Analysis of Control Traffic in a Geo-distributed Collaborative Cloud

Tatiana Sciammarella*, Rodrigo S. Couto†, Marcelo G. Rubinstein†,
Miguel Elias M. Campista* and Luís Henrique M. K. Costa*
* Universidade Federal do Rio de Janeiro - PEE/COPPE/GTA - DEL/POLI
† Universidade do Estado do Rio de Janeiro - FEN/DETEL/PEL
Email: {tatiana,miguel,luish}@gta.ufrj.br, {rodrigo.couto,rubi}@uerj.br

***Abstract*—Geo-distributed clouds are composed of several virtual machine servers scattered over sites interconnected by a wide area network. In this scenario, the traffic generated by control messages can be prohibitive, because of the high bandwidth costs of these networks. This paper evaluates the impact of this type of traffic in the scalability of the cloud. Based on experiments with the OpenStack orchestrator, we estimate that 100 servers with 15 instantiated virtual machines each send in average 2.7 Mb/s of control traffic to the infrastructure controller. Traffic bursts produced upon the creation and destruction of virtual machines may add to this average control traffic, further aggravating the problem. These results point out that, if the IaaS cloud is poorly designed or the network under provisioned, control traffic can represent a bottleneck especially for small clouds, typically the case of collaborative ones.**

I. Introduction

Cloud computing is growing at a fast pace in all production sectors, as a cost-effective solution for IT outsourcing. Different service models exist [1]. Infrastructure-as-a-Service (IaaS) stands out since it potentially reduces OPEX and CAPEX costs with computational resources. In IaaS, the infrastructure provider offers customized virtual machines (VMs) on demand to its clients. Each VM can be accessed via the Internet, even through a web browser. IaaS clouds can be geo-distributed, i.e., the infrastructure can be scattered throughout different sites and interconnected by a wide area network (WAN). As a consequence, the WAN connectivity can impact cloud resiliency and performance [2].

Large geo-distributed clouds are composed of interconnected autonomous sites, with hundreds or thousands of servers each [2]. These servers, whose main role is to host clients' VMs, are managed by a cloud orchestrator, such as OpenStack (www.openstack.org) or CloudStack (cloudstack.apache.org). The orchestrator performs different infrastructure control tasks, such as VM hosting, server selection, VM creation and deletion, user authentication, and statistics collection. Typically, each geo-distributed cloud has one or more machines acting as controllers, executing the main OpenStack or CloudStack modules. These modules exchange messages with modules at each site to accomplish different control tasks.

An important scenario for geo-distributed IaaS clouds is resource sharing among different institutions. For instance, universities and research centers have computational resources that can be shared in a single IaaS cloud. This type of service is called collaborative IaaS cloud, in which different institutions with the same interests share computational resources. Collaborative IaaS clouds, unlike other geo-distributed ones, can also include institutions with modest computational resources, instead of only relying on providers with large distributed datacenters. The drawback is that a controller cannot exist per site, since some of them may not have enough resources. A single centralized controller then can be used to control the entire cloud through a WAN, even though possibly introducing infrastructure deployment challenge concerned with system scalability. The control traffic is concentrated toward the controller, possibly overloading it. This higher control traffic can augment costs with bandwidth, which are typically higher for higher distances, as in WANs.

This work analyzes the impact of the control traffic between the controller and the VM servers in a collaborative cloud using OpenStack. The goal is to evaluate to which extent the control traffic can be considered an obstacle to collaborative cloud scalability. The contribution of this work is twofold: we unveil potential bottlenecks to the geo-distributed infrastructure and we analyze the message exchange between the cloud controller and the VM servers at the multiple sites. From our results, we conclude, for instance, that each VM server added to the infrastructure can increase the network traffic in average of 15 kb/s, whereas each idle VM can contribute with 0.77 kb/s. Consequently, we can estimate that in an IaaS cloud with 100 servers and 15 VMs each, an average control traffic of 2.7 Mb/s would be generated toward the controller. Although this value is not enough to surpass current network capacities, it can not be neglected in the cloud design, especially for smaller clouds, such as those called collaborative.

This work is organized as follows. Section II presents the collaborative cloud architecture. Section III overviews the cloud orchestrator used, OpenStack. Section IV describes the experimental setup and the obtained results. Section V presents the related work, pinpointing the original aspect of the analysis conducted in this paper. Finally, Section VI concludes this paper and investigates future work.

II. Collaborative Cloud Architecture

In this work, we rely on a geo-distributed architecture between three universities located in Rio de Janeiro, Brazil.

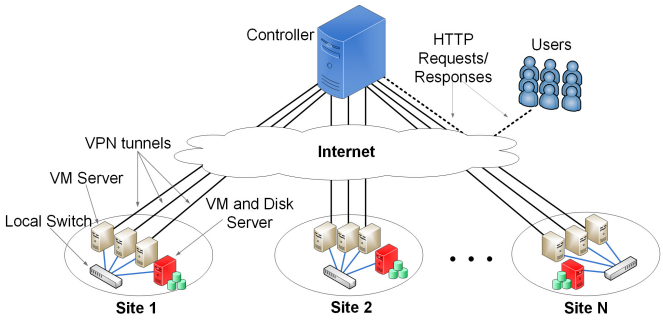


Figure 1. Geo-distributed architecture of our collaborative cloud.

The main goal is to share computational resources among participants from the institutions. Thus, during utilization peaks of the local infrastructure, users can use idle computational resources from other participants, shared through the cloud. On the one hand, participants can increase their computational power thanks to the cloud elasticity; and on the other hand, they must offer part of their own resources to the cloud. This is the idea of collaboration, possible because of the demand for resource utilization at different points in time.

The collaborative cloud architecture (Figure 1) is composed of sites installed at the universities, which are connected to a central Controller. Each site has machines playing the role of "VM Server" and "VM and Disk Server", interconnected by a local switch. Both machine types are used to host VMs, using the KVM (www.linux-kvm.org) hypervisor. The VM and Disk Server additionally centralizes the virtual disks of all VMs within a site, allowing live intra-site VM migration. The Controller, besides all site management functions, is also responsible for receiving and handling the requests from users for VM creation. Yet, it provides users with access interfaces to their VMs. VM Servers as well as VM and Disk Servers are connected to the Controller using VPN (Virtual Private Network) tunnels through the Internet. To manage the architecture of Figure 1, both Controller and Servers execute the OpenStack orchestrator, detailed in the next section.

The proposed scenario also supports small sites (e.g., laboratories), possibly with only one server available. In this case, running all cloud components in a single site would not reasonable, motivating our assumption of a single cloud Controller. Besides, participants often do not have enough background to deal with cloud management. In this case, all procedures can be conducted remotely by the cloud administrator. Note that the proposed cloud can be integrated to other collaborative and geo-distributed solutions, such as federated clouds [3], since the complete infrastructure is seen as a single OpenStack cloud. Federation may be one of the solutions needed to increase the number of supported sites without having a single control authority.

III. OpenStack

OpenStack (http://www.openstack.org) is an open-source software that manages computing resources of cloud infrastructures. This platform stands out for its great community of developers and for companies involved, such as Google and Intel. OpenStack has a modular architecture divided in projects which favors its customization for different scenarios. These projects are specialized services for each management task of the cloud. In this work, we use the Juno version of OpenStack.

A. Projects

OpenStack projects have different purposes. In this section, we focus the projects related to the services needed for an IaaS cloud, which is the case of our collaborative cloud. . The OpenStack projects of interest are:

- Dashboard (*Horizon*): Provides a web interface for system management. This project communicates with other projects by APIs (Application Programming Interfaces). It allows administrators and users to access cloud resources taking into account predefined policies.
- Compute (*Nova*): Interacts with the hypervisor to provide on-demand computing resources, by provisioning and managing virtual machines. Nova is composed by several modules that communicate with each other and are distributed in the infrastructure machines.
- Image Service (*Glance*): Provides discovery, registration, recovery, and storage of images for new VM creations. An image can be a new installation of an Operating System (OS) or even a disk copy of a pre-configured VM. Users can submit images to the Glance repository.
- Block Storage (*Cinder*): Provides persistent block-level storage as volumes to be used with VMs. These volumes represent virtual disks and can be accessed when associated to VM instances. When containing an image, a volume can be an initialization unity (boot) for a VM.
- Identity Service (*Keystone*): Manages user identities and provides access control to services.

OpenStack projects are divided into modules. In each project, modules communicate with each other using message queues implemented by the RabbitMQ middleware (www.rabbitmq.com). Also, a MySQL database is used for information storage related to each project.

Nova, Cinder, Glance, and Keystone projects, all used on our collaborative cloud, are examples of OpenStack projects divided into modules. The Controller hosts all modules related to APIs (nova-api, glance-api and cinde-api) and scheduling (cinder-scheduler and nova-scheduler), the database, a message queue and other auxiliary modules. Moreover, the Controller has modules for interacting with the user, e.g. to access VMs (nova-novncproxy) or for authentication (nova-consoleauth and Keystone project). A VM Server has two modules, one to provide communications with hypervisors (nova-compute) another to provide network functionalities to VMs (nova-network). These modules are not present on the Controller, since VMs are not hosted on this node. A VM and Disk Server has all the modules of a VM Server plus the cinder-volume module, which is used for managing volumes, and the NFS (Network File System) service used for virtual disk storage.

B. Communication between Services and Modules

In OpenStack, modules can communicate via message queues or directly by APIs. Moreover, modules communicate with the database in the Controller using MySQL commands. A message queue is only used for communication among modules of the same project. Direct communication is used for interaction among different project modules and with the database, which is centralized in the Controller in our architecture.

Message queues use the AMQP (Advanced Message Queuing Protocol) [4], implemented by the RabbitMQ message middleware. AMQP is an open protocol that aims at proving message exchanges among different applications, regardless their implementation. RabbitMQ follows a publish/subscribe model for message exchange. Hence, messages generated by modules are sent to the middleware and further stored on the appropriate queue. These messages remain stored until the modules interested in receiving them are ready to consume. Due to the geo-distributed characteristic of the collaborative cloud, messages generated on servers are sent to the RabbitMQ queue hosted on the Controller.

For direct communication, each OpenStack project has an API which provides a standard set of requests to allow other applications to access their services. These APIs are implemented as web services using the REST (REpresentational State Transfer) standard. This way, OpenStack services can be accessed via the Internet. Besides being used for interacting with components external to the infrastructure, APIs are used for communications among modules of different projects. As API services are hosted on the infrastructure Controller, this machine also centralizes traffic of this kind of communication. Moreover, the Controller also centralizes MySQL traffic, as this machine hosts the database used in all projects.

As previously mentioned, the Controller is responsible for all interactions among the modules, centralizing control traffic. Consequently, it is important to study the traffic generated on these interactions, to better dimension the network capacity allocated to the Controller node.

IV. Experimental Analysis

Our analysis focus on the traffic between the Controller and the different Servers. To accomplish that, experiments are conducted in an infrastructure similar to the one of Figure 1. Traffic is captured from the VPN interface of the Controller, in that way the WAN traffic is measured.

We only use VM and Disk Servers in our testbed. Compared with VM Servers, VM and Disk Servers have an additional component (cinder-volume) which communicates with the Controller via the VPN tunnel. Hence, VM and Disk Servers represent the worst case for the proposed analysis. In some experiments, we use four VM and Disk servers, characterizing a four-site cloud; while in other experiments we use only one server. To guarantee control over the experimental scenario and isolation to external factors, all sites are located at the same physical location. Also, these sites are connected by a local network using VPN tunnels, instead of the Internet. Table I contains the hardware configuration of each machine in our testbed.

Table I
Configuration of the machines used in our experiments.

Machine	CPU	RAM
Controller	Intel Core i7 CPU 860 @ 2,80 GHz	8 GB
Server 1	Intel Core i7-4930K CPU @ 3,40 GHz	32 GB
Server 2	Intel Core i7 CPU 860 @ 2,80 GHz	8 GB
Server 3	Intel Xeon CPU E3-1241 v3 @ 3,50 GHz	32 GB
Server 4	Intel Core2 Quad CPU Q9400 @ 2,66 GHz	6 GB

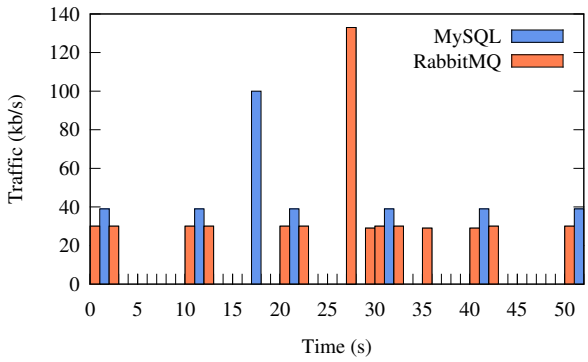


Figure 2. Traffic sample between the VM and Disk Server and the Controller along the time.

For all experiments, except those using samples along the time, we compute averages and confidence intervals of 95%. Intervals do not appear in some figures because they are too small. Based on traffic measurements, we conduct different analyses, presented next.

A. Impact of the number of VM and Disk Servers

In this experiment, we initially measure the traffic between a VM and Disk Server, with no VMs instantiated, and the Controller. Figure 2 presents the traffic generated by the message exchange between RabbitMQ and the modules nova-compute and nova-network, and also between the database and the cinder-volume. In the used infrastructure, the modules nova-compute and nova-network at a VM and Disk Server periodically send updates to the nova-conductor module, which runs at the Controller, using the message queue system of RabbitMQ. The nova-conductor then inserts this information at the database of Nova. Similarly, the cinder-volume module, present in each VM and Disk Server, periodically sends updates do the Controller. Nevertheless, unlike Nova, it directly communicates with the database of Cinder using MySQL.

Figure 2 shows that RabbitMQ communicates with server modules every 10 s to update service state. Between 20 and 30 s there is a burst corresponding to an update of the list of server instances (this update is done every 60 s). Similarly, the cinder-volume (MySQL label in the figure) reports its state every 10 s and, between 10 and 20 s there is a burst, related to the updates concerning information of volumes (periodically

Table II
RABBITMQ AND MYSQL TRAFFIC GENERATED FROM A SINGLE VM AND DISK SERVER TO THE CONTROLLER.

Type of Traffic	Average Traffic (kb/s)
RabbitMQ	9.72
MySQL	6.03
Total	**15.75**

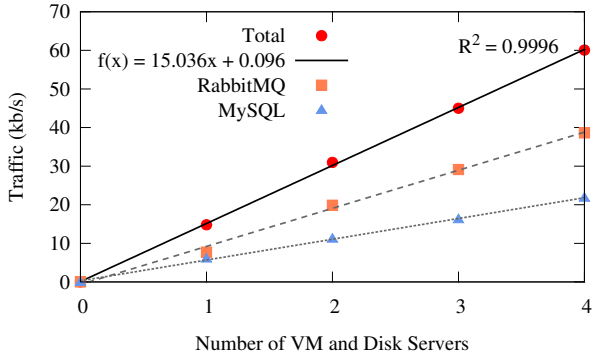


Figure 3. Network traffic for an increasing number of VM and Disk Servers.

done every 60 s). Table II[1] shows the contribution of these two communication types to the average network traffic, computed during 60 s, in 25 different experiments when only Server 1 is connected to the Controller. We can observe that most traffic involves RabbitMQ, i.e., the traffic generated by Nova. Moreover, it is possible to verify that the total traffic is near 15 kb/s.

Then, the number of VM and Disk Servers connected to the Controller is incremented and the impact on the traffic is evaluated. For each VM and Disk Server added, we measure the traffic during 60 s. This procedure is repeated 10 times. Figure 3 presents the total network traffic related only to RabbitMQ and MySQL, as a function of the number VM and Disk Servers. We can observe that the network traffic grows linearly with the number of servers. Since there is a linear behavior, we conduct linear interpolation of the network traffic, which results in the lines plotted in the same figure. The R^2 value indicates the quality of the fit of the linear regression. R^2 varies from 0 to 1, where $R^2 = 1$ represents a perfect line. Concerning the total network traffic, the R^2 value found is 0.9966. Hence, from the linear function found it is possible to conclude that each server adds, in average, 15 kb/s to the network traffic. This result matches that of Figure II. Hence, by extrapolation, in a 100-server scenario, 1.5 Mb/s would flow through the Controller interface.

B. Impact of the number of VMs per VM and Disk Server

In this experiment, we vary the number of VM instances in a single VM and Disk Server (Server 1) to analyze the impact on the network traffic. As the number of VM instances and volumes grows, more data needs to be sent to the Controller to update the Nova and Cinder databases. To evaluate the impact of an increasing number of VMs instantiated in the network, we capture the traffic generated after a well-succeeded creation. In other words, we do not consider the traffic related to the VM creation procedure. To each number of instantiated VMs, the traffic is measured during 60 s. This experiment is repeated 10 times for each number of VMs. The VMs used in the experiment and in the remaining of the paper run the CirrOS operating system, have 64 GB of RAM and 1 virtual CPU. It is worth noting that control messages do not vary neither according to the VM configuration nor with the number of CPUs because the messages sent are generic information such as VMs state, VMs IDs, etc. Consequently, the specific VM configuration does not change the behavior observed in the following experiments.

[1]The table does not show the confidence interval since it is negligible.

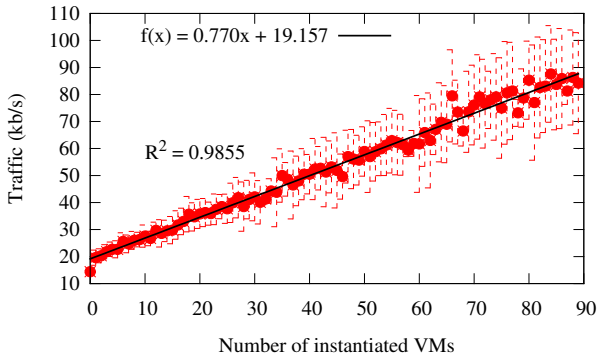


Figure 4. Network traffic for an increasing number of VMs.

Figure 4 shows the experiment result. We can observe a linear behavior, even though the variation grows with the number of instantiated VMs. The linear regression of the obtained results generates a function with R^2 equal to 0.9855. Considering the linear function, we estimate that each VM generates an additional average traffic of 0.77 kb/s. In a scenario with 100 servers containing 15 VMs each, 1,500 VMs in total, the traffic generated would be 1.155 Mb/s. In this scenario, summing up the traffic from VMs to the traffic from the servers, analyzed in Section IV-A, we have a total traffic of approximately 2.7 Mb/s.

C. Impact of creation and deletion of multiple VMs

These experiments analyze the traffic behavior during VM creation and deletion. OpenStack allows the instantiation of the VM from an image stored in an ephemeral disk unit, which stores data as long as the associated VM exists; or from an image stored in a Cinder volume, which offers persistent storage. When a user requests a VM initialization without volume for the first time, Glance sends an image to the VM and Disk Server through the VPN tunnel, which is locally stored at the server. In this case, if a new VM instance with the same image is created in the same VM and Disk Server, the image does not need to be sent another time through the network. In case of instances with initialization from the volume, first an empty volume is created and then the image

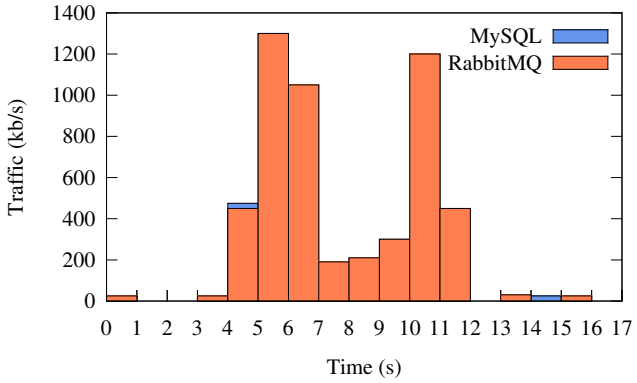


Figure 5. Traffic generated in VM creation and deletion.

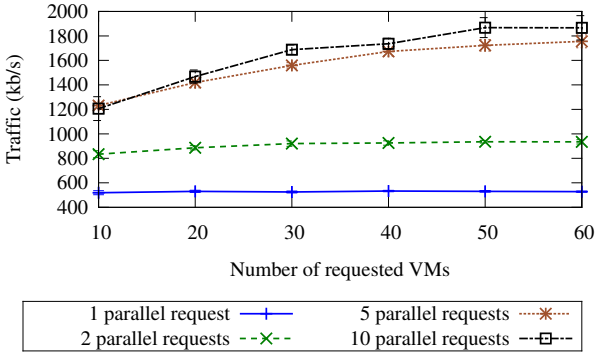


Figure 6. Traffic generated in parallel VM creation and destruction.

is transferred to the VM and Disk Server, and copied to the volume managed by Cinder. Unlike Nova, Cinder does not have a cache policy [5]. As a consequence, whenever a new instance with volume initialization is created, the image is transferred through the network.

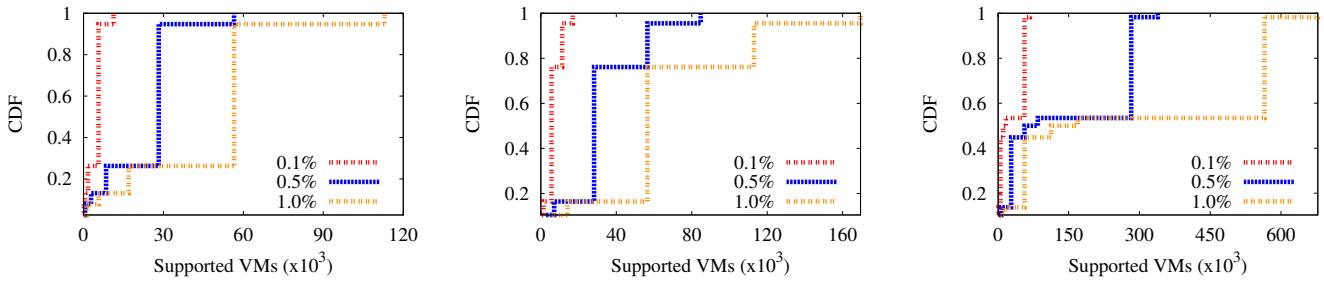
The typical VM creation in an IaaS cloud is from an ephemeral disk (i.e., based on image), given the high traffic generated for VM creation using volumes. Thus, cloud use cases must be rethought to restrict VM utilization initialized from volume in order to avoid overloading the network. To evaluate only the control messages generated, in this section, we analyze the network traffic during consecutive creation and destruction of VMs based on image. The traffic is evaluated after the existence of a VM image in the cache of Glance. As a consequence, the experiments do not depend on the configuration set for the VMs. First, Figure 5 shows the impact on the network traffic of a user request for a VM instance creation and, at the end of the process, for VM destruction. Note that the maximum traffic is approximately 1.3 Mb/s for VM creation and 1.2 Mb/s for destruction.

To provide more details, we employ an OpenStack benchmark called Rally (http://wiki.openstack.org/wiki/Rally), that performs different cloud utilization patterns. This benchmark offers pre-defined scenarios, such as VM creation requests. Rally was employed in related work [6], [7]. In the following experiment, we employ a Rally scenario called `boot-and-delete.json`, which is one of the simplest scenarios available in Rally. In this scenario, we define a given number of VMs to be created in a cloud. In this scenario, Rally generates a request for the Controller to create a VM, waits for its creation and then requests its destruction. The same procedure is repeated until the defined number of VMs is reached. Moreover, using the `boot-and-delete.json` scenario, we can specify the number of parallel requests sent to the Controller, simulating the existence of concurrent users in the cloud. For example, when setting five parallel requests, Rally sends five requests to the Controller at the same time and, after a VM is created, a new creation request is sent to maintain five parallel requests. In our experiments, we perform requests to the Controller to create VMs in Server 1. We employ different number of parallel requests, varying from 1 to 10. The experiment is performed 10 times for each number of requested VMs and parallel requests. Figure 6 plots the results of this experiment. Note that, even for a small number of concurrent users, the control traffic is not negligible. For example, ten users requesting 10 VMs generate a traffic of 1.2 Mb/s. This can lead to a high occupancy in low capacity networks, especially if we also consider the traffic generated periodically by control messages, as seen in Sections IV-A and IV-B.

Figure 6 also shows that the average traffic doubles when we increase the number of parallel requests from one to two. Nevertheless, the traffic generated for five parallel request is not five times greater than that of a single request (one parallel request). Moreover, from five to ten parallel requests, the traffic does not change, since our VM and Disk Server does not support all requests at the same time.

D. Impact of control traffic in real WAN

To analyze the impact of the control traffic in a real WAN, we evaluate the number of VMs supported in a network considering the capacity of its links. In other words, we evaluate the number of VMs supported in a cloud if all flows between VM and Disk Servers and the Controller were sent through a bottleneck link (e.g., the case where this link is the last hop to access the Controller). We employ three Research and Education Networks, that span a wide geographical area: RNP (www.rnp.br), from Brazil; RENATER (www.renater.fr), from France, and GEANT (www.geant.org), from Europe. For each network, we evaluate the number of supported VMs in a link assuming that a given amount of its capacity was reserved for the control traffic: 0.1%, 0.5%, and 1%. Figure 7 shows the CDFs (Cumulative Distribution Functions) for the number of Supported VMs on the links of each network. In all networks, we can note that the choice of links to access the Controller, which depends on the Controller location, highly impacts the number of supported VMs. For example, in RNP with 1% of capacity reserved, 13.15% of the links supports less than or equal to 5,600 VMs, although there are links that support about 56,000 VMs. Hence, depending on the Controller location, we can experience a large difference on the



(a) RNP. (b) RENATER. (c) GEANT.

Figure 7. Supported VMs in each link, considering different fraction of reserved bandwidth.

number of supported VMs. If the amount of traffic is too high, designing a cloud network would have to take the possibility of multiple Controllers into account, even in collaborative scenarios.

Note that we provide a baseline analysis on the number of supported VMs, showing that even the control traffic can highly impact this number. Nevertheless, in a cloud design, we must add the traffic generated from VM applications and cloud operations (e.g., when creating VMs).

V. Related Work

A scalability analysis of OpenStack components is performed by Gelbukh in [6]. Gelbukh evaluates the number of parallel requests that an OpenStack infrastructure supports. The results show that the infrastructure supports 250 VM creation requests in parallel. Moreover, Gelbukh shows that it is possible to create 75,000 VMs in the analyzed infrastructure. Indeed, these results are specific for a given hardware infrastructure and the OpenStack modifications proposed. Nevertheless, the work shows that it is indeed possible to achieve a high scalability level in OpenStack-based clouds.

Cisco also performs experiments to stress the OpenStack components and determine, for example, the number of VM servers a single controller can support and the number of VMs can be hosted in a VM server [7]. The results show that the number of VM servers is limited by RabbitMQ, which is the solution employed to exchange messages between controller and servers, as described in Section III-B. Moreover, the results show that the time needed to create a VM increases as the number of VMs grows, even when these VMs are idle. Using these results and other findings, the work states that the number of VMs that can be created is limited, even if RAM memory is still available for VM creation. Hence, the analysis suggests that a server is able to use 40% of its theoretical capacity to host VMs, evaluated as a function of the amount of memory available [7].

Those studies consist of large-scale experiments which stress different OpenStack modules. They do not analyze, however, the charge generated by control messages. In this way, given the importance of the network in a geo-distributed cloud, our works fills a gap in the literature by analyzing the scalability from the viewpoint of network capacity.

VI. Conclusions and Future Work

This work analyzed the impact of the control traffic on a collaborative cloud, which employs VM servers connected to a centralized controller through a WAN. This scenario is typically employed in small and medium-size collaborative clouds. Our experiments showed that, although the control traffic is relatively small, it must be considered when choosing the controller location in the WAN. For example, in a cloud with 100 servers and 15 VMs per server, we estimate an average control traffic of 2.7 Mb/s. This number can increase if we consider the traffic generated to create and destroy VMs.

As future work, we plan to extend our analysis to consider other OpenStack projects, such as Neutron (advanced networking service) and Ceilometer (monitoring service). In addition, we plan to analyze the impact of the network latency in OpenStack components, caused by the communication between the Controller and Servers in a WAN.

Acknowledgment

The authors would like to thank RNP, CNPq, CAPES, and FAPERJ for partially funding this work.

References

[1] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, 2015.

[2] C. Develder, M. De Leenheer, B. Dhoedt, M. Pickavet, D. Colle, F. De Turck, and P. Demeester, "Optical networks for grid and cloud computing applications," *Proceedings of the IEEE*, vol. 100, no. 5, pp. 1149–1167, 2012.

[3] C. A. Lee, "Cloud federation management and beyond: Requirements, relevant standards, and gaps," *IEEE Cloud Computing*, vol. 3, no. 1, pp. 42–49, Jan 2016.

[4] ISO/IEC, *ISO/IEC 19464. Information technology – Advanced Message Queuing Protocol (AMQP) v1.0 specification*. ISO/IEC, 2014.

[5] Cinder, *Generic image cache functionality*, OpenStack Cinder Team, Dec. 2015, http://specs.openstack.org/openstack/cinder-specs/specs/liberty/image-volume-cache.html - Acessado em Dezembro de 2015.

[6] O. Gelbukh, *Benchmarking OpenStack at megascale: How we tested Mirantis OpenStack at SoftLayer*, Mirantis, Feb. 2014, http://www.mirantis.com/blog/benchmarking-openstack-megascale-tested-mirantis-openstack-softlayer/ - Acessado em Dezembro de 2015.

[7] Cisco, *OpenStack Havana Scalability Testing*, Cisco Systems, Feb. 2014, http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/OpenStack/Scalability/OHS.pdf - Acessado em Dezembro de 2015.