Building an IaaS Cloud with Droplets: a Collaborative Experience with OpenStack

Rodrigo S. Couto[a,*], Hugo Sadok[b], Pedro Cruz[b], Felipe F. da Silva[b], Tatiana Sciammarella[b], Miguel Elias M. Campista[b], Luís Henrique M. K. Costa[b], Pedro B. Velloso[b], Marcelo G. Rubinstein[a]

[a] *Universidade do Estado do Rio de Janeiro - PEL/DETEL*
CEP 20550-013, Rio de Janeiro, RJ, Brazil
[b] *Universidade Federal do Rio de Janeiro - COPPE/PEE/GTA - POLI/DEL*
P.O. Box 68504 - CEP 21941-972, Rio de Janeiro, RJ, Brazil

Abstract

The goal of a collaborative cloud is to promote resource sharing among different institutions using existing computational infrastructure. To accomplish that, we propose the PID (Platform for IaaS Distribution) cloud, where users from research institutions can benefit from IaaS services with minimum management and maintenance effort and with no subscription fees. The only requirement is to share at least one server capable of running virtual machines. To build PID, we rely on the OpenStack orchestrator as the basis of our implementation and, consequently, modifications are needed to accommodate the geo-distributed architecture. We present such modifications, which are concentrated on a different virtual machine scheduling, new hierarchical roles for users, and adapted web interfaces for management and control. Based on these modifications, we deploy a prototype in three universities in Rio de Janeiro. We experimentally evaluate PID regarding the latency between sites and the scalability according to the number of servers and virtual machines. Based on these experiments, we can estimate that an OpenStack cloud with a few hundred servers with tens of virtual machines each would add, on average, a few Mb/s of control traffic toward the Controller. We then conclude that the amount of control traffic generated by OpenStack services is not enough to surpass network capacities of our academic sites. Our results also show that the additional latency expected in a geo-distributed cloud has little or no impact on the OpenStack network component, called Neutron.[1]

Keywords: Cloud computing; collaborative cloud; IaaS; OpenStack.

1. Introduction

Cloud computing provides users with a cost-effective solution to processing and storage services. The key idea consists of using virtualized resources instantiated on demand at the cloud, instead of using only local

*Corresponding author.
Email addresses: rodrigo.couto@uerj.br (Rodrigo S. Couto), sadok@gta.ufrj.br (Hugo Sadok), cruz@gta.ufrj.br (Pedro Cruz), felipe@gta.ufrj.br (Felipe F. da Silva), tatiana@gta.ufrj.br (Tatiana Sciammarella), miguel@gta.ufrj.br (Miguel Elias M. Campista), luish@gta.ufrj.br (Luís Henrique M. K. Costa), velloso@gta.ufrj.br (Pedro B. Velloso), rubi@uerj.br (Marcelo G. Rubinstein)

[1] The final publication is available at Elsevier via http://dx.doi.org/10.1016/j.jnca.2018.05.016

resources. By delegating computing services to the cloud, users can reduce costs with hardware acquisition and also with infrastructure management and maintenance. Cloud services typically run on single-owned third-party data centers, such as those from Amazon EC2 [1] or Microsoft Azure [2] and, consequently, are not for free. Users have at least to pay for fees according to the amount of virtual resources used. Even though this cost is probably lower than maintaining a whole physical infrastructure, users from universities and research labs may be still not willing to pay the price. Especially for scientific computing, that usually deals with large amounts of data storage and processing-intensive applications, related fees may be prohibitive. Although building its own cloud infrastructure seems an option at a first sight, this would take away almost all the benefits of cloud computing.

In this paper, we propose a collaborative and distributed cloud infrastructure, hereinafter called *PID – Platform for IaaS Distribution*, based on the Infrastructure-as-a-Service (IaaS) model. We borrow from the volunteer cloud paradigm [3] the concept of users contributing with their own idle resources to build the cloud infrastructure. The difference in our case, however, is that universities and research labs must collaborate with a set of dedicated machines able to run a virtual environment, similar to PlanetLab [4], NOVI [5] and FITS [6]. This simplifies the resource allocation problem, resulting in a new paradigm named *collaborative cloud*. In PID, the collaborative cloud can be composed of several droplets, defined as a small set of dedicated resources shared by each participant. These droplets together offer an aggregated geo-distributed pool of resources, allowing users to instantiate and to have access to virtual machines running within their own sites or on any participant site. PID, therefore, can be built upon using existing idle resources available at universities and research labs. As a result, costs with hardware and with the corresponding management and maintenance drop down; at the same time, all fees related to cloud utilization vanish. It is important to notice that our main goal is not to replace existing clouds but to manage underutilized resources in research labs using a cloud orchestrator. The proposed architecture enables the deployment of a community cloud, where users from different organizations can share a cloud infrastructure [7]. Thus, research labs might share machines with potential idle resources that do not provide critical services. For instance, machines dedicated to simulations are suitable to be shared with the cloud since long periods of idle resources might be frequent.

The collaborative nature of PID leads to a geo-distributed topology, *i.e.*, droplets at different sites composing a single cloud. This characteristic intrinsically provides resilience to PID [8] since a failure on a site would not bring down the entire cloud. In our collaborative cloud, the distributed underlying infrastructure is partially transparent to the users, who can choose where to instantiate their virtual machines. Nevertheless, local administrators (*i.e.*, droplet administrators) may want to restrict the access to their resources to avoid exhaustion of their computational power by external users. Hence, PID implements a hierarchical role-based management system that allows local administrators to control their own resources. Note, however, that even though we have observed that several universities and research labs can only share modest computational resources, our system can also include institutions with less restricted infrastructure. In this sense, our architecture is general, as it can deal with droplets and resource-rich sites.

The design of this scientific collaborative cloud faces different challenges. The cloud must allow users to specify the positions of their virtual machines in the network. They may need virtual machines close to each other (reducing latency) or sparsely distributed (privileging resilience). The authentication policies of each institution must be considered in the cloud. That is, local authentication services must be federated and access to resources must be controlled. We address these challenges in the proposed architecture and deploy a collaborative cloud prototype using the OpenStack [9] orchestrator. To achieve the design goals, we add a new type of user to OpenStack, called local administrator, to perform management tasks on a single site. In addition, we modify the default OpenStack scheduler to allow deciding where to instantiate virtual machines, considering the geo-distributed infrastructure. Finally, we configure OpenStack to meet our goals by adding, for example, an identity federation. For the sake of simplicity, we assume that droplets may not be able to manage a cloud controller and this task must be conducted at a single and reliable point. This centralization raises questions concerning cloud scalability. In this work, we then experimentally evaluate the control traffic exchanged between the PID Controller and an increasing number of servers and virtual machines. The goal is to evaluate to which extent the control traffic can be considered an obstacle to collaborative clouds. From our results, we observe that each VM (Virtual Machine) server adds an average of 15 kb/s of control traffic to the network; while each idle VM can contribute with 0.77 kb/s. Consequently, we can estimate that an IaaS cloud with a few hundred servers with tens of VMs each would add, on average, a few Mb/s of control traffic toward the Controller. In addition, we verified in our experiments that the number of simultaneous VMs creation and deletion is upper-bounded, limiting the amount of control traffic in 2 Mb/s. Although these values should not be neglected in the cloud design, especially if smaller droplets are selected to host the Controller, we conclude that the amount of control traffic is not enough to surpass most network capacities of our academic sites. For example, one of our sites is able to support up to 2,085 VMs if we reserve only 1% of its link capacity to control traffic. We also evaluate the latency in our infrastructure, which presents low values and cannot cause severe impact on interactive applications.

The paper is organized as follows: Section 2 describes related work. Section 3 introduces the PID cloud, detailing OpenStack modifications and the employed mechanisms. Section 4 provides an experimental analysis of the control traffic exchanged in our infrastructure. Finally, Section 5 concludes this work and points our future research directions.

2. Related Work

Traditionally, cloud architectures are single-owned, *e.g.*, Amazon EC2 or Microsoft Azure, and are generally geo-distributed. The notion of distributed clouds refers to two different architectural aspects. First, it may refer to the control, namely, the cloud orchestrator, which manages the cloud resources and user requests. The main purpose of the distributed control is to provide scalability, by load balancing the Controller; and resilience, by employing redundancy. Second, in opposition to a single-location, a cloud might be geographically distributed aiming at placing resources closer to the users and to the data center. The

main objective is to reduce network load and the delay experienced by users with the cloud services.

There are solutions that implement geo-distributed clouds using traditional cloud orchestrators. OpenStack cascading [10] integrates multiple clouds in a concept also known as "cloud over clouds". The basic idea is that an OpenStack parent cloud (Cascading OpenStack) manages other sites (Cascaded OpenStacks) using the standard OpenStack API (Application Programming Interface). A single Controller hides the distributed aspect of the architecture from the customer or the cloud user. OpenStack cascading allows at the same time lower costs for each cloud owner and control over the respective infrastructure. Other solutions propose the concept of cloud federation [11, 12]. Fogbow [13] federation integrates multiple clouds, running different orchestrators, to build a collaborative cloud. The main idea is that a given participant is able to borrow resources from other clouds if their local resources are insufficient. The aforementioned solutions, however, create a geo-distributed cloud but the sites actually work independently behind a single endpoint, which provides API services. In this case, each participant must deploy its own cloud to add it to the infrastructure. This is reasonable since most geo-distributed clouds are composed of large-sized sites with hundreds or thousands of servers each. However, in the droplet case, these solutions may not scale since sites are simple. For example, in PID, the cloud must support sites with only one server and with no staff able to manage a cloud infrastructure. It is important to notice that, as PID employs OpenStack, our cloud can be easily integrated into a federation such as Fogbow.

In addition to geo-distributed clouds, there are several collaborative clouds based on the volunteer cloud paradigm, in which the key idea consists of employing underused resources of non-dedicated machines [3]. This type of cloud is multi-owned by nature since multiple participants add their resources to the cloud. Based on this concept, there are some initiatives to use idle resources from desktop computers to provide a Desktop Cloud. For instance, Cloud@Home [14] exploits underused resources of personal computers at home to build a cloud network composed of individual users over the Internet. Another interesting project is Nebula [15], which is a distributed edge cloud for data storage and computation, in which the main assumption is that data is widely distributed in many applications. Thus, Nebula applies the volunteer cloud concept to use edge resources that are closer to the data. The UnaCloud [16] platform is also a collaborative cloud and takes advantage of idle desktop resources inside a single university, increasing the utilization of these computers. Similar to UnaCloud, PID is a collaborative cloud that aims at using resources from universities. The main difference in our approach consists of expanding the cloud to multiple sites from different universities geographically distributed. In addition, PID employs dedicated machines to simplify resource management. Consequently, PID is an intermediate solution between employing volunteer cloud, given our multi-owned collaborative resources, and deploying a single-owner private cloud, since we employ dedicated machines. In [17], the authors propose a collaborative storage cloud. PID creates a cloud that provides not only storage but a complete IaaS environment. Finally, the project Harmony [18] is a platform for collaborative cloud environments, but while its focus is on the reputation and resource management of cloud providers, PID concentrates on the distributed administration aspects of the collaborative cloud.

In this article, we extend our previous work [19]. Our experiments in [19], which are also presented in this work, analyze the control traffic generated in an OpenStack cloud. They show that if the cloud network is underprovisioned, the control traffic can impact the cloud capacity in terms of the number of supported VMs. Based on these results, we propose in this article an entire collaborative cloud architecture, detailing the main aspects of our cloud. Then, we experimentally evaluate performance metrics in our prototype to verify if our cloud scales. There are also other studies in the literature analyzing OpenStack scalability. However, they do not analyze the network traffic generated by control messages in OpenStack, as performed in this work. Gelbukh [20] performs experiments to analyze the scalability of OpenStack-based clouds. Gelbukh shows that it is possible to create 75,000 VMs in the considered infrastructure and that it is able to support 250 VM creation requests in parallel. Although his work is specific for a given hardware configuration, it shows that OpenStack can achieve high scalability levels. In a technical report presented in [21], Cisco evaluates the number of VMs that can be managed by a single Controller and that can be hosted on a single VM server. This report shows that RabbitMQ, which is the solution adopted by OpenStack to exchange control messages, limits the number of VM servers in the infrastructure. The results also show that the time to create VMs increases as the number of VMs in the cloud increases, even when these VMs are idle. Finally, the report recommends that even if there is enough RAM to instantiate more VMs in a server, we should keep the physical memory utilization under 40%, considering all instantiated VMs. This limitation is useful to avoid, for example, high utilization of cloud components such as RabbitMQ.

The interconnection between different data centers of cloud providers is also studied in the literature. While the authors in [22] use, among other attributes, the network to compare different cloud providers, CloudSurf [23] creates a platform to monitor the networks of cloud providers. In the same direction, Persico *et al.* [24] evaluate the performance of the wide-area networks connecting VMs running on single-owned clouds. In the present paper, we conduct a similar analysis of PID prototype, in order to check if the networks connecting PID droplets are a bottleneck. Finally, Garcia *et al.* [25] show that transferring data between data centers of cloud providers can lead to very high fees and propose a strategy to reduce such costs. PID cloud eliminates such fees by following the community cloud paradigm.

3. PID Cloud

PID cloud has five main design goals: *Elasticity*, *Local control*, *Cost efficiency*, *Affordability*, and *Resilience*. Elasticity can be achieved as virtual machines are instantiated on demand over the whole distributed infrastructure. Local control is possible as at least the local infrastructure remains under control of the local administrator. Additionally, to the Cost efficiency inherited from traditional clouds, in PID cloud costs are limited to the shared infrastructure and no other fees are imposed. Affordability is obtained as participants may enter the system with small infrastructure contributions (droplets). Finally, Resilience is provided by the distributed nature of the collaborative infrastructure.

PID cloud follows the IaaS service model, allowing its users to create and have access to VMs hosted

at the servers (VM Servers) assigned by the participating institutions. These VM Servers are standard machines with a hypervisor installed, in our case KVM [26]. The use cases of PID cloud target users such as researchers, professors, and students. For example, students and researchers may use their VMs to perform simulations. These VMs can be easily replicated through the infrastructure running, for example, multiple parallel simulations, reducing the completion time. In addition, VMs can be used in hands-on courses, so that students can access pre-configured VMs built specifically for that course. These VMs can be accessed from any machine that has access to the Internet, reducing the cost building these labs. Hence, the lab class can be focused on teaching a given method or technology, rather than wasting time configuring machines and libraries in each physical machine. For example, a single VM with Mininet [27] and OpenDaylight [28] can be used in a Software-Defined Networking (SDN) hands-on course. This VM can be pre-configured with all the needed software and then replicated to all students, which can access their VMs even in their homes.

Figure 1 shows the architecture employed in the PID cloud. Each participating institution is called a site or a droplet and its VM Servers are connected to the Controller machine through VPN (Virtual Private Network) tunnels established over the Internet. The Controller machine manages the overall infrastructure, deciding, for example, where a particular VM will be hosted, namely, in which VM Server. The Controller also offers a web interface that allows users to manage all the lifecycle of their VMs, for example, offering functions to create VMs.

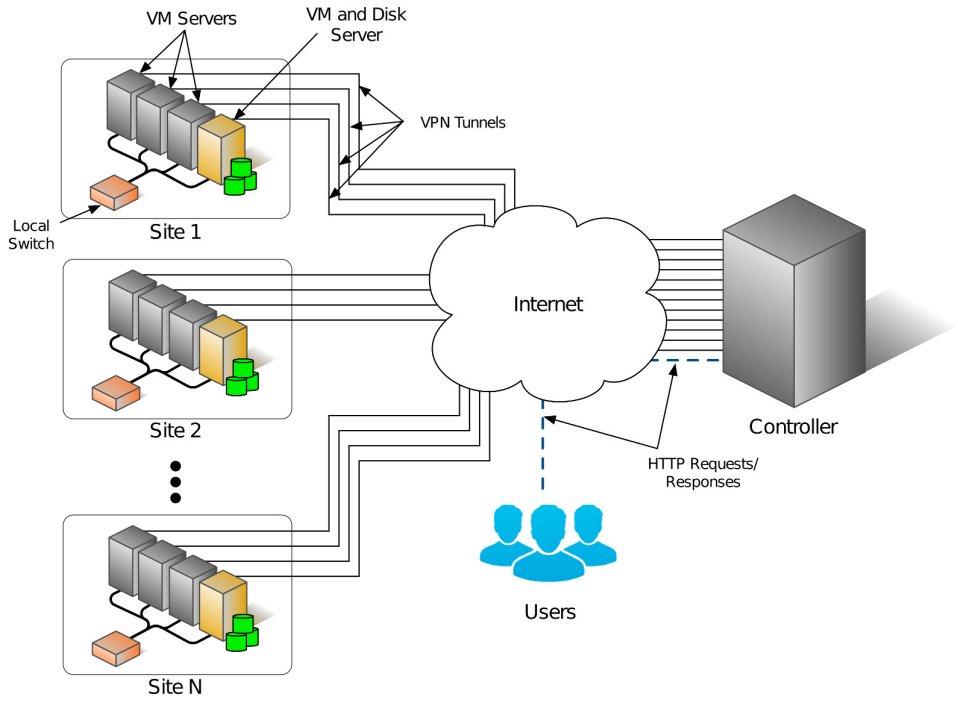


Figure 1: PID collaborative cloud architecture.

As shown in Figure 1, each site also has a special VM Server called VM and Disk Server. In addition to all functions performed by the VM Server, the VM and Disk Server also provides storage to all VMs in the

site. Storage is performed by NFS (Network File System) and volume services. Each site in this architecture must have at least one VM and Disk Server. We decided to centralize the storage of all VMs in a single server to yield seamless live VM migration inside each site, as detailed in Section 3.2. All servers in a site are connected using a local switch.

The architecture of Figure 1 is implemented using the OpenStack orchestrator. OpenStack has a modular architecture composed of components, following a "divide-and-conquer" approach. Each component deals with a specific task and defines a set of APIs for operation and development. Cloud services are then provided for users from the integration of these multiple OpenStack components, which may even include external services, such as the Amazon S3 storage. The organization into components is transparent, simplifying cloud management and utilization. Note that the modular architecture contrasts with other cloud orchestrators such as CloudStack, which uses a monolithic architecture. We adopt the modular approach of OpenStack since it makes the development easier. Table 1 describes the main OpenStack components employed in the PID cloud, called projects by the OpenStack community. Moreover, this table indicates where each component is located in the architecture of Figure 1. Note that some components are employed by different machine types. This happens because some components are composed of different services. For example, Nova has an API service hosted by the Controller, and a provision one hosted in VM Servers and VM and Disk Servers. Figures 2, 3 and 4 show, respectively, the services running in the Controller, in VM Servers and in VM and Disk Servers. In these figures, each service is represented as an ellipse. Each one of these services is a part of the components described in Table 1. Horizon and Keystone are not subdivided into small services and are represented in the figure with rectangles. Figure 2 shows that the Controller hosts all API services and the schedulers. The communication between the services of Figures 2, 3 and 4 is detailed in Section 3.6. In addition, Appendix A describes each one of the employed OpenStack services.

3.1. VM Instantiation

The default OpenStack operation assumes a centralized data center. In such model, a user requests a VM and the scheduler chooses from a pool of hosts one that fulfills the VM specifications (such as the amount of RAM, disk, and the number of CPU cores). With a distributed data center, a new set of requirements arises. After a given user request, VMs can be instantiated in batches or can be individually created depending on the application. Users may want to improve their service reliability by spreading VMs across different sites or may want to improve application performance by running multiple VMs together at the same site, so as they can efficiently share a disk or exchange data. To meet such contrasting requirements, we have extrapolated the OpenStack concept of Availability Zones and modified the VM scheduler.

OpenStack originally uses Availability Zones for logical segregation of physical nodes within the same centralized data center. This segregation usually represents physical properties, such as nodes sharing a rack. PID, however, uses Availability Zones to separate geographically distinct sites. The main advantage of this approach is the ability to create integrated sites without losing track of their physical locations. The OpenStack default scheduler places VMs in a particular Availability Zone only if the user explicitly

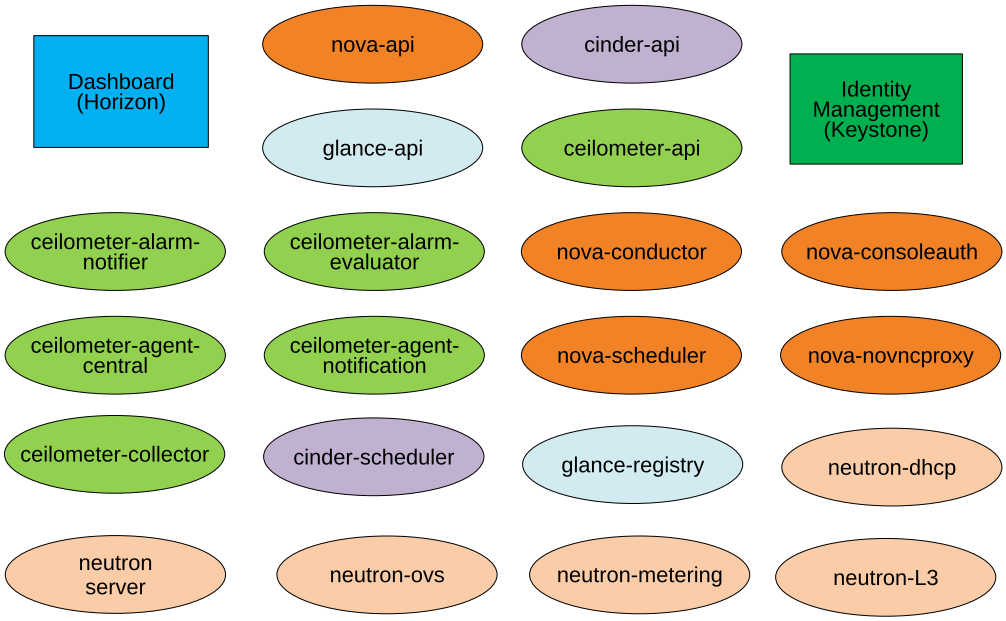


Figure 2: Components installed in the Controller.

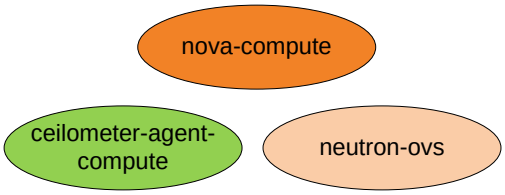


Figure 3: Components installed in the VM Servers.

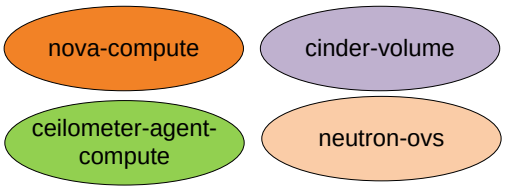


Figure 4: Components installed in the VM and Disk Servers.

Table 1: Summary of the main components of OpenStack.

Components (***Project name***)	**Description**	**Location in PID**
Compute (*Nova*)	Controls the hypervisor of each VM Server. Hence, it is in charge of providing on-demand computing resources, by provisioning and managing virtual machines. This module controls states and respective transitions of a virtual machine. Nova is designed to scale horizontally on standard hardware.	Controller; VM Server; VM and Disk Server
Dashboard (*Horizon*)	Provides a graphical interface for administrators and users to access cloud resources, taking into account pre-defined policies. In addition, the interface allows users to interact with other OpenStack tools, such as monitoring.	Controller
Identity Service (*Keystone*)	Provides authentication services for the cloud by maintaining a central directory of users. These users are mapped into services they can access according to pre-defined policies. The identity service can provide support to other authentication methods, including federated systems.	Controller
Block Storage (*Cinder*)	Provides persistent block level storage, called volumes, for use with virtual machines. With this component, virtual machines can have block devices attached and detached by the cloud administrator and by users according to their storage needs. It can be seen as a virtual hard drive.	Controller; VM and Disk Server
Image Service (*Glance*)	Provides discovery, registration, and delivery services for virtual machine images. Images are pre-installed systems that can be provided to or built by users.	Controller
Network (*Neutron*)	Provides network and IP address management, including customized networking models for different applications or user profiles. It provides Internet access to virtual machines using public or private IP addresses that can be dynamically or statically assigned. In addition, users can create their own virtual networks composed of multiple virtual machines running on different LANs. Other features include VLANs for network isolation and SDN for higher-level services.	Controller; VM Server; VM and Disk Server
Telemetry (*Ceilometer*)	Provides measurement services of cloud components for billing systems. For instance, billing can be established based on the amount of CPU or network consumed by each user. Data are collected by either notification or polling, *i.e.*, existing services can send notifications or information can be obtained by polling the infrastructure.	Controller; VM Server; VM and Disk Server

determines it, *i.e.*, a zone cannot be automatically chosen. To solve this issue, we introduce a zone scheduler. The zone scheduler is analogous to the traditional host scheduler but chooses zones instead of hosts. It is composed of three phases: filtering, weighting, and distribution. During the filtering phase, the scheduler iterates over all zones, evaluating each one against a set of filters. During the weighting phase, the remaining zones are ranked using some criteria. Finally, in the distribution phase, the scheduler spreads VMs among

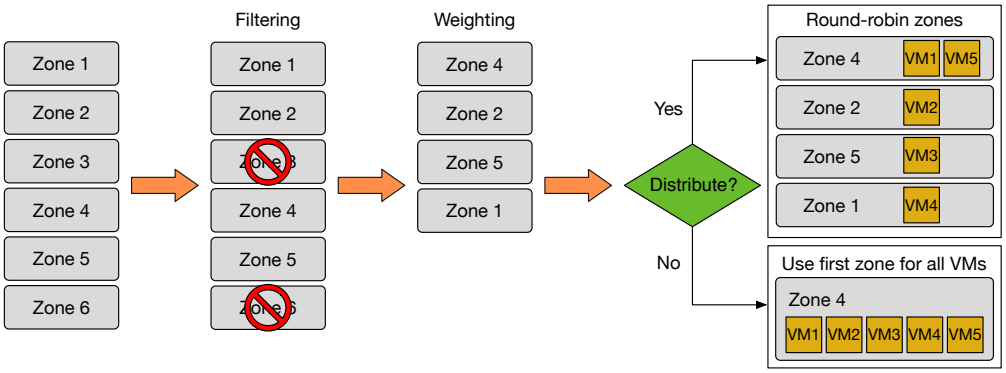


Figure 5: Zone scheduler being used to schedule five VMs across six zones.

the ranked zones, in a round-robin fashion, or it allocates all VMs in the first zone, according to a distribution property that is configured by the user. Figure 5 shows an example of the scheduling of five VMs across six zones. In the filtering phase, zones 3 and 6 do not pass one of the filters (*e.g.*, zones do not have enough capacity to accommodate the requested VMs) and are eliminated. In the weighting phase, zones are ranked, making zone 4 the first (*e.g.*, the zone with most available resources). In the distribution phase, VMs are either distributed among all zones or scheduled only in zone 4. After the zone scheduler selects a zone for each VM, it calls the regular OpenStack scheduler to select a host within this zone. In addition to the distribution property, users can set other properties to change the behavior of filters and weights. For example, we have implemented a simple filter (`SameZoneFilter`) that allows users to specify the zone where they want to place their VMs. It only activates if a user specifies a zone using the `same_zone` property. In such case, it rejects all zones but the one specified. Otherwise, if the `same_zone` property is not set, it accepts all zones. Other application-specific filters may be added to consider other policies, such as latency requirements and communication patterns between VMs.

Besides Availability Zones, OpenStack employs other methods to support geo-distributed clouds, such as Cells and Regions [29]. Nevertheless, these methods segregate the cloud, namely, each site runs as a complete cloud deployment. Since a site in PID usually has a small number of servers, it is not interesting to run all the cloud components on every site. As a consequence, we preferred to employ Availability Zones, which permits the main cloud components to run in a centralized controller. Even with a logically centralized controller, as employed in PID, OpenStack can have multiple physical controllers. Hence, the Controller of Figure 1 can be replaced by multiple machines without changing our current architecture.

3.2. VM Migration

VM migration has been widely used for tasks such as scheduled maintenance and server consolidation. In our architecture, we consider two types of migration: intra-site (hosts migrate within the same site) and inter-site (hosts migrate across different sites). Both migrations can be live, which means that there is no need to shut down a VM to migrate it. Even though migrated VMs keep their states, there may be a period of

unavailability during the data transference between the VM Servers. In this context, the inter-site migration is more affected since a disk copy may take more time to finish as data is transferred via the Internet. The intra-site migration, on the other hand, may work seamlessly as servers in the same site share the same local storage. Currently, our VM migration is performed manually, but cloud administrators can design schemes to automatically migrate VMs and provide optimal VM placement [8].

3.3. Network

In a centralized cloud, the VMs of a given user communicate using private IP addresses. This can be easily implemented since the VMs are on the same local network. By contrast, PID is geo-distributed, which may lead to communications involving VMs from different sites. One simple approach is to assign public IP addresses to these VMs, allowing them to be accessible over the Internet. As public IP addresses are scarce, this would severely limit the maximum number of VMs supported by the cloud. Our solution to this problem is to use the Neutron component, which, as described in Table 1, provides network services to OpenStack. Using Neutron, we can assign private IP addresses to the VMs and make them accessible to the entire cloud, including inter-site communication. To connect the VMs, Neutron employs virtual networks. These networks can be composed of virtual routers, which forward packets between different subnetworks; and VLANs, which allow logical division of the physical network [30].

Neutron builds virtual networks using bridges and tunnels created by Open vSwitch [31] and OpenFlow [32]. Basically, the VMs of a given site are connected using tunnels created in the local network whereas the VMs of different sites are connected using VXLAN (Virtual eXtensible LAN) tunnels passing through the Controller. Consequently, all packets for inter-site communications must be handled by the Controller, which simplifies the network management at cost of traffic convergence at a single point. Another advantage of Neutron is the virtualization of the IP address space. This feature is based on Linux namespaces, which allow different users to reuse the same IP addresses for their VMs. Finally, Neutron is integrated with Horizon, allowing network configuration through the graphical interface.

3.4. Administrative Hierarchy

In PID, the administration is organized in two levels: Global and Local. The Global Administrator (*i.e.*, the cloud administrator already used in OpenStack, named `admin`) is responsible for managing the whole IaaS cloud, whereas Local Administrators deal only with management procedures local to a site. For instance, a Local Administrator can only migrate a VM within its own site (intra-site migration); the Global Administrator, on the other hand, combines Local Administrator capabilities to specific ones such as global migration (inter-site migration), user creation (addition of new users, user groups, or Local Administrators that use the infrastructure), and quota assignment (definition of maximum limits for resource utilization, *e.g.*, the amount of memory and the number of CPU cores assigned for users or groups).

The access control in OpenStack is implemented, by default, using RBAC (Role Based Access Control). In RBAC, each user type is defined by a set of roles, which identifies the functions that a user of a given

type can execute. In PID, end-users have only the `member` role assigned, which allows VM creation and visualization. Local Administrators, on the other hand, have the `member` role assigned as well as two more roles: the `adm_local` role, which grants access to the local administration web interface, and the `adm_site` role, which allows the execution of management tasks in machines within a specific site. Figure 6 shows the set of roles for each user type in a cloud with two sites: UFRJ and UERJ. The Local Administrator from UFRJ has the `member+adm_local+adm_UFRJ` roles. The Global Administrator has specific roles, as well as all Local Administrator and end-user roles.

In the physical infrastructure level, we control new coming VM Servers by controlling their access to our VPN. This VPN is configured by using keys issued by a Certificate Authority, managed by the Global Administrator. For each VM Server, we issue a different pair of public and private keys. Hence, when a Local Administrator wants to add a new VM Server, the Global Administrator creates a new pair of keys for this server. Currently, for security reasons, this procedure is performed manually by the Global Administrator.

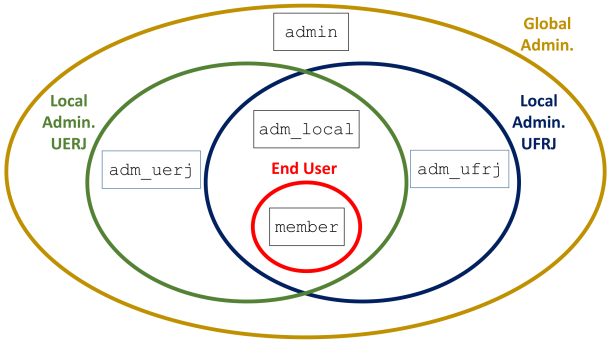


Figure 6: RBAC example in PID.

It is also important to discuss aspects of security and privacy issues in PID cloud. These aspects are similar to those faced by public clouds, where users must trust the cloud administrator, which provides low-level security and privacy mechanisms. However, given the flexibility of IaaS clouds, some high-level mechanisms, such as data cryptography, should be implemented by users [33]. Users are also responsible for the security of their applications and operating systems. Different from public clouds, however, PID users may choose to place privacy- or security-sensitive VMs in their own sites.

3.5. Federated Login

In a collaborative cloud environment, an authenticated user from any of the collaborating institutions should have access to every cloud resource designated to their home institution. This means that PID has to accept user authentication from every collaborating institution. One naive solution is to centralize the authentication to a given entity, which creates new users and defines policies to each one based on user institution. In this scenario, every institution would have to report changes in their authentication policies and user database, which would have to be accommodated by the central entity. This can be expensive and error-prone. Alternatively, institutions can be autonomous regarding the identity management. A given

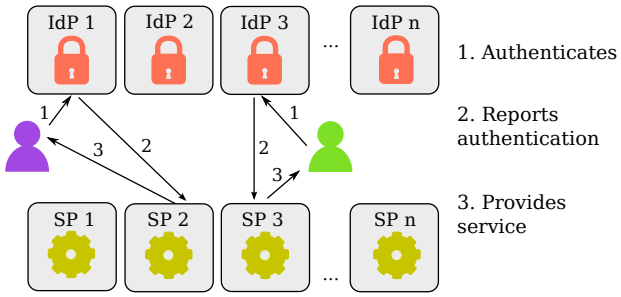


Figure 7: Federated login example.

institution should be free to control the access of its own users and to define the amount of resources available to each one. This is performed using an identity federation [34].

In a federation, a Service Provider (SP) offers services to users who are authenticated by an Identity Provider (IdP). Figure 7 shows a federated login example, where multiple IdPs provide authentication to multiple SPs. After requesting a given service to an SP, the user is then redirected to a selected IdP and requests authorization to it (step 1). After a successful login, the IdP sends user identity attributes (*e.g.*, email, id, name) back to the SP (step 2). Based on these attributes, the SP provides the service to the user (step 3). The PID cloud is thus one of the SPs in the federation. Collaborating institutions can designate one or more IdPs of their own trust to authenticate their users.

The OpenStack platform provides support to federated login, but an additional framework is needed to enable the communication between OpenStack and an IdP. This framework must be able to receive from the IdP the attributes of each incoming user and insert these attributes into Keystone. To this end, the PID cloud uses a federation identity solution called Shibboleth [35].

Most use cases in PID require persistent user operations. For instance, if a user launches VMs and then logs out, it is important that, on a new access from the same user, they can retrieve and access these previously instantiated VMs. Hence, a federated user must also be represented by a local user in the Keystone database and, on every login, they must be attached to their respective local user in OpenStack. This procedure, implemented in PID, is called mapping. After the federated user is mapped into an internal user using the attributes provided by Shibboleth, Keystone can take such attributes and generate an access token to the user. This token grants access to OpenStack services, either through the API or the web interface.

3.6. Communication between Services and Modules

As already mentioned, each OpenStack project is composed of different modules, that communicate with each other to offer the designated services. For example, Nova has an API module (*i.e.*, `nova.api`), installed on the Controller machine of PID architecture. This module communicates with VM provisioning modules (*i.e.*, `nova.compute`) installed on the servers. The modules of the same OpenStack project communicate with each other using an AMQP (Advanced Message Queuing Protocol) [36] message queue, implemented by the RabbitMQ middleware. Hence, a given module sends messages to a RabbitMQ queue installed in

Table 2: 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

the Controller. These messages remain in the queue until they are consumed by other modules. Modules from different projects communicate through the APIs hosted by the Controller. Each project has its own API offering its services to other projects and external applications. Finally, OpenStack provides a database that is used by different projects. The communication between each project and the database is performed using MySQL commands. Consequently, the Controller centralizes all MySQL traffic. As can be noted, the Controller machine plays a central role in the communication of PID cloud. Hence, in the next section, we analyze the impact of this centralization.

4. Experimental Analysis

This section evaluates the impact of key architectural choices on the performance of the proposed collaborative cloud. The first group of experiments aims at evaluating the amount of control traffic produced in the architecture. Since collaborative infrastructures may be distributed among different sites interconnected via low-capacity networks, the traffic generated by the control messages can be prohibitive as the number of sites and servers grow. In the PID architecture, the Controller plays a central role since it manages the entire infrastructure. As a consequence, we focus on the control traffic exchanged with it. In the first experiments, we use a controlled environment with colocated machines.

Then, in the second group of experiments we analyze the impact of control traffic produced in the current deployment of the PID collaborative cloud, presented later in this section. Moreover, we apply the results obtained in the controlled environment of the first experiments to predict the behavior of the system on WAN deployments, using real Research and Education Network (REN) topologies as input. Finally, in the third group of experiments, we analyze the impact of geo-distribution in the data traffic between VMs.

Table 2 summarizes the hardware configuration of the Controller and the Server machines used in our testbed. For all experiments, except those using samples along the time, we compute averages and confidence intervals of 95%, represented by vertical error bars in the graphs. Some bars do not appear in the figures because they are too small. Next, we present the different experiments conducted.

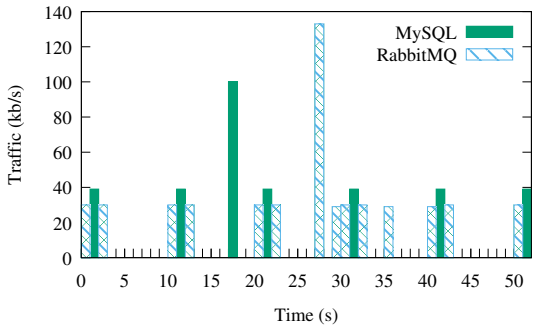


Figure 8: Traffic sample between a Server and the Controller along the time.

4.1. Control Traffic Analysis

In this section, we analyze the amount of control traffic needed in the operation of PID cloud. We focus on the traffic between the Controller and the Servers. Since all VM Servers talk to the Controller, this is the network bottleneck in terms of control traffic. To evaluate the control traffic, we employ a topology similar to that of Figure 1. We collect the traffic from the VPN interface of the Controller. This is the network interface that is connected to the WAN in our real deployment, presented later.

Referring to the PID architecture, in the present experiments we only analyze the traffic exchanged with VM and Disk Server machines. VM and Disk Servers provide an upper bound in terms of control traffic since they provide disk server functionalities in addition to the VM server functionalities provided by VM Servers. More precisely, Cinder, which is only present in VM and Disk Servers, communicates with the Controller via the VPN tunnel. For sake of simplicity, in the rest of the evaluation, we simply use the term "Server" for the used VM and Disk Servers.

In the experiments of this section, we use one to four Servers, whose configurations are shown in Table 2. With four Servers, we emulate a cloud with four sites for the purposes of our analysis. Nonetheless, to guarantee control over the experimental scenario and to eliminate external factors, all sites are located in the same physical location. Also, these sites are connected by VPN tunnels established over a local network, instead of the Internet.

4.1.1. Varying the number of Servers

Our first experiment measures the traffic between an "idle" (with no VMs instantiated) Server and the Controller. The goal of this first experiment is to analyze the contributions of different components to the control traffic. Subsequently, we analyze the control traffic produced by n Servers.

In the first experiments, we use Server 1 connected to the Controller (Table 2). Figure 8 shows, as a function of time elapsed, the traffic generated by Nova modules exchanging messages between each other using RabbitMQ, and the traffic produced by the database transactions performed by Cinder. The Nova modules installed on the Server periodically send updates to the Nova modules running at the Controller, using the message queue system of RabbitMQ. These updates are then inserted in the database installed on

Table 3: RabbitMQ and MySQL traffic generated from a single Server to the Controller.

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

the Controller. Similarly, a Cinder module running on the Server periodically sends updates to the Controller. Unlike Nova, however, this Cinder module directly communicates with the database using MySQL.

Figure 8 shows that RabbitMQ traffic is produced by the Server every 10 s, to update service states. Between 20 and 30 s, there is a traffic burst that corresponds to the update of the list of server instances. That update is performed every 60 s. Similarly, the Cinder module in the Server reports its state every 10 s, producing the MySQL traffic of the figure. Between 10 and 20 s there is a burst of MySQL traffic, which is a consequence of the updates concerning information of volumes. Those are also performed every 60 s. Table 3 shows the contribution of these two communication types to the average network traffic, computed during 60 s. The results are an average of 25 experimental runs, and the table does not contain confidence intervals which were negligible. We can observe that most of the traffic is RabbitMQ, *i.e.*, traffic generated by Nova. Moreover, it is possible to verify that the total measured traffic is around 15 kb/s.

In the second set of experiments, we vary the number of Servers connected to the Controller to evaluate the additional control traffic produced. We use the Controller and up to four of the Servers of Table 2. For each added Server, we measure the traffic during 60 s. The whole procedure is repeated 10 times. Figure 9 presents the traffic related to RabbitMQ (orange squares), to MySQL (blue triangles), and the total traffic (red dots), as a function of the number of Servers. The network traffic grows linearly with the number of Servers. Since there is a clear linear behavior, we perform a linear regression of the total network traffic, which results in the solid line plotted in Figure 9. The R^2 value indicates the quality of the fit of the linear regression. R^2 varies from 0 to 1, where 1 means a perfect fit. For the total network traffic, $R^2 = 0.9966$. Our results of R^2 very close to 1, along with the plot in Figure 9, show that traffic follows a linear relationship with the number of servers. Thus, we can safely assume that each server adds, on average, 15 kb/s of control traffic. This result matches our previous experiment using one Server (Table 3). As a consequence, using extrapolation, 1.5 Mb/s of control traffic would flow through the Controller interface assuming a scenario with one Controller and 100 Servers.

4.1.2. Varying the number of VMs per Server

In this experiment, we evaluate the influence of the number of VM instances in a single Server (Server 1) on the network traffic. Clearly, increasing the number of VM instances and volumes implies a larger amount of traffic sent to the Controller in order to update the Nova and Cinder databases. Thus, to accomplish our goal, we measure the traffic sent just after a well-succeeded VM instantiation, not taking into account

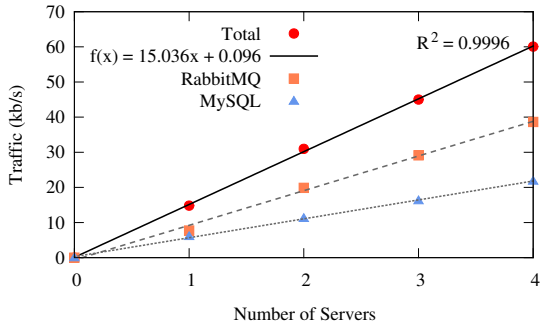


Figure 9: Network traffic for an increasing number of Servers.

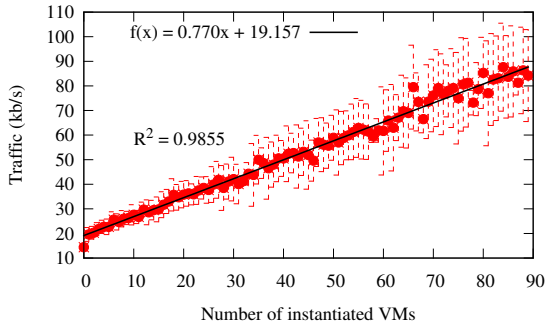


Figure 10: Network traffic for an increasing number of VMs.

the traffic generated by the instantiation procedure. We vary the number of Servers, measuring the traffic for 60 s. In this experiment, each VM is set with 64 GB of RAM and one virtual CPU using the CirrOS operating system. The remaining experiments in this paper use the same VM configuration. It is worth mentioning that the VM configuration does not affect our measurement since control messages consist of generic information such as VM states and VM IDs that are not influenced by the VM configuration.

Results depicted by Figure 10 indicate a linear behavior when we increase the number of instantiated VMs. Applying a linear regression on the results produces a function with R^2 equals to 0.9855. Therefore, based on the linear function presented in the figure, we estimate that each VM is responsible for 0.77 kb/s of additional traffic. Hence, considering a scenario with 100 servers hosting 15 VMs each, which represents 1,500 VMs, the total traffic would be 1.155 Mb/s. If we sum up the traffic generated by all VMs to the traffic from the Servers, analyzed in Section 4.1.1, the total control traffic would be approximately 2.7 Mb/s.

4.1.3. Creating and deleting multiple VMs

We now analyze the traffic behavior during VM creation and deletion. OpenStack provides two ways to instantiate a VM. The first one uses an image stored in an ephemeral disk unit, which stores data as long as the associated VM exists, whereas the second one uses an image stored in a Cinder volume, which offers persistent storage. When a user requests an initialization of a VM without volume for the first time, Glance sends an image to the Server through the VPN tunnel. Then this image is locally stored at the Server and

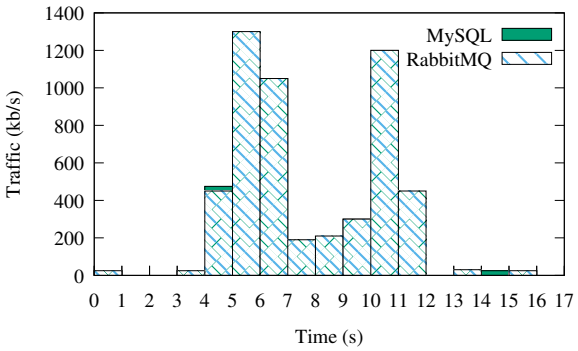


Figure 11: Traffic generated when creating and deleting one VM.

copied to the ephemeral disk unit, which is also located at the Server. If a new VM instance with the same image is created in the Server, the image will not be restored through the network. In the case of instances with initialization from volume, the following procedures are executed: An empty volume is created, the image is transferred to the Server, and then the image is copied to the volume managed by Cinder. Unlike Nova, Cinder does not use cache [37]. As a consequence, whenever a new instance with initialization from volume is created, the image is transferred through the network.

Typically, a VM in an IaaS cloud is created from an ephemeral disk (*i.e.*, based on image), due to the high traffic generated when creating VMs using volumes. Therefore, we analyze network traffic during consecutive creation and destruction of VMs based on image, in order to evaluate only the generated control messages. Traffic is evaluated after a VM image is inserted into the cache of Glance. In this way, the experiments do not depend on the configuration of the VMs.

Figure 11 shows the impact on the network traffic of one user request for creating one VM instance and, at the end of this process, for destructing the VM. Note that the maximum traffic is approximately 1.3 Mb/s during creation (between 5 and 6 s) and 1.2 Mb/s for destruction (between 10 and 11 s).

To better analyze such traffic, we also use an OpenStack benchmark named Rally [38] that performs scalability and performance tests on real deployments. This benchmark offers predefined scenarios, such as one related to VM creation requests, and has been employed in related work, such as [20, 21]. In the following experiment, we employ a scenario named `boot-and-delete.json`, which is one of the simplest scenarios available in Rally. We vary two different parameters: the number of VMs to be sequentially created in a cloud and the number of parallel requests sent to the Controller. If the number of parallel requests is one, Rally generates one request for the Controller to create one VM, waits for its creation, and then requests its destruction. The same procedure is repeated until the specified number of VMs is reached. If the number of parallel requests is different from one, emulating the existence of concurrent users in the cloud, Rally simultaneously sends that number of requests to the Controller. When one of these VMs is deleted, a new creation request is sent to maintain the same number of parallel requests. In our experiments, we perform requests to the Controller to create VMs in Server 1. We employ different numbers of VMs to

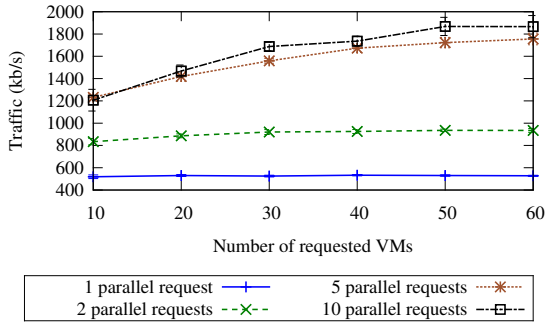


Figure 12: Traffic generated when creating and destroying VMs in parallel.

be created (from 10 to 60) and different numbers of requests in parallel (from 1 to 10). The experiment was performed 10 times for each number of requested VMs and parallel requests.

Figure 12 shows that, even for a small number of concurrent users, the control traffic is not negligible. For example, 10 users requesting one VM each (10 requested VMs in total) generate about 1.2 Mb/s. This can lead to a high utilization in low capacity networks, especially if we also take into account the traffic periodically generated by control messages, as seen in Sections 4.1.1 and 4.1.2. Figure 12 also shows that the average traffic approximately doubles when the number of parallel requests is increased from one to two. Nevertheless, the relationship between the number of parallel requests and the amount of control traffic is not linear. For example, traffic generated for five parallel requests is not five times greater than that of one request. Moreover, from five to ten parallel requests, the traffic remains approximately the same since our Server does not support the creation of all the VMs at the same time. Hence, when creating and destroying VMs in parallel, the traffic tends to an upper bound due to hardware limitations of the VM Servers and not due to network bottlenecks. This is true since our local network is a Gigabit Ethernet, and the generated traffic is in the order of Mb/s.

4.2. Traffic Impact in the Current Deployment and in real WANs

4.2.1. Applying the results in the current deployment

Currently, the PID cloud is deployed in three universities, in Rio de Janeiro, Brazil: UFRJ, UERJ, and UFF.

To analyze to which extent the control traffic limits the VM hosting capacity in the cloud, we measure the TCP throughput between each site and the Controller by using the iPerf tool [39]. To perform the experiment, we instantiate a VM in UFF and another in UERJ to individually send TCP traffic to a VM in UFRJ for five minutes. We perform several trials of this experiment. Using a confidence level of 95%, the throughput observed between UFRJ and UFF is 370.56 ± 31.03 Mb/s and between UFRJ and UERJ is 21.61 ± 0.65 Mb/s. Considering an example where each Server instantiates 15 VMs, a Server generates a traffic of 26.55 kb/s using the results of Section 4.1 (*i.e.*, 15 kb/s for the server and 11.55 kb/s for the 15 VMs). Hence, if we are conservative and dedicate 1% of the measured link capacity for the control traffic,

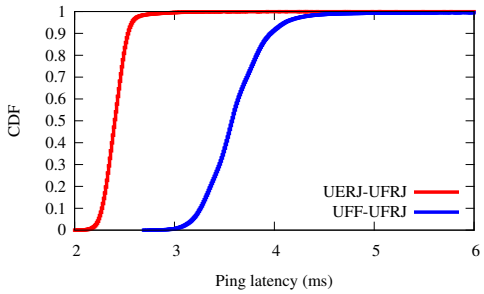
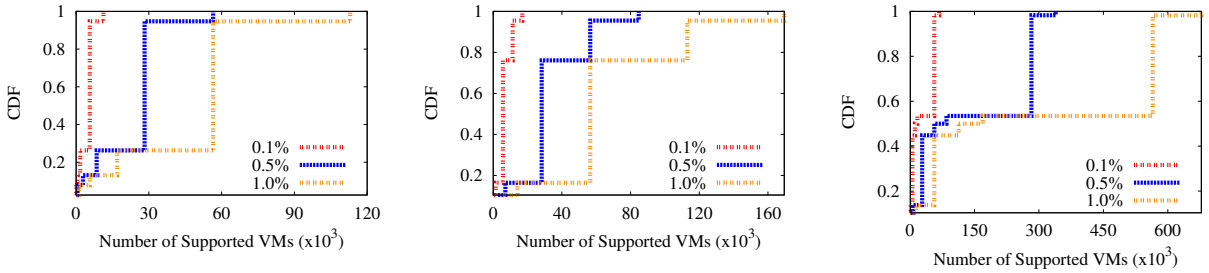


Figure 13: Latency between each site and the Controller.

we can have approximately 8 servers (*i.e.*, 120 VMs) in UERJ and 139 servers (*i.e.*, 2,085 VMs) in UFF. Even in UERJ, where the network capacity is smaller, the maximum number of VMs is well suited for a droplet, while in UFF we can host a medium-sized site. The considerable difference between the capacity of UERJ and UFF is probably caused by network policies defined in each institution since they are interconnected through the same Gigabit MAN (Metropolitan Area Network).

One factor that might impact control messages is the latency between cloud sites and the Controller. Hence, we analyze the latency in our cloud by performing pings between a VM hosted in a remote site (*i.e.*, UFF and UERJ) and a VM hosted in the same site where the Controller is located (*i.e.*, UFRJ). Figure 13 shows the CDF (Cumulative Distribution Function) of ping samples collected over a period of 24 hours. It shows that latency values are low, being at levels that do not cause severe impact on interactive applications [40]. Obviously, if we install sites at locations far from the Controller, such as in another continent, this latency will increase due to the propagation delay. To circumvent such problem, one might install an additional Controller near distant sites, which is a subject of our future work.

4.2.2. Applying the results in WANs



(a) RNP. (b) RENATER. (c) GÉANT.

Figure 14: Supported VMs in each link considering different fractions of reserved bandwidth.

We now evaluate the impact of the control traffic in real WANs. More specifically, we investigate the maximum number of VMs we are able to instantiate when we reserve a small fraction of the links ($0.1\% \sim 1\%$) to control traffic. We apply our results from the previous sections to three different network topologies:

RNP [41], from Brazil; RENATER [42], from France; and GÉANT [43], from Europe. These Research and Education Networks span wide geographical areas and have links with different capacities. In our analysis, we evaluate the scenarios where every flow comprising the Controller is bottlenecked by a particular link, *i.e.*, we change the Controller location in the network so that every link becomes the last hop to the Controller.

Figure 14 shows the Cumulative Distribution Functions for the number of supported VMs using every link in the network as the bottleneck, for different fractions of reserved bandwidth (0.1%, 0.5%, and 1%). We can note that the curves are different for each network since they have different link capacities. In all networks, the Controller location has a huge influence on the number of supported VMs. As an example consider the RNP network with 1% reserved bandwidth, while 13.15% of the links uphold less than 5,600 VMs, there are links that support up to 56,000 VMs. This result shows the importance of carefully placing Controllers when designing a collaborative cloud. Also, note that if the amount of traffic is too high one should consider the use of multiple Controllers in geographically distinct locations. Even though our evaluation is limited to the control traffic, we show that this traffic can also impact the number of supported VMs. However, when designing a cloud, one must account for the traffic generated from applications and cloud operations (*e.g.*, VM creation).

4.3. Data Traffic Analysis

In this experiment, we analyze how the data traffic between VMs is affected by the geo-distribution and the virtual network options available in Neutron. We evaluate a scenario where the communicating VMs belong to the same tenant and another one where different tenants communicate with each other. A tenant, named "project"[2] in OpenStack, is a group of users that share the same VMs. In our case, for instance, a tenant can be a group of users performing the same scientific experiment. VMs of the same tenant can have L2 connectivity between each other if they are in the same virtual network. Each tenant, by default, has its own set of virtual networks.

VMs belonging to different tenants can communicate using two options. The first one consists of using floating IP addresses. This type of IP address can be allocated to VMs in OpenStack, making them accessible from external networks. In other words, a machine that does not belong to the same virtual network of a given VM can reach this VM by using this IP address. Floating IP addresses can be public IPs, which make the VMs accessible through the Internet, or private ones, which make them accessible only inside the OpenStack cloud. Neutron employs virtual routers to forward the traffic associated with floating IPs, using the `neutron-L3` module in the Controller. The second option is to create a shared virtual network. This type of network can be shared among tenants. Each tenant can attach VMs to this network and thus VMs of different tenants can have L2 connectivity between each other.

We evaluate a scenario where a client VM sends TCP traffic to a server VM with iPerf. Simultaneously to the TCP flow, we send ping probes from the client VM to the server VM, to measure the communication

[2]The name "project" is also employed by the OpenStack community to refer to the components described in Table 1 .

Table 4: Data traffic between VMs in the same physical machine.

Traffic Type	**Throughput (Mb/s)**	**Ping Latency (ms)**
Same Tenant	26331.94 ± 70.86	0.76 ± 0.01
Different Tenants - Shared Network	26198.86 ± 124.37	0.76 ± 0.01
Different Tenants - Floating IP	171.00 ± 0.84	5.33 ± 0.05

latency. We perform experiments when both VMs belong to the same tenant and when they belong to different tenants. In this last case, we analyze the communication between VMs when the traffic is sent using floating IPs and when they share the same virtual network. All results in this section are obtained as average values for 10 experimental samples, with a 95% confidence level.

In the first experiment, the client VM and the server VM are on the same physical machine, which is the Server 1 of Table 2. The Controller is the same as specified in Table 2. Server 1 and the Controller are on the same local network, but running a VPN to use the same architecture of Figure 1. Table 4 shows the throughput and ping latency measured for each case. The results show that when VMs are on the same tenant, the traffic is forwarded inside the physical machine, achieving high throughput and low latency. This happens because Neutron is aware of each VM location and configures the Open vSwitch module of Server 1 (i.e., `neutron-ovs` of Figure 3) to forward packets inside this physical machine. The same situation happens when different tenants communicate using a shared network, as can be noticed by the high throughput and low latency values. However, when VMs communicate using floating IPs, the traffic must pass through a virtual router in the Controller. This happens because the floating IP is in a different subnetwork than the one where the VMs connected. As the traffic passes through the Controller, Table 4 shows that the throughput is limited by the network between Server 1 and the Controller, achieving a throughput much lower than the case of shared networks and with a higher latency. We can overcome this situation by installing and configuring `neutron-L3` in VM Servers. However, this adds complexity to the VM Server deployment since routing mechanisms need to be configured. To preserve the simplicity of our architecture, our design decision keeps `neutron-L3` only in the Controller. Consequently, we recommend our users to use shared networks to communicate with different tenants.

Afterward, we evaluate the performance when the client VM and the server VM are on different physical machines, for the three network configurations analyzed in the previous sections. The client runs in Server 1 of Table 4, while the server runs in Server 3. To isolate our experiments from external factors, all physical machines are on the same local network. In addition, to emulate a geo-distributed scenario, we induce different values of latency in the physical network. This induction is performed by using Linux TC (Traffic Control) [44] in Server 1. Figure 15 shows the throughput obtained for the three network configurations when the induced latency increases. The confidence intervals are very small and thus are barely visible in the figure. We suppress the ping latency results since, in our experiments, the induced latency dominates

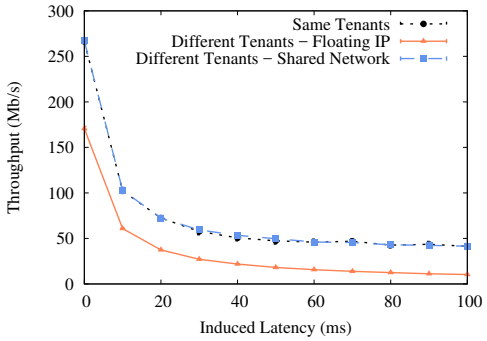


Figure 15: Data traffic between VMs in different physical machines.

the value of this metric. In all the three cases, the results show that the throughput is proportional to $\frac{1}{RTT}$, where RTT is the round-trip time between the VMs, which is mainly affected by the induced latency. This behavior is expected due to TCP congestion control [45]. Hence, in our scenario, the additional latency in a geo-distributed cloud has little or no impact on Neutron, and the TCP traffic presents the same behavior as in traditional networks. The results of Figure 15 also show that the communication inside the same tenant has the same performance as when different tenants share a virtual network. However, the throughput drops when a floating IP is employed, showing that the performance drop of the floating IP is caused by the virtual router implementation.

5. Conclusions and Future Work

This paper has introduced PID (Platform for Iaas Distribution), a collaborative cloud designed for scientific computing. Unlike other architectures, PID is built upon dedicated servers contributed by participant institutions. The number of servers assigned and the amount of resources needed at each institution can be kept low, motivating the term droplets. The main goal is to reduce costs of IaaS services by using already existing and idle computational resources from institutions. Our implementation of PID is based on the OpenStack cloud orchestrator, chosen due to its modular architecture, which largely simplifies customization.

We have performed experimental tests to analyze the control traffic related to the PID Controller. Results have shown that each VM server adds, on average, 15 kb/s of control traffic to the network; while each idle VM can contribute with 0.77 kb/s. In addition, we have observed in our experiments an upper-bound on the number of simultaneous VMs creation and deletion, limiting the amount of control traffic generated in 2 Mb/s. We have also noticed that, in our prototype, the latency between each site and the Controller does not affect interactive applications. Finally, considering real WAN topologies, we have noticed that the amount of traffic generated to or from the Controller is not enough to become a clear bottleneck, except at networks with really low capacities. We then conclude that the PID cloud scales with the number of Servers and virtual machines, even considering a central Controller. This centralization might raise other scalability

issues, such as the need of processing and memory capacity to control the cloud as the number of sites increases. OpenStack solves these issues by allowing multiple Controllers in the cloud. We can thus employ multiple physical Controllers to act as a single logical Controller, which does not change our architecture. Alternatively, we can employ a hierarchy of Controllers, using cloud federation or the concept of "cloud of clouds", stated before in this article.

In addition to the control traffic analysis, we have evaluated the data traffic performance in our architecture. Results show that the additional latency expected in a geo-distributed cloud has little or no impact on Neutron. Consequently, the TCP traffic behaves the same as in traditional networks. In this data traffic experiment, we also show that, depending on the network configuration, the traffic between different tenants has a throughput drop as compared to the communication inside the same tenant. Nevertheless, when employing shared networks in the multi-tenant case, our results show that the throughput is equal to the case where a single tenant is used.

As future work, we plan to extend our measurements and evaluate PID in a production environment. In addition, we look forward to providing a full analysis of our cloud, including the evaluation of economic aspects (*e.g.*, measuring how much money we can save using PID), elasticity (*e.g.*, measuring to which extent the cloud is able to support an increasing demand of a given user), and other performance metrics, such as latency to access VMs and time to instantiate a VM.

Acknowledgments

This work was partly funded by FAPERJ, CAPES, CNPq, and grants #15/24494-8 and #15/24490-2, São Paulo Research Foundation (FAPESP). We would also like to thank Jean Philipe S. da Fonseca, Rafael M. da Costa, and Igor M. Moraes for their valuable help.

References

[1] AWS – Elastic Compute Cloud. 2017. Accessed in September 2017; URL `http://aws.amazon.com/ec2/`.

[2] Microsoft Azure. 2017. Accessed in September 2017; URL `http://azure.microsoft.com/`.

[3] Høimyr, N., Blomer, J., Buncic, P., Giovannozzi, M., Gonzalez, A., Harutyunyan, A., et al. BOINC service for volunteer cloud computing. In: International Conference on Computing in High Energy and Nuclear Physics. 2012, p. 1–6.

[4] PlaneLab. 2017. Accessed in September 2017; URL `http://www.planet-lab.org`.

[5] Maglaris, V., Papagianni, C., Androulidakis, G., Grammatikou, M., Grosso, P., Van Der Ham, J., et al. Toward a holistic federated future internet experimentation environment: the experience of NOVI research and experimentation. IEEE Communications Magazine 2015;53(7):136–144.

[6] Moraes, I.M., Mattos, D.M.F., Ferraz, L.H.G., Campista, M.E.M., Rubinstein, M.G., Costa, L.H.M.K., et al. FITS: A flexible virtual network testbed architecture. Computer Networks 2014;63:221–237.

[7] Mell, P., Grance, T.. The NIST definition of cloud computing. Tech. Rep.; 2011.

[8] Couto, R.S., Secci, S., Campista, M.E.M., Costa, L.H.M.K.. Network design requirements for disaster resilience in IaaS clouds. IEEE Communications Magazine 2014;53(10):52–58.

[9] OpenStack. 2017. Accessed in September 2017; URL `http://www.openstack.org`.

[10] OpenStack, . OpenStack cascading solution. 2015. Accessed in September 2017; URL `http://wiki.openstack.org/wiki/OpenStack_cascading_solution`.

[11] Liaqat, M., Chang, V., Gani, A., Hamid, S.H.A., Toseef, M., Shoaib, U., et al. Federated cloud resource management: Review and discussion. Journal of Network and Computer Applications 2017;77(Supplement C):87 – 105.

[12] Assis, M., Bittencourt, L.. A survey on cloud federation architectures: Identifying functional and non-functional properties. Journal of Network and Computer Applications 2016;72(Supplement C):51 – 71.

[13] Fogbow. 2017. Accessed in September 2017; URL `http://www.fogbowcloud.org/`.

[14] Cunsolo, V.D., Distefano, S., Puliafito, A., Scarpa, M.. Volunteer computing and desktop cloud: the Cloud@Home paradigm. In: IEEE International Symposium on Network Computing and Applications (NCA). 2009, p. 134–139.

[15] Jonathan, A., Ryden, M., Oh, K., Chandra, A., Weissman, J.. Nebula: Distributed edge cloud for data intensive computing. IEEE Transactions on Parallel and Distributed Systems 2017;:1–14In Press.

[16] Osorio, J.D., Castro, H., Brasileiro, F.. Perspectives of UnaCloud: An opportunistic cloud computing solution for facilitating research. In: IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. 2012, p. 717–718.

[17] López-Fuentes, F.A., García-Rodríguez, G.. Collaborative cloud computing based on P2P networks. In: IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA). 2016, p. 209–213.

[18] Shen, H., Liu, G.. An efficient and trustworthy resource sharing platform for collaborative cloud computing. IEEE Transactions on Parallel and Distributed Systems 2014;25(4):862–875.

[19] Sciammarella, T., Couto, R.S., Rubinstein, M.G., Campista, M.E.M., Costa, L.H.M.K.. Analysis of control traffic in a geo-distributed collaborative cloud. In: IEEE International Conference on Cloud Networking (CloudNet). 2016, p. 224–229.

[20] Gelbukh, O.. Benchmarking OpenStack at megascale: How we tested Mirantis OpenStack at SoftLayer. Mirantis; 2014. Accessed in September 2017; URL http://www.mirantis.com/blog/benchmarking-openstack-megascale-tested-mirantis-openstack-softlayer/.

[21] Cisco, . OpenStack Havana Scalability Testing. Cisco Systems; 2014. Accessed in September 2017; URL http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/OpenStack/Scalability/OHS.pdf.

[22] Li, A., Yang, X., Kandula, S., Zhang, M.. Cloudcmp: comparing public cloud providers. In: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC). 2010, p. 1–14.

[23] Persico, V., Montieri, A., Pescapé, A.. Cloudsurf: a platform for monitoring public-cloud networks. In: IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI). 2016, p. 1–6.

[24] Persico, V., Botta, A., Marchetta, P., Montieri, A., Pescapé, A.. On the performance of the wide-area networks interconnecting public-cloud datacenters around the globe. Computer Networks 2017;112:67–83.

[25] Garcia-Dorado, J., Rao, S.. Cost-aware multi data-center bulk transfers in the cloud from a customer-side perspective. IEEE Transactions on Cloud Computing 2015;.

[26] Kernel Virtual Machine (KVM). 2017. Accessed in September 2017; URL http://www.linux-kvm.org.

[27] Mininet, . An Instant Virtual Network on your Laptop (or other PC). Mininet Team; 2018. Accessed in March 2018; URL http://mininet.org/.

[28] Foundation, L.. OpenDaylight. Linux Foundation; 2018. Accessed in March 2018; URL https://www.opendaylight.org/.

[29] OpenStack, . Capacity planning and scaling. 2017. Accessed in September 2017; URL https://docs.openstack.org/arch-design/capacity-planning-scaling.html.

[30] Camilo, B.C.V., Couto, R.S., Costa, L.H.M.K.. Assessing the impacts of IPsec cryptographic algorithms on a virtual network embedding problem. Computers & Electrical Engineering 2017;:1–16In Press.

[31] Open vSwitch. 2017. Accessed in September 2017; URL http://openvswitch.org/.

[32] Open Networking Foundation. 2017. Accessed in September 2017; URL http://www.opennetworking.org/.

[33] Takabi, H., Joshi, J.B., Ahn, G.J.. Security and privacy challenges in cloud computing environments. IEEE Security & Privacy 2010;8(6):24–31.

[34] Lee, C.A.. Cloud federation management and beyond: Requirements, relevant standards, and gaps. IEEE Cloud Computing 2016;3(1):42–49.

[35] Shibboleth Consortium. 2017. Accessed in September 2017; URL http://www.shibboleth.net/.

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

[37] Cinder, . Generic image cache functionality. OpenStack Cinder Team; 2016. Accessed in September 2017; URL http://specs.openstack.org/openstack/cinder-specs/specs/liberty/image-volume-cache.html.

[38] Rally – OpenStack. 2017. Accessed in September 2017; URL http://wiki.openstack.org/wiki/Rally.

[39] iPerf. 2017. Accessed in September 2017; URL http://iperf.fr/.

[40] Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.. The case for VM-based cloudlets in mobile computing. IEEE Pervasive Computing 2009;8(4):14–23.

[41] Rede Nacional de Pesquisa. 2017. Accessed in September 2017; URL http://www.rnp.br.

[42] Réseau National de Télécommunications pour la Technologie. 2017. Accessed in September 2017; URL http://www.renater.fr.

[43] GÉANT. 2017. Accessed in September 2017; URL http://www.geant.org.

[44] Almesberger, W.. Linux network traffic control - Implementation overview. White Paper available in https://infoscience.epfl.ch/record/143/files/TR98_037.ps; 2001.

[45] Padhye, J., Firoiu, V., Towsley, D.F., Kurose, J.F.. Modeling tcp reno performance: a simple model and its empirical validation. IEEE/ACM transactions on Networking 2000;8(2):133–145.

[46] Libvirt: The virtualization API - accessed in march 2018. http://libvirt.org/; 2018.

[47] Richardson, T., Stafford-Fraser, Q., Wood, K.R., Hopper, A.. Virtual network computing. IEEE Internet Computing 1998;2(1):33–38.

Appendix A OpenStack Services

We employ in PID the following OpenStack services, whose locations are depicted by Figures 2, 3, and 4.

- `nova-api` provides an API for all Nova services. Note that the actions regarding VM management in the web interface (Horizon), such as VM instantiation and removal, are internally forwarded to `nova-api`. A given interaction in Horizon may also send requests to other API services, such as `cinder-api` and `glance-api`, detailed later;
- `nova-compute` communicates with the hypervisor APIs, to execute VM lifecycle management, such as VM creation and virtual resource resizing. In our prototype, these APIs are provided by Libvirt [46];
- `nova-novncproxy` provides a proxy to access VM screens through VNC (Virtual Network Computing) [47], which is a protocol to manipulate graphical interfaces;
- `nova-conductor` is a service that controls the interaction between `nova-compute` and the database, which stores different information about VMs and the infrastructure. Hence, it provides isolation to the database;
- `nova-scheduler` chooses in which VM Server a given requested VM is instantiated;
- `cinder-api` receives requests for the Cinder component, to create and remove virtual hard disks;
- `cinder-volume` interacts with the logical disks installed in its associated VM and Disk Server;
- `cinder-scheduler` chooses where a given virtual hard disk is installed. In our cloud, this module always install the disk in the same site that hosts its associated VM;
- `glance-api` receives requests for the Cinder component, such insertion of new VM images;
- `glance-registry` stores and process the image metadata;
- `ceilometer-api` receives requests from external application regarding resource utilization statistics. These requests are replied by consulting the Ceilometer database;
- `ceilometer-agent-compute` requests resource utilization statistics to the nova-compute service;
- `ceilometer-agent-central` requests resource utilization statistics to the services and components installed in the Controller;
- `ceilometer-collector` monitors and sends to the database notifications and statistics received from the `ceilometer-agent-compute` and `ceilometer-agent-central`;
- `ceilometer-alarm-notifier` allows the configuration of alarms based on resource utilization thresholds;

- `ceilometer-alarm-evaluator` controls the alarm triggers, by comparing the resource utilization statistics with the defined thresholds;
- `ceilometer-agent-notification` sends alarms based on the information received by the `ceilometer-alarm-evaluato`
- `neutron server` centralizes different network management functions and provides APIs to other components and services. For example, it receives requests to create virtual networks and forwards these requests to the VM Servers to configure network software, such as Open vSwitch;
- `neutron-ovs` configures and manages the Open vSwitch bridges in a given VM Server. This service also sets firewall rules;
- `neutron-L3` connects virtual routers between each other and to the Internet. Hence this service is responsible for performing routing inside the cloud;
- `neutron-metering` collects network utilization statistics. This information can be used, for example, by the Ceilometer component;
- `neutron-dhcp` provides DHCP services to the VMs.