Designing a Massively Distributed IaaS Manager by Leveraging OpenStack

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

Laurent Pouilloux
Avalon Research Group
ENS / Inria /LIP UMR 5668
Lyon, France

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

firstname.lastname@inria.fr

ABSTRACT

The deployment of micro/nano data-centers in network point of presence offers an opportunity to deliver a more sustainable and efficient infrastructure for Cloud Computing. Among the different challenges we need to address to favor the adoption of such a model, the development of a system in charge of turning such a complex and diverse network of resources into a collection of abstracted computing facilities that are convenient to administrate and use is critical. In this paper, we introduce the premises of such a system. The novelty of our work is that instead of developing from scratch, we revised the OpenStack solution in order to operate such an infrastructure in a distributed manner leveraging P2P mechanisms. More precisely, we revised the Nova service by leveraging a distributed key/value store instead of the centralized SQL backend. We present a first scenario that enabled us to validate the correct behavior of our prototype across distinct servers of the Grid'5000 testbed. We believe that such a strategy is promising considering the architecture complexity of IaaS managers and the velocity of open-source initiatives.

Keywords

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

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 [10].

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

The concept of micro/nano DCs at the edge of the backbone [11] 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. A preliminary study has established the fundamentals of such an *in-network distributed cloud* referred by the authors as the *Locality-Based Utility Computing* (LUC) concept [2]. However, the question of how operating such an infrastructure still remains. Indeed, at this level of distribution, latency and fault tolerance become primary concerns, and collaboration between servers of differents location must be organized wisely.

In this article, we propose, first, to 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 [6, 15, 16]. We explain, in particular, why federated approaches [3] are not satisfactory enough and why designing a fully distributed system that operates all resources makes sense. Moreover, we describe the fundamental capabilities the LUC OS should deliver. As these capabilities are sim-

[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.

ilar to existing IaaS manager and because it would be a non-sense technically speaking to develop the LUC OS from scratch we chose to instanciate the LUC OS concept on top of the OpenStack solution [16]. More precisely, we introduce, second, a first implementation of a distributed version of the *Nova* service of the OpenStack solution and validate its correct functionning through a first use-case deployed on top of Grid'5000 [1]. 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 how we revised it. 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 and a broker is in charge of resources provisioning by picking on each cloud. We claim that such a model cannot be considered as adequate as it relies on centralized brokers, which are exposed to problems like scalability and single point of failure (SPOF). Moreover, federations of clouds do not go far enough to operate a network of micro DCs as it would use the clouds rather than operate them.

The second 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, thus preventing from SPOFs and scalability issues. While working with a federation of heterogeneous clouds means working on the least common denominator APIs, a LUC OS will define and leverage its own software interface, thus extending capacities of a Cloud with its API and a set of tools. This allows to go beyond classical federations of Clouds and to implement, in a fully distributed manner, mechanisms that are traditionally centralized (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 [4, 5] and academia [8, 9]. 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 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. These conditions correspond to the LUC infrastructure ones. 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 [17] were constructed over the same concepts and that a reference architecture for IaaS manager can could defined [14]. 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 a solution such as the OpenStack one [16] can be revisited to fulfill the LUC infrastructure requirements. This strategy would enable 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.

3. REVISITING OPENSTACK

OpenStack is an opensource 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 a CC infrastructure as depicted in Figure 1.

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.

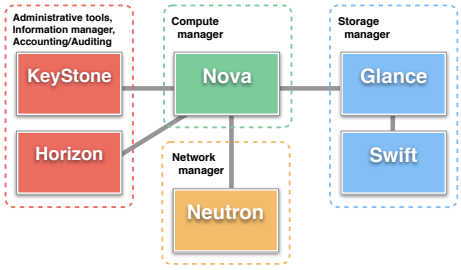

Figure 1: Services composing OpenStack.

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. The first way to bypass this limitation is to deploy each controller database on each location and to synchronize the different DB instances with a dedicated mechanism [13]. 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* [7] 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 *RIAK*[18], a *key/value store* that extends the principles followed by the *Dynamo*.Figure 2 shows the new architecture of the Nova controller.

The architecture used for the Nova service 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

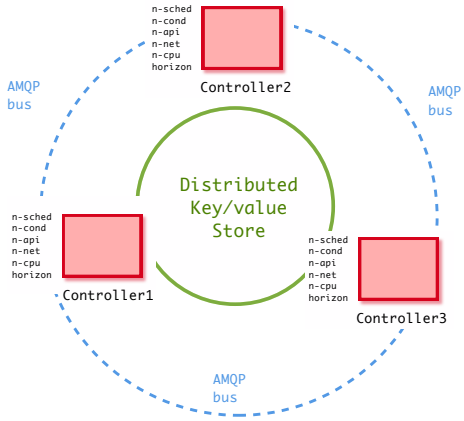

Figure 2: Nova controllers are connected through a shared key/value backend and the AMQP bus.

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 RIAK driver. This enables to make Nova's services working with RIAK by only changing the database driver: this limits the level of intrusiveness in the original source code.

4. EXPERIMENTAL VALIDATION ON GRID'5000

The validation of our prototype has been performed thanks to the Grid'5000 testbed [1]. Grid'5000 is a large-scale and versatile experimental testbed for experiment driven research in Computer Science, which enables researchers to get an access to a large amount of computing resources ($\sim$ 1000 nodes spread over 10 sites). This delivery of computing resources takes the form of bare-metal machines, on which fully customised software stacks can deployed, thus giving a very fine control of the experimental conditions.

Various tools have been developed to provide an ease of use, such as monitoring information about networking and power consumption or programming libraries to fine tune each aspect composing an experiment. With this in mind, we developed our prototype using the Execo framework [12] which helped us to deploy and configure each node composing our chosen software stack (Ubuntu 14.04, a modified version of OpenStack "devstack", and the RIAK key/value store).

Our validation scenario has consisted in the creation of 30 VMs through 10 different Nova controllers deployed on one cluster located in Nantes. This experiment enabled us to confirm that OpenStack's services were working correctly with the key/value store. Although further experiments are required to test the scalability as well as the effect of geographical distances on the reliability and efficiency, we are confident about our approach as several distributed key/value stores are already used WANWide.

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 paper, 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 and validated this prototype through a first series of experiments on top of Grid'5000. 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: as its images are already stored in fully distributed cloud storage software (SWIFT), 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.

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

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

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

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

[14] 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.

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

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

[17] 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.

[18] Documentation of RIAK, a distributed key/value store. `http://docs.basho.com/riak/latest/theory/dynamo/`.