to use ZeroMQ in place of RabbitMQ in OpenStack[2], we chose to focus our efforts on the DB challenge.

B. Distributing the Databases

As of today, only a few actors such as Rackspace [3] have been dealing with large-scale challenges. In other words, most of the OpenStack deployments only involve a few compute nodes and do not require more than a single database (DB) node. The use of a second DB is generally proposed to meet 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).

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.

[2]https://wiki.openstack.org/wiki/ZeroMQ (valid on Sep 2016)
[3]http://rack.ly/6010B2xpQ (valid on Sept 2016)

The first one consists in partitioning the infrastructure into groups called *cells* configured as a tree. The top-cell is in charge of redistributing requests to the children 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 II-A), we highlight that additional mechanisms are mandatory in order to make collaboration between cells possible (there is no communications nor interactions between children cells). Consequently, this approach is more comparable to a brokering solution than to a native collaboration between IaaS core-services.

The second approach consists in federating several OpenStack deployments throughout an *active/active* replication mechanism of DBs [18] through solutions such as *Galera*: 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 built on top of recent P2P key/value stores seem to have more suitable properties for highly distributed contexts, providing better scalability and built-in replication mechanisms. The challenge consists in analyzing how OpenStack internals can be revised to be able to store service states in a key/value store instead of a classical SQL system. In the next section we present how we revised the Nova component in this direction.

C. The Nova POC: From MySQL to RedisS

Nova is the OpenStack core-service in charge of VM management. Its software architecture is organized in a way which ensures that each of its sub-services does not directly manipulate the DB (see Figure 4). 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.

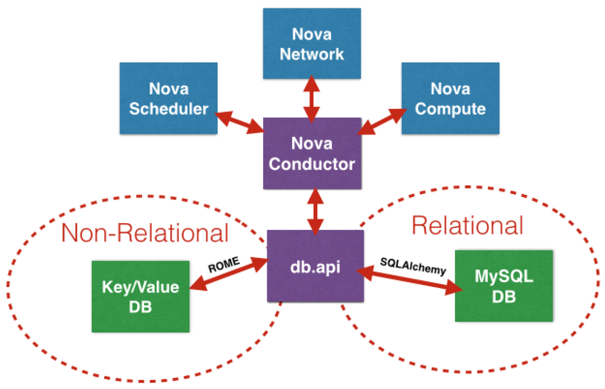


Fig. 4. Nova - Software Architecture and DB dependencies.

Thanks to this software pattern, we developed *ROME*, an ORM library that enables interactions with key/value stores in the same way *SQLAlchemy* interacts with relational DBes. *ROME* has been developed in a non intrusive way by proposing the same API as SQLAlchemy as well as the same internal mechanisms (such as Queries, JOIN operations and Sessions —the SQLAlchemy equivalent of transactions). These implementations have been achieved in order to take into account key/value store specifics: First, we developed a distributed lock system developed on top of *Redis* as well as a two-phase commit approach to support *sessions*, enabling atomic modifications of the key/value store; second we included a secondary index in order to speed up complex queries containing join operations.

The integration of *ROME* within the OpenStack Nova code was restricted to the **"db.api"** file where every call to *SQLAlchemy* has been replaced with a call to *ROME*. Thanks to this modification, it is now possible to operate and use a multi-site OpenStack infrastructure by relying solely on Redis, a P2P key/value store solution. We chose to use Redis in our prototype because of its deployment/usage simplicity.

Figure 5 depicts such a deployment where a key/value store and a global shared AMQP bus enabled all controllers to interact. The number of controller nodes on each site can vary according to the expected demand created by end-users: sites with a large number of compute nodes can have several controllers whereas a controller node can be be mutualized with a compute node as illustrated for Site 3.

We highlight that any controller node can provision VMs by invoking services on the whole infrastructure and not only on the site where it is deployed. In other words, because the states and the communications are managed globally, any service can satisfy any request.

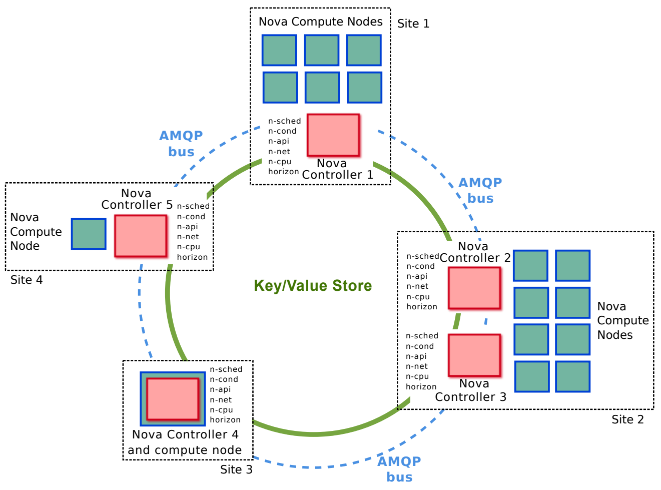


Fig. 5. Nova controllers (in light-red) are connected through a shared key/value backend and the AMQP bus. Each controller runs all nova services and can provision VMs on any compute node (in light blue).

IV. Experimental Validation

The validation of our prototype has been done *via* three sets of experiments. The first one aimed at measuring the impact of the use of Rome/Redis instead of SQLAlchemy/MySQL 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 enabled us to evaluate whether our low level revisions impact higher level OpenStack mechanisms such as the *host-agregates* capability.

All Experiments have been performed on Grid'5000 [1], a large-scale and versatile experimental testbed that enables researchers to get access to a large amount of bare-metal computing resources with very fine control of the experimental conditions. Our experiments use the Parasilo and Paravance clusters, that share a similar configuration[4]: two 4-core Intel Xeon E5-2630v3 CPUs @ 2.4 GHz, 128 GB of RAM and two 10 GB ethernet network interface. We deployed and configured each node involved in the experiment with a customized software stack (Ubuntu 14.04, OpenStack Kilo 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 on the first site (entitled master node on Figure 6).

[4]https://www.grid5000.fr/mediawiki/index.php/Rennes:Hardware

A. *Impact of Redis w.r.t MySQL*

1) Time penalties: Changes made over Nova's source code to support Redis is likely to affect its reactivity. The first reason is that Redis does not support operations like joining, and thus the code we developed to provide such operations implies a computation overhead. The second reason is related to networking: unlike a single MySQL node, data is spread over several nodes in a Redis deployment, leading to several network exchanges for one request. Finally, Redis provides a replication strategy to deal with fault tolerant aspects, also leading to possible overheads.

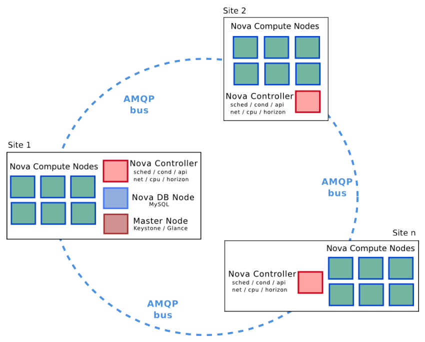
TABLE I
Average response time to API requests for a mono-site deployment (in ms).

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

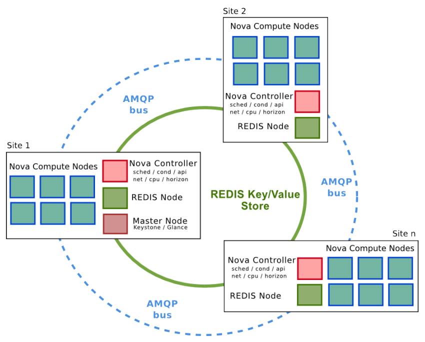
TABLE II
Time used to create 500 VMs on a single cluster configuration (in sec.)

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

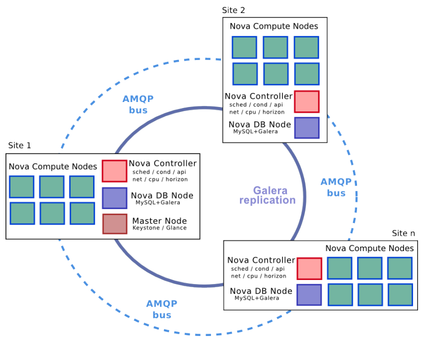
Table I compares average response times used to satisfy API requests made during the creation of 500 VMs on an infrastructure deployed over one cluster (containing 1 controller node and 6 compute nodes), using either Redis or MySQL under three scenarios: i) a single-node MySQL database, ii) a 4-node Redis database with no replication and iii) a 4-node Redis database with replication. 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 Rome/Redis and SQLAlchemy/MySQL may look critical at first sight (124% higher). However, it must be mitigated with Figure 7 and Table II. Figure 7 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%), Rome/Redis delivers better performance than SQLAlchemy/MySQL. 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 I and we need to conduct deeper investigations to identify the kind of requests and how they can be handled in a more effective way.



(a) Single centralized DB



(b) Redis Key/Value Store



(c) Galera

Fig. 6. Investigated deployments on top of G5K and role of each server node.

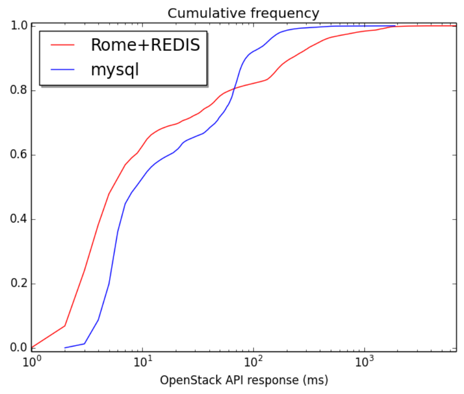


Fig. 7. Statistical distribution of Nova API response time (in ms.).

Overall, we can see that even with these slow-requests, the completion time for the creation of 500 VMs is competitive with respect to SQLAlchemy/MySQL as illustrated by Table II. In other words, some API functions have a more significant impact than others on the VM creation time.

2) Networking penalties: As we target the deployment of an OpenStack infrastructure over a large number of geographical sites linked together through the Internet, the quantity of data exchanged is an important criterion for the evalution of our solution. In particular, as data is stored in the key/value store 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 and OpenStack nodes.

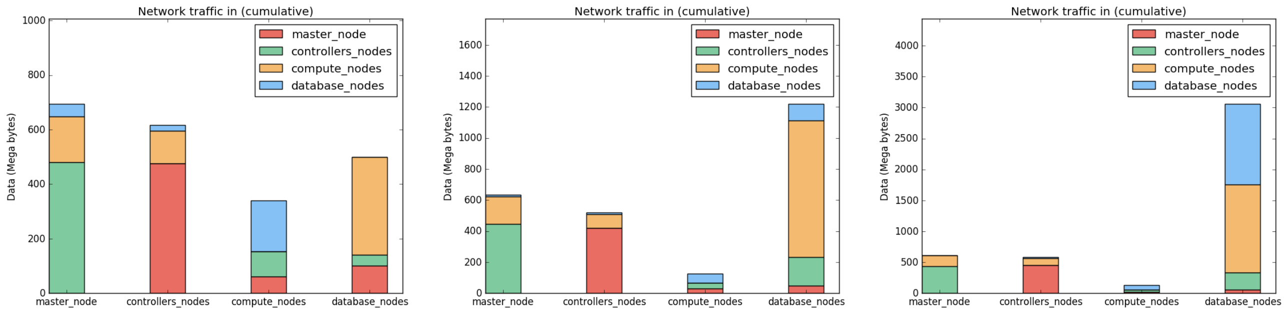
To determine wether the level of network-overhead is acceptable or not, networking data has been collected during the previous experiments. Table III compares the total amount of data exchanged over network depending of the database configuration that has been used. As MySQL does not store serialized objects, *i.e.*, objects are serialized on the client-side by the ORM, only raw data is exchanged over the network. We, thus, consider the single node MySQL as the reference solution, which has been measured at 1794MB. Our solution deployed over a single Redis node exchanges 2190MB, which means that the networking overhead related to the combination of ROME and Redis is estimated around 22%. Doing the same experiment with a 4-node 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.

TABLE III
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	-

Because those values cumulates the network traffic overall, it is not possible to analyze whether the replacement of SQLAlchemy/MySQL by Rome/Redis leads to additional traffic between the controller/compute and the DB nodes. To answer such a question, we used the **iftop** [5] tool, and gathered information about the origin and destination of TCP/IP messages exchanged during the creation of VMs. Figure 8 depicts the data exchanges in function of the origin (horizontal axis) and the destination (vertical bars), with varying database configurations. For example, on the first bar, the green area corresponds to the in network of controller nodes where the source is the master node. Regarding the exchanges between the different kinds of nodes, we observe that there is no significant variation between the three scenarios. The only variation concerns the

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



(a) Single centralized DB (b) 4 nodes Redis cluster without replication (c) 4 nodes Redis cluster with replication (1 replica)

Fig. 8. Amount of data exchanged per type of nodes, varying the DB configuration (Y-axis scales differ).

DB nodes (Figures 8(a) and 8(b)). This confirms that the networking overhead depicted in Table III comes from the DB nodes. Finally, Figure 8(c) confirms that most of the overhead observed when enabling data replication is also caused by additional data exchanged between database nodes. This enables us to conclude that using Rome/Redis in place of SQLAlchemy/MySQL is acceptable from the network viewpoint.

B. Multi-site Scenarios

The second experiment we performed consisted in evaluating a single OpenStack deployment spread over several locations. Our goal was to compare the behavior of a single MySQL OpenStack with the advised Galera solution and our Rome/Redis proposal. Figure 6 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 this scenario enabled us to get an indication of the maximum performance we can expect. Indeed, in this scenario, there is no synchronization mechanism and consequently no overhead related to communications with remote DB nodes. Moreover, conducting such an experiment at large scale enabled us to see the limit of such a centralized approach.

Regarding the experimental methodology, *distinct locations* (*i.e.*, clusters) have been emulated by adding latency between group of servers thanks to the `TC` Unix tool. This enables us to ensure reproducibility between experiments. Each *cluster* contains 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 in total. The latency between each cluster has been set to 10 ms and then 50 ms.

We evaluate the stress induced on the controllers when provisioning and booting 500 VMs at the same time. VM provisioning queries are fairly distributed amongst the controllers.

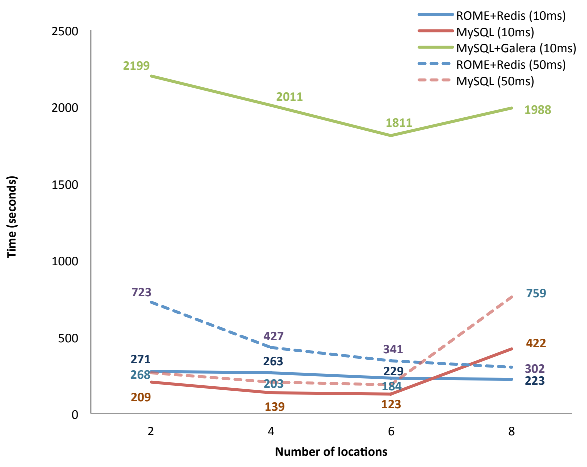


Fig. 9. Time to create 500 VMs with a 10ms and 50ms inter-site latency (due to synchronization issues, MySQL+Galera with 50 ms delay is absent).

Figure 9 presents the observed times. 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 with a 10ms latency show that our approach takes a rather constant time to create 500 VMs, which stabilizes around 220 seconds whatever the number of sites involved in the experiments. Although a single MySQL node scales up to 6 locations, the performance degrades from 8 locations. In this case, the single MySQL performs 89% slower than our approach and 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 synchronization issues appear with a 50 ms latency, preventing the collection of trustable results. Such pathological behaviors are due to 1) the high latency between clusters and 2) the burst mode we used to create the 500 VMs.

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 performance.

C. Compatibility with Advanced Features

The third experiment aimed at validating the correct behavior 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). Similarly to the previous experiments, our scenario involved the creation of 500 VMs in parallel on a multi-sites OpenStack infrastructure deployed on top of Rome/Redis. 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). Being compatible with the host-aggregate feature is important because as previously mentioned in our flat approach, any controller can be involved in any provisioning request.

Experimental results show that the host-aggregate/availability-zone mechanism behaves correctly on top of our proposal: VMs are correctly balanced according to the locations where they have been started while the flat multi-site deployment led to a non uniform distribution with respectively 26%, 20%, 22%, 32% of the created VMs for a 4-location experiment.

V. Challenges and Opportunities of Fog/Edge Platforms Managed in an Unified Way

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 and challenges. Here we take a first glimpse at what these challenges are and we attempt to identify which communities they concern. In the long run, we hope to gather a community with experts from all the domains represented in this section around the development of an ambitious Internet-scale IaaS manager.

A. Locality

Deploying a massively distributed multi-site IaaS infrastructure operated by OpenStack is challenging as communication between nodes of different sites can be subject to important network latencies. Moreover, some objects in OpenStack may be manipulated by any service, and, by extension, should be visible to any controller. On the contrary, some objects may benefit from a restrained visibility: for example, when a user builds an OpenStack project (tenant) based on a few sites, appart from data-replication there is no need for storing the related objects on other sites. Restraining the storage of such objects to certain sites only would enable to save network bandwidth, to settle policies for applications such as privacy and to implement efficient data-replication.

B. Installation and Upgrade Process

OpenStack is a complex and tedious ecosystem composed of tens of services including more than 2 millions of lines of code released on 6-month cycles. In a Fog/Edge context where those services will be distributed amongst all sites –like we did with Nova– it will become impractical for administrators to deploy and upgrade the services on-demand and with the current System Configuration Tools [10]. Hence models and mechanisms for deployment and reconfiguration of services in an autonomous manner will be paramount. Those systems must be capable of automatically installing and upgrading any service without impacting the execution of hosted applications; this process will require some computations and data to be relocated to other locations.

C. Peering Agreement

The *WAN-wide elastic* nature of our revised OpenStack would technically allow administrators from distinct institutions to easily combine two distinct infrastructures; then arise economic/administrative questions, similarly to peering agreements that exists between network providers. Indeed, the two OpenStack system would join to form a single infrastructure where every user on each side would be able to transparently provision resources from both parties. In this case, specific security and quota policies should also be addressed.

D. Cloud Storage

Cloud storage services could be revised to mitigate the overhead of transferring data from their sources to the different locations where there are needed. Programmers could for example want to favor a pulling mode, as opposed to a pushing one. Nowadays, data is mostly uploaded to remote clouds without considering whether the induced data movements will be of any use in the future. A challenge for the storage services (Swift in the case of OpenStack) is to enable data to stay as close to their source as possible, and be transferred on longer distances only when necessary. 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 the Internet of Things will generate.

E. New Cloud APIs

New Application Programming Interfaces should be delivered to allow users to exploit the unique features of Fog/Edge platforms. It should be possible for developers to deploy services according to specific criteria like locality or strong latency requirements. Similarly, the storage capabilities discussed in the previous Subsection will require some sort of declarative language for programmers to express how the data is expected to be used.

F. Renewable Energies

Finally, the last opportunity and challenge we envision is related to the use of renewable energies to power the infrastructure. Similarly to the follow-the-moon/follow-the sun approach, the use of several sites spread across a large territory will offer opportunities the use various energy sources (solar panels, wind turbines, etc.). This opportunity, already underlined in [2], will be enabled by the native capability of our revised OpenStack to federate distinct sites and acomodate entire DCs going periodically offline. Since the whole infrastructure is supervised by a single system, providing users with a transparent infrastructure while implementing advanced load-balancing strategies will be simplified compared to a federated approach.

VI. Conclusion

Proposing a software solution to manage "in network" Cloud Computing infrastructures is a key challenge of our community. In this paper, we presented our view of how such software stack can be achieved by leveraging the OpenStack solution and P2P mechanisms. As a proof-of-concept, we presented a revised version of the Nova service that uses our ROME library to store states of the OpenStack services in Redis, a P2P key/value store system. We discussed several experiments validating the correct behavior of our prototype and showed promising performance when operating 8 locations with a single decentralized OpenStack instance.

Our ongoing activities focus on two aspects. First, we are integrating similar changes in other OpenStack core services such as Glance and Neutron. Second, we study how it can be possible to restrain the visibility of some objects manipulated by the different controllers. We are in particular investigating whether network pooling strategies such as the ones proposed in Cassandra [19] may be more appropriate to the geo-distribution of the infrastructure.

Although delivering an efficient distributed version of an IaaS manager is a challenging task, we believe that addressing it is the key to favor the massive adoption of Fog/Edge computing infrastructures. Choosing OpenStack is a key element as it should enable the scientific community to attract key industrial actors. As examples, Orange Labs, British Telecom as well as major European NRENs already expressed their interest in our action. Moreover, we believe that it is time for the scientific community to get involved and contribute to the OpenStack software in the same way it has been done for the Linux ecosystem. We hope this initial work will have a decisive impact in both the scientific and the open-source community to gather experts around the same goal. We emphasize this work enabled us to exchange with the OpenStack foundation and to take part to different discussions/working groups. For instance, we are conducting performance evaluations in order to identify major bottlenecks in the OpenStack Vanilla code. Finally, we promoted the creation of a new working group dedicated to the massively distributed use-case. Being involved in such actions is important as it allows the community to identify complementary actions and move forward faster. As an example, IBM will soon present an evaluation of the use of Redis for addressing the scheduler scalability issue during the next OpenStack Summit[6].

Acknowledgments

Most of the materials presented in this article such as our prototype are available on the Discovery initiative website. Supported by the Inria Project Lab program, Discovery is an Open-Science Initiative aiming at implementing a fully decentralized IaaS manager: http://beyondtheclouds.github.io.

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.

[6]https://www.openstack.org/summit/barcelona-2016/summit-schedule/events/15424/a-nova-scheduler-for-public-cloud-scale

[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] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[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] 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.

[10] T. Delaet, W. Joosen, and B. Van Brabant. A survey of system configuration tools. In *LISA*, volume 10, pages 1–8, 2010.

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

[23] 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.

[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4):14–23, 2009.

[25] A. Simonet, A. Lebre, and A. C. Orgerie. Deploying distributed cloud infrastructures: Who and at what cost? In *Proceedings of the Intercloud Workshop 2016 (co-located with IEEE International Conference on Cloud Engineering)*, pages 178–183, April 2016.

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

[27] R. Want, B. N. Schilit, and S. Jenson. Enabling the internet of things. *IEEE Computer*, 48(1):28–35, 2015.

[28] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz. The cloud is not enough: Saving iot from the cloud. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, Santa Clara, CA, July 2015. USENIX Association.