The DISCOVERY Initiative - Overcoming Major Limitations of Traditional Server-Centric Clouds by Operating Massively Distributed IaaS Facilities

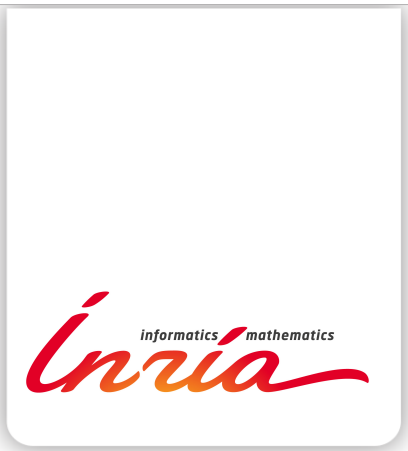
Adrien Lebre, Jonathan Pastor, . The Discovery Consortium

The DISCOVERY Initiative Overcoming Major Limitations of Traditional Server-Centric Clouds by Operating Massively Distributed IaaS Facilities

A. Lebre, J. Pastor, The Discovery Consortium

RESEARCH REPORT
N° 8779
Sep 2015
Project-Team Ascola, Asap, Avalon, Kerdata, Myriads

ISSN 0249-6399 ISRN INRIA/RR--8779--FR+ENG



The DISCOVERY Initiative Overcoming Major Limitations of Traditional Server-Centric Clouds by Operating Massively Distributed IaaS Facilities

A. Lebre*†, J. Pastor*†, The Discovery Consortium*

Project-Team Ascola, Asap, Avalon, Kerdata, Myriads

Research Report n° 8779 — Sep 2015 — 14 pages

* 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

Abstract:
Instead of the current trend consisting of building larger and larger data centers (DCs) in few strategic locations, the DISCOVERY initiative[‡] proposes to leverage any network point of presences (PoP, *i.e.*, a small or medium-sized network center) available through the Internet. The key idea is to demonstrate a widely distributed Cloud platform that can better match the geographical dispersal of users. This involves radical changes in the way resources are managed, but leveraging computing resources around the end-users will enable to deliver a new generation of highly efficient and sustainable Utility Computing (UC) platforms, thus providing a strong alternative to the actual Cloud model based on mega DCs (i.e. DCs composed of tens of thousands resources).
Critical to the emergence of such distributed Cloud platforms is the availability of appropriate operating mechanisms. Although, some of protagonists of Cloud federations would argue that it might be possible to federate a significant number of micro-Clouds hosted on each PoP, we emphasize that federated approaches aim at delivering a brokering service in charge of interacting with several Cloud management systems, each of them being already deployed and operated independently by at least one administrator. In other words, current federated approaches do not target to operate, remotely, a significant amount of UC resources geographically distributed but only to use them. The main objective of DISCOVERY is to design, implement, demonstrate and promote a unified system in charge of turning a complex, extremely large-scale and widely distributed infrastructure into a collection of abstracted computing resources which is efficient, reliable, secure and friendly to operate and use.
After presenting the DISCOVERY vision, we explain the different choices we made, in particular the choice of revising the OpenStack solution leveraging P2P mechanisms. We believe that such a strategy is promising considering the architecture complexity of such systems and the velocity of open-source initiatives.

Key-words: Locality-Based Utility Computing, Peer To Peer, Self-*, Sustainability, Efficiency, Future Internet.

L'initiative DISCOVERY - les infrastructures IaaS massivement distribuées comme solution aux principales limitations des plateformes de Cloud Computing actuelles

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[4] 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, Internet du futur

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

1 Introduction

To satisfy the escalating demand for Cloud Computing (CC) resources while realizing 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 in regions close to the polar circle to leverage free cooling techniques [11].

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 the UC paradigm 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 [12] is a promising solution to address 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 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 and their needs (see Figure 1).

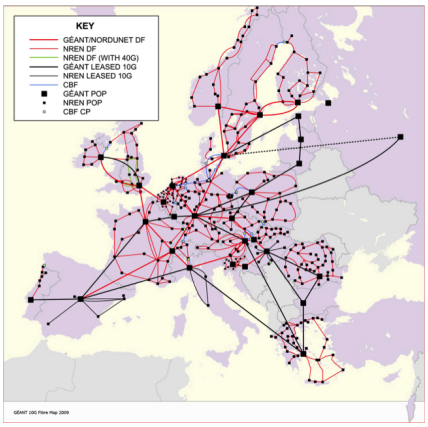


Figure 1: The European GÉANT backbone
GÉANT is the federation of all European Research and Educational Networks. Each black square corresponds to one network point of presence (*a.k.a.*. a PoP) that can host a nano/micro DC.

A preliminary study has established the fundamentals of such an *in-network*

[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.

distributed cloud referred by the consortium as the *Locality-Based Utility Computing* (LUC) concept [2]. However, the question of how operating such an infrastructure is still under investigations. Indeed, at this level of distribution, latency and fault tolerance become primary concerns, and collaboration between servers of differents location must be organized wisely.

In this vision paper, we discuss some key-elements that motivate our choices to design and implement the *LUC Operating System*, a system in charge of turning a LUC infrastructure into a collection of abstracted computing facilities that are as convenient to administrate and use as available Infrastructure-as-a-Service (IaaS) managers [7, 21, 22]. We explain, in particular, why federated approaches [3] are not satisfactory and why designing a fully distributed system that operates all resources makes sense.

As the capabilities of the LUC OS are similar to existing IaaS managers and because it would be a non-sense technically speaking to develop the LUC OS from scratch we chose to revise the OpenStack solution [22], leveraging P2P mechanisms. Our current efforts focus on the validation of a distributed version of the *Nova* service on top of Grid'5000 [1]. Historically, *Nova* relies on a MySQL centralized database, preventing it to natively scale beyond one site. To reach such a goal, we replaced the MySQL component by the REDIS backend, a distributed key/value store. Such a modification enables us to deploy several *Nova* controllers on distinct sites giving the illusion that there was only one global infrastructure (each controller manipulating the *Nova* internal states throughout the REDIS system). This first validation paves the way toward a complete LUC OS leveraging the OpenStack ecosystem.

The remaining of the article is as follows: Section 2 explains our design choices. Section 3 describes the OpenStack and gives first details of our current Proof-of-Concept. A preliminar validation of our prototype focusing on the Nova service is presented in Section 4. Finally Section 5 concludes and discusses future actions.

2 The LUC OS: Design Discussion

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. Thus the system in charge of operating such an infrastructure should be able to manage up to thousands of servers spread geographically. Delivering such a system is a tedious task where wrong design choices could prevent to achieve our goal. In this section we first discuss few conceptual considerations that led us to the LUC OS proposal and second remain the major services that the LUC OS should deliver.

2.1 From Centralized to Distributed Management

The first way that comes generally to the mind to pilot and use distinct clouds is to rely on classical models like federated approaches: each micro DC hosts and operates its own Cloud infrastructure and a brokering service is in charge of resources provisioning by picking on each cloud. While such approaches can be acceptable for elementary usages, advanced brokering services are mandatory to meet production environment requirements (monitoring, scheduling, automated

provisioning, SLAs enforcements ...). In addition to dealing with scalability and single point of failure (SPOF) issues, brokering services should integrate mechanisms similar to those that are already implemented at the level of IaaS managers [4, 15]. Consequently, the development of a brokering solution is as difficult as the one of a IaaS manager but with the complexity of relying only on the least common denominator APIs. While few standards such as OCCI [19] 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 like in the case of federated systems prevent from going beyond the provided APIs (or require when possible, intrusive mechanisms that must be adapted to the different systems).

The other way to operate such infrastructure is to design and build a dedicated system, *i.e.*, the *LUC Operating System*, in charge of operating all the geographically spread micro DCs in a distributed manner. A LUC OS will define and leverage its own software interface, thus extending capacities of traditional Clouds with its API and a set of dedicated tools. This offers a unique opportunity to go beyond classical federations of Clouds by addressing all crosscutting concerns of a software stack as complex as a IaaS manager and by revising in a fully distributed manner, mechanisms that have been traditionally implemented in a centralized one (service nodes).

The following question is now to analyze whether the collaborations between instances of the system, that is the service nodes, should be structured either in hierarchical way or in a P2P (*i.e.*, flat) one. Few hierarchical solutions have been proposed during the last years in industry [5, 6] and academia [9, 10]. Although they may look easier 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). 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, Peer-to-Peer (P2P) file sharing systems are a good example of software that works well at large scale and in a context where computing resources are geographically spread. While largely unexplored for building operating systems, peer-to-peer/decentralized mechanisms have the potential to natively handle the intrinsic distribution of LUC infrastructures as well as the scalability required to manage them. Hence, we propose to leverage advanced P2P mechanisms like overlay networks and distributed hash tables to design the LUC OS building blocks.

2.2 Cloud Capabilities

From the administrators and end-users point of views, the LUC OS should deliver a set of high level mechanisms whose assembly results in an operational IaaS system. Recent studies have showed that state of the art IaaS manager [23] were constructed over the same concepts and that a reference architecture for IaaS manager can could defined [20]. 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.

The challenge is thus to propose a distributed version of the aforementioned services by relying on advanced P2P mechanisms. However, designing and developing a complete LUC OS from scratch would be an herculean work, including several non-sense actions aiming at simply providing basic mechanisms available in most IaaS solutions. Instead of reinventing the wheel, we propose to minimize both design and implementation efforts by reusing as much as possible existing piece of codes. With this in mind, we propose to investigate whether the OpenStack solution [22] can be revised to fulfill the LUC infrastructure requirements. Concretely, we propose to determine which mechanisms can be directly used and which ones must be revisited with P2P algorithms. This strategy enables us to focus the effort on the key issues such as the distributed functioning, fault tolerance mechanisms, and the organization of efficient collaborations between service nodes of the infrastructure according to the constraints/requirements of the LUC OS.

3 Revising OpenStack

OpenStack is an open-source project that aims at developing a complete cloud management system. Similary to the reference architecture described in the previous Section, it is composed of several services, each one dealing with a particular aspect of a Cloud infrastructure as depicted in Figure 2.

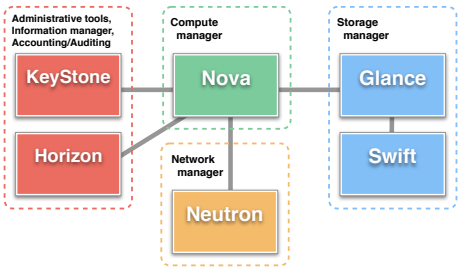


Figure 2: Services composing OpenStack.

OpenStack relies on two kinds of nodes: controller and compute node. The former is in charge of managing and distributing work to the latter that provides computing/storage resources to end-users. In other words, the controllers

correspond to the different services introduced in the previous section while the compute nodes host the VMs.

From the software point of view, the OpenStack architecture is based on the "shared nothing" principles: each controller (*i.e.*, each service) is connected to the others via two different way:

- **A messaging queue** that enables the collaboration between sub-services of a controller.
- **A SQL database** (DB) that stores inner states of a controller.

Finally, the controllers interact with each other through REST APIs or directly by accessing the inner-state that are stored in the differents DBs.

Considering the current structure of OpenStack, the main limitation to make it distributed is related to the SQL databases. Indeed, while OpenStack relies on the RabbitMQ messaging service, which is articulated around a centralized broker, there are few implementations of P2P messaging service such as ActiveMQ [24] or ZeroMQ [14] that would be adapted to the LUC requirements. The first way to bypass the MySQL limitation is to deploy each controller DB on each location and to synchronize the different DB instances with a dedicated mechanism [17]. By such a mean, when a controller processes a request and performs some actions on one site, changes in the inner-state are also propagated to all the other locations. From a certain point of view, it gives the illusion that there is only one DB for each service. Although the technique described has been used in different proof-of-concepts, current DB synchronization mechanisms are not scalable enough to cope with a LUC infrastructure deployed on large number of geographical sites.

Another approach is to replace the DBs used in OpenStack by a more suitable storage backend that would provide a better scalability. Distributed Hash Tables (DHTs) and more recently key/value systems built on top of the DHT concept such as *Dynamo* [8] have demonstrated their efficiency in terms of scalability and fault tolerance properties. In light of this, we have revisited the Nova controller, *i.e.*, the VM manager of OpenStack, in order to to replace the current MySQL DB system by *REDIS* [13], a *key/value store*. Technically speaking, we modified the Nova database driver. Indeed, the Nova software architecture has been organised in a way which ensures that each of its sub-services does not directly manipulate the database: they have an indirect access through a service called "nova-conductor" which in turn works with an implementation of the **"nova.db.api"** programming interface. Developers of Nova provide an implementation of this interface that is using *SQLAlchemy* to manipulate a relational database. We developed a second implementation of this interface that replaces every call to the *SQLAlchemy* by a call to a custom key/value store driver. This enables Nova's services to work with REDIS by only changing the database driver, limiting the level of intrusiveness in the original source code. Thanks to this modification, it is possible to instanciate a distributed cloud and operate it through a single instance of OpenStack composed of several Nova controllers deployed on distinct sites. Figure 3 depictes such a deployment. Each controller executes a REDIS instance that is configured to work in a clustering way with other instances. One or several controllers can be deployed on each site according to the expected demand in terms of end-users. Finally, a controller can be deployed either on a dedicated node or be mutalized with a compute one

as illustrated for Site 3. We higlight that any controller can provision VMs by orchestrating services on the whole infrastructure and not only on the site where it is deployed. Such a behavior is possible thanks to the AMQP bus and the key/value store that go through all controllers. Finally, it is noteworthy that key/value stores that focus on high-availability and partition tolerance criteria like Cassandra [18] would be more appropriate than REDIS for a production deployment. We chose REDIS for its usage simplicity.

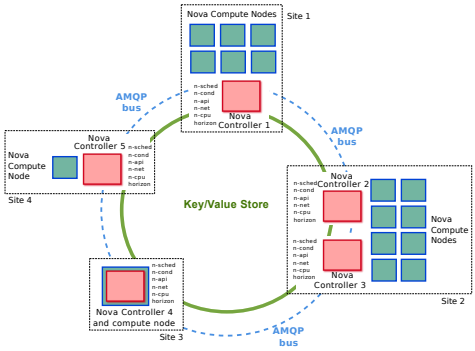


Figure 3: Nova controllers are connected through a shared key/value backend and the AMQP bus.

Our prototype is under evaluation. However, preliminary experiments have been performed throughout 4 sites of Grid'5000 including 12 compute nodes and 4 controllers overall. While this infrastructure was rather small in comparison to our target, it aimed at validating the interconnection of several controllers WANwide and the correct behaviour of OpenStack using our noSQL backend. Our prototype suceeded to provision 500 VMs in 300 seconds (each controller creating 125 VMs in parallel). A second experiment validated the provisionning of 2000 VMs in less than 30 min. We are currently performing comparisons between OpenStack using the historical MYSQL backend *v.s.,* using a key/value store backend. Our goal is to validate that manipulating internal states of Openstack through a noSQL deliver performances in the same order of the MySQL ones.

4 Experimental Validation

The validation of our proof-of-concept has been done via two 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 (*i.e.*, servers were located on the same geographical site). The second set focused on multi-site scenarios by comparing the impact of the latency on our distributed Nova service with the "State of the art" approach of distributed OpenStack.

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 conditions. 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 [16].

4.1 Impact of REDIS w.r.t MySQL

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 key/value store 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, data is spread over several nodes in a REDIS system. 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 on one cluster (containing 1 controller node and 6 compute nodes), using either REDIS or the MySQL backend. The first line of Table 1 shows the overhead related to the use of the combination of a key/value store and our code instead of MySQL. Our approach leads to an average API response time that is twice higher than MySQL. While our code is probably not mature enough, we need to conduct additional experiments to see the distribution of the completion time. Indeed, we expect that some requests (in particular the ones that should deal with advanced SQL features such as join or transaction) take a significant time to be achieved in the NoSQL world. The second line of Table 1 illustrates that distributing the key/value store on several nodes does not increase the overhead related to networking (in our case, data is distributed over four nodes). The third line represents measurements with data replication enabled. The overhead is 10% compared to a clustered configuration without replication. This small overhead is achievable thanks to the asynchronous strategy used in REDIS.

While the REDIS overhead may look critical at the first sight, it must be mitigated with Table 2 which shows that higher API response time does not necessarily mean lower overall reactivity. Indeed, with REDIS, OpenStack delivers performances that are in the same order of magnitude as with MySQL. Each result of Table 1 corresponds to the average response time for all API functions, which is not necessary correlated to the VM creation time as some API functions have a more significant impact than others on the VM creation time. As explained above, such results motivate us to conduct deeper analysis to understand the time distribution of API requests.

4.2 Multi-site Scenarios

The second experiment we made 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 REDIS proposal. While the deployment of a single MySQL node is a non sense in a production infrastructure as we mentioned 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 mechanisms.

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, 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 database backend, the creation of the 500 VMs has been fairly distributed among the available controllers in parallel.

Table 3: Time used 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: Time used 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	-

Table 3 and Table 4 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 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 peforms 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 500VMs to be created (*i.e.*, several bugs occur leading Nova to consider a lot of 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).

5 Conclusion and Future Work

Distributing the way Cloud are managed is one solution to favor the adoption of the distributed cloud model. In this document, we have presented our view of how such distribution can be achieved. We highlighted that it has however a design cost and it should be developed over mature and efficient solutions. With this objective in mind, we chose to design our system, the LUC Operating System, over OpenStack. This choice presents two advantages: minimizing the development efforts and maximizing the chance of being reused by a large community. As as first step, we modified the Nova SQL backend by a distributed key/value system. Although a more advanced validation of this change is required and the question of which metrics to use remains, this first prototype paves the way toward the distribution of additional OpenStack services.

Among the remaining services, the next candidate is the image service Glance. Indeed, as its images are already stored in fully distributed cloud storage software (SWIFT or CEPH), the next step to reach a fully distributed functionning with Glance is to apply the same strategy that we did with Nova. On the other hand, the situation may be different with some other services: Neutron works with drivers that may not be intented to work in a distributed way. In such situation alternatives will have to be found.

Finally, having a wan-wide infrastructure can be source of networking overheads: some objects manipulated by OpenStack are subject to be manipulated by any service of the deployed controllers, and by extension should be visible to any of the controllers. On the other hand, some objects may benefit from a restrained visibility: 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 enable to save network bandwidth and to settle policies for applications such as privacy and efficient data- replication.

We believe, however, that adressing all these challenges are key elements to promote a new generation of cloud computing more sustainable and efficient. Indeed, by revising OpenStack in order to make it natively cooperative, it would enable Internet Service Providers and other institutions in charge of operating a network backbone to build an extreme-scale LUC infrastructure with a limited additional cost. The interest of important actors such as Orange Labs that has officially announced its support to the initiative is an excellent sign of the importance of our action.

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] 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.

[3] 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.

[4] 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.

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

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

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

[8] 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.

[9] 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.

[10] 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.

[11] J. V. H. Gary Cook. How dirty is your data ? Greenpeace International Report, 2013.

[12] 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.

[13] J. Han, E. Haihong, G. Le, and J. Du. Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE, 2011.

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

[15] 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.

[16] 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.

[17] B. Kemme and G. Alonso. Database replication: A tale of research across communities. *Proc. VLDB Endow.*, 3(1-2):5–12, Sept. 2010.

[18] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[19] 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.

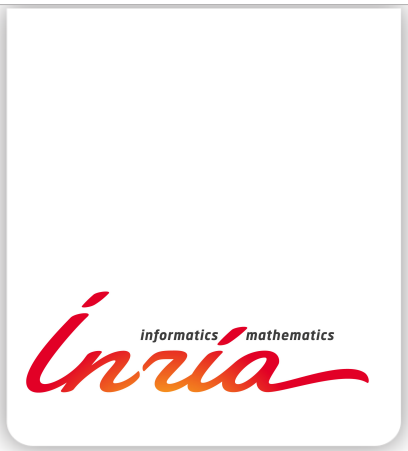
[20] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer*, 45(12):65–72, 2012.

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

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



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 0249-6399