The DISCOVERY Initiative

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

Adrien Lebre, Jonathan Pastor
ASCOLA Research Group
Mines Nantes / Inria / LINA UMR 6241
Nantes, France

Frédéric Desprez
Corse Research Group
Inria / LIG UMR 5217
Grenoble, France

ABSTRACT

Instead of the current trend consisting of building larger and larger data centers (DCs) in few strategic locations, the DISCOVERY initiative[1] 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.

Keywords

Cloud computing, IaaS Architecture, OpenStack, NoSQL, Peer to Peer

[1] `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[2] 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).

A preliminary study has established the fundamentals of such an *in-network distributed cloud* referred by the consortium as the *Locality-Based Utility Computing* (LUC) concept [2]. However, the question of how operating such an

[2] 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.

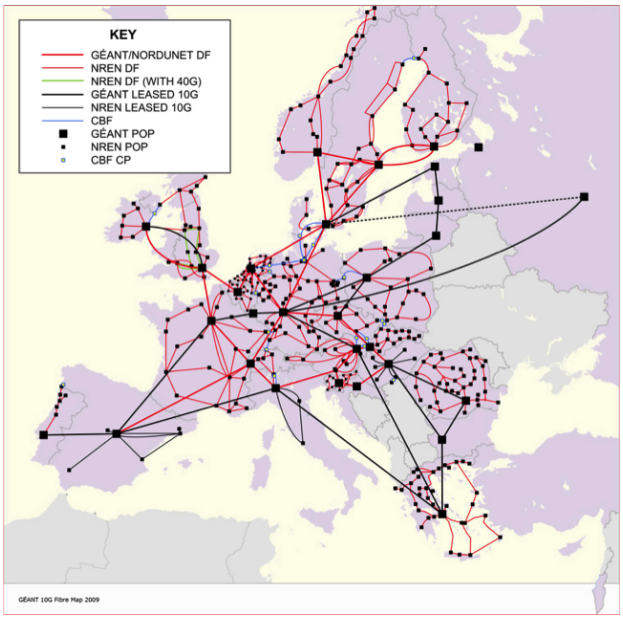

Figure 1: The European GÉANT backbone
GÉANT is the federation of all European Research and Educational Networks. Each black square corresponds to one network point of presence (*a.k.a.*. a PoP) that can host a nano/micro DC.

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, 20, 21]. 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 [21], 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. Finally Section 4 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 [18] 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 [22] were constructed over the same concepts and that a reference architecture for IaaS manager can could defined [19]. 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 [21] 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.

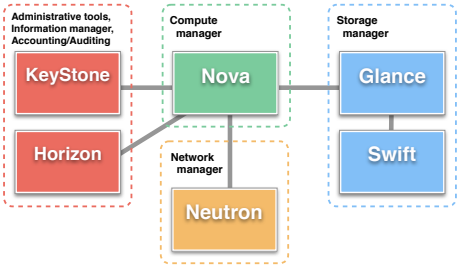

Figure 2: Services composing OpenStack.

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 [23] 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 [16]. 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 [17] would be more appropriate than REDIS for a production deployment. We chose REDIS for its usage simplicity.

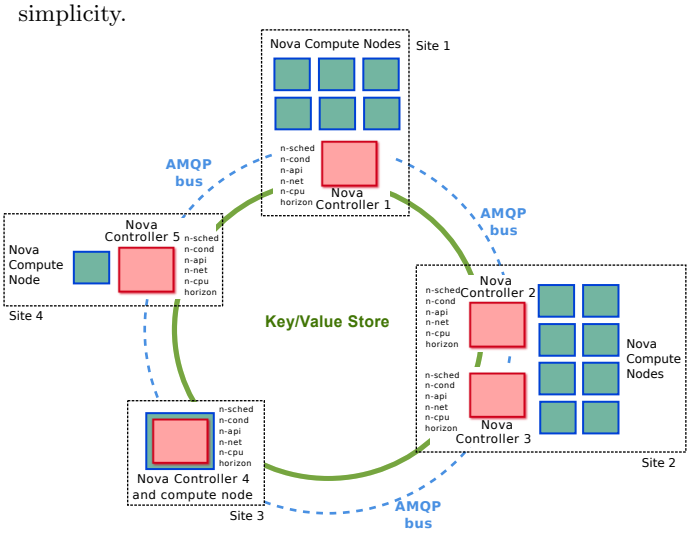

Figure 3: Nova controllers are connected through a shared key/value backend and the AMQP bus.

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. 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.

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

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

[18] 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.

[19] 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.

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

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

[22] 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.

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