Research and implementation of open source cloud computing platform based on Open Stack

ANG WANG¹, JING HUANG¹

Abstract. Implementation of open source cloud computing platform based on Open Stack is developed. Open Stack is a cloud computing operating system which aims to provide an open source cloud computing platform for both public and private clouds. In terms of infrastructure as a service (IaaS) platform, the Open Stack cloud platform faces a resource allocation limitation problem. This paper involves a customization of Open Stack Dashboard (horizon) to generate virtual network requests and then processing these requests. It also uses Open-Stack Computer to provide bandwidth limitations for the virtual network requests, as well as a modification of the data structure used between the Open Stack Horizon and Nova Modules. It implements a proposed virtual network embedded problem algorithm (VNE-Greedy) on top of the Open Stack Cloud platform. This paper also includes extended research of the Open Stack cloud computing system and its associated virtual network embedding strategies.

Key words. Evaluation model, implementation, open source, cloud computing platform, Open Stack.

1. Introduction

Cloud computing involves a large number of computers connected through a communication platform such as the Internet. It is similar to utility computing. Cloud computing is also defined as a large-scale distributed computing model. Here, the cloud providers and users can have their own private infrastructure, and several types of services can be provided to clients using virtual machines hosted by the providers. It includes utilization techniques for improving the efficiency of the system. These include: Network utility, Disk I/O utility, CPU utilization of a system as well as available memory for performing operations. Cloud Computing is a term that renames some common technologies and techniques that we know in IT. It can be understood to mean data center hosting [1–2]. The principal concept of computing goes back to the 1950s.

¹Henan University of Chinese Medicine, Henan, 450046, China

http://journal.it.cas.cz
This includes things like processing power resources at each node and bandwidth resources on the links that the embedding must fulfill [3]. For example, to run an experiment, a researcher may need 1 GHz of CPU for each virtual node and 10 Mbps for each virtual link [4]. Added to these requirements are the constraints on location and any link propagation delay problems [5]. In another example, a gaming service needs virtual nodes in many cities, as well as virtual links with very small propagation delays. These combinations of node and link constraints would make the embedding problem computationally difficult to approach and solve, see [6–7]. Figure 1 shows a simple virtual network request and how it was mapped on a substrate network.

![Fig. 1. Virtual network embedding](image)

### 2. Open Stack overview

In 2012, the Open Stack community was deeply involved in core technology development, as well as deploying and managing Open Stack projects [8]. Currently, Open Stack consists of seven core components: Compute, Object Storage, Block Storage, Network, Dashboard, Image Service and Identity. Few studies or research have conducted to test the usage and the benefits of implementing virtual network embedding strategies within the Open Stack cloud operating system architecture. Currently, there is much need for this type of research since Open-Stack is a new and emerging Internet technology, and as such, is facing a resources allocation problem. Open Stack is the leading cloud computing technology and it has now received attention from members of the scientific community. In this chapter, we provide a brief introduction to Open Stack, including its components, development, and research status. We also provide a brief introduction into the current research and study implications for connecting Open Stack and virtual networking. In this paper, we will discuss the virtual network embedding problem and the existing strategies for solving this problem. Open Stack is a cloud operating platform which controls resources of computing, networking and storage throughout a datacenter, in which all are managed through a dashboard that gives administrators control and authorizes their users to use resources through a web interface (dashboard). The goals of the Open Stack originality are to support cloud services and allow businesses to
build cloud services in their own data centers. Open Stack is available under the Apache 2.0 license freely, referred as "the Linux of the Cloud" and also comparing to Eucalyptus and the Apache Cloud Stack projects. Open Stack has a modular architecture that currently has three main components: compute, storage and image service [9–10].

3. Open Stack model and algorithm

Open Stack consists of a modular architecture along with various codes for the components. It also has several shared services which extend the three main components (computing, storage and networking). This makes it much easier to implement and operate on your own cloud. These services integrate the Open Stack components with each other along with external systems to deliver an integrated experience for users.

1. Open Stack Computer Nova is a cloud computing controller (considered as the IaaS system’s main component). It uses Python language as well as many external libraries like Event let, Kombu (for AMQP communication), and SQL Alchemy (for database access). Nova’s architecture is designed to be scaled horizontally on substrate hardware with no additional hardware or software requirements. This simultaneously provides the ability to integrate it with current legacy systems and third party technologies. Nova is designed to manage pools of computer resources and it can work with almost all available virtualization technologies. It can also work with high-performance computing (HPC).

2. Open Stack Object Storage (Swift) is a scalable storage system in which objects and files are written to multiple disks in the data center servers. Open Stack ensures data replication and integrity across the cluster. Storage clusters scale simply by adding new servers. If a server or a hard driver fails, Open Stack replicates its content to new locations in the cluster. Due to the fact that Open Stack uses software logic to ensure data replication and distribution across different devices, inexpensive servers and hard drivers can be used.

3. Open Stack Block Storage (Cinder) provides the software to create and manage a service that provides storage in the format of block devices known as Cinder volumes. Cinder provides persistent storage to guest virtual machines (instances) which are managed via Open Stack Compute. It can also be used independently with other Open Stack services.

4. Open Stack Network (Neutron) provides virtual networking service for the compute module. The networking module can manage IP addresses, which allows for dedicated DHCP or IPs. Network module suits the requirements of applications or user groups.

5. Open Stack Dashboard (Horizon) gives users and administrators an interface to access, automate and provide cloud resources. The design makes it easy
to plug in as well use third party products and services. This includes such things as monitoring, billing, and management tools. The dashboard can be customized for service providers and vendors who desire to use it. Dashboard is also a way to interact with Open Stack software resources.

6. Open Stack Image Service (Glance) provides location and delivery for services for disks and server images. It has the ability to snapshot and copy a server image and store it away. This is something very useful about the Open Stack cloud operating system. The stored snapshots and images can be used to get servers mining faster and more consistently. It also can be used to catalogue and store many backups.

7. Open Stack Identity (Keystone) controls the central directory of users mapped to the Open Stack services and can access it. It works as an authentication system across every part of the cloud operating system and can be integrated with already existing backed directory services such as LDAP. The Keystone module supports many types of authentication, including username and password credentials, as well as AWS-style and token based systems.

Open Stack Compute (Nova) is a controller for cloud computing and managing large networks of virtual machines (VMs). Open Stack Object Storage is a storage system that provides support for both block storage and object storage. Image Service is a Service which provides discovery as well as registration for visual disk images. Among many Open Stack services and projects (the list is growing with every release), only Compute is considered within this paper. Compute or "Nova" is the service responsible for providing a compute provisioning function to clouds. It can be considered as a management layer which operates on top of a free option of supported hypervisors, exposing a REST API for the purpose of management and provisioning. It consists of a set of service binaries that work together to accomplish one common goal. They all interact directly through messaging and through a shared state which is stored in a central database. This is shown in Fig. 2. To interact with other services, we can directly target the REST API or use the Python language provided by in the Python-Nova client library. This also includes a command-line client. Other interfaces, such as the web-based Dashboard, use this as client libraries for interacting with the different Open Stack services as well. Provisioning requests, which enter the API and then pass the initial authorization and verification, will step before being sent out to the Nova-scheduler to decide which one of the available compute nodes should be handling the request. Our main focus of this chapter is the customization of the Nova Open Stack main component. The actual code for the Nova services are in /nova and the corresponding unit tests are in the related directory under nova tests. The following represents a short explanation of the Nova source directory structure.

The basic equation of key algorithm is

\[(N, sk) \leftarrow Key(1^k), \quad (1)\]
where $N$ is the node number, $sk$ is the scheduler and $k$ is the database dimension. This formula is used to generate file checksum parameter which is denoted as

$$r \leftarrow \{0, 1\}^k; sk \leftarrow \{e, d, r\};$$

.Output\{N, sk\}; 

(2)

where $r$ is the hypervisor. The Euler function is

$$\phi(N) = (p - 1)(q - 1),$$

(3)

where $p, q$ are the parameters of the file storage block. The function $\phi(N)$ satisfies the following constraints

$$\gcd(e, \phi(N)) = 1, \quad 1 < e < \phi(N),$$

(4)

where $e$ is the natural constant and $\gcd()$ returns the maximum common divisor.

4. Experiment and discussion

The Hardware Layer and the Virtualization Layer are operated by the cloud services provider, while the Client Layer is provided by the end users. Basically, clouds can be defined in three ways.

1. Private Clouds: where data and processes are managed in the organization without security exposures and legal required menu.

2. Public Clouds: where a set of computers and computer network resources, in which a service provider provide resources (like storage and applications) is available to the general public over the Internet.
3. Hybrid cloud: a combination of two or more clouds (private, public and community) that remain different entities but bound together. These provide the benefits of multiple cloud models. Therefore, hybrid cloud means the ability to connect, manage and dedicate services with cloud resources. A hybrid cloud service crosses isolation boundaries so it cannot simply be categorized as a private, public or community cloud. Virtual network embedding (VNE) has been a major challenge for future Internet (FI). The problem of embedding virtual networks within a substrate network is the main resource allocation in network virtualization.

The target of the VNE problem is the allocation of virtual resources in nodes and links. Therefore, it is divided in two sub problems: first, Virtual Node Mapping (VNoM), where virtual nodes are mapped in physical nodes; and second, Virtual Link Mapping (VLiM), where the virtual links linking virtual nodes have to be mapped on paths connecting these nodes in the substrate network. Future Internet architectures have been evolving to become based on the Infrastructure as a Service (IaaS). This is a model that divides the role of current Internet Service Providers (ISPs) into two new main roles: first, the Infrastructure Provider on P which deploys and keeps the network equipment operating; and second, Service Provider (SP). Service provider (SP) is responsible for deploying various network protocols and providing end to end services. For example, Voice over IP (known as VoIP) can run on a virtual network which provides anticipated performance.

This is done by provisioning dedicated resources and employing routing protocols which can ensure fast recovery from any equipment failures that might occur. On the other hand, online banking runs on a virtual network that provides security guarantees. This is done through addresses and secure routing protocols. Making efficient usage of the substrate resources demands effective techniques for visual network embedding (a new virtual network mapping). The VNE problem is extremely difficult for two main reasons, the first of which being node and link constraints. Each VN request has resource limitations.

Open Stack Compute scheduler is also known as the Nova-scheduler service. It is responsible for mapping instance requests onto the physical hosts named compute nodes. Compute nodes execute the Nova-compute service on top of a hypervisor. When the scheduler service is launched, it will load a scheduler driver which holds the actual scheduling logic and policies. A scheduler driver is derived from a base driver class and implements interface. A number of simple scheduler drivers are included in Open Stack Nova. Advanced filters can be written as long as they able to implement the required interface of the base driver. In addition to defining the interface requirements, it also holds some of the basic requirements which are needed by every scheduler; this includes easy access to the global system state and some utility methods used by most schedulers. Figure 3 shows us the existing network and connected virtual bridges. When entering the command to view virtual bridge and virtual network port status we bet the Fig. 4.

The core Open Stack services are: API, Compute, Scheduler and Network. We also need the Glance Image Service as a guest of OS images (which is backed using the Swift Storage Module Service). We will now dive into each of these services,
and I will explain each one’s job and task within the module. API is the HTTP interface used in Nova. Compute communicates with the hypervisor serving each host (usually one Compute service for each host). Network manages IP address real-time communicating with the routers, switches, firewalls and other devices. Scheduler chooses the most appropriate computing node from the available pool (though it may also be used in selecting the Volumes). The database is not considered to be a Nova service, but a database can be retrieved directly from any of Nova services (however it should not be accessed via the Compute service). We can also run a stand-alone Authentication service or Volume service for disk management, but this is not required. Open Stack Nova uses AMQP (also known as Rabbit MQ) as the communication pass between the services. AMQP messages are written to special queues and one of the related services picked for processing. This is actually how Nova scales. If you find a single Compute node that can’t handle the requests number, you can throw in another Compute node service. The same goes for the other services. Figure 5 shows the Iperf bandwidth test and Fig. 6 shows the Iperf client side bandwidth test.

Figure 7 below is snapshot of the use of the tool iperf to test the bandwidth between IPL2 1 2 and IPL2 1 3 before implementing the proposed method. As we call notice, the bandwidth is not limited and it has allocated large resources of bandwidth for the given request. The initial allocation for the bandwidth is 83 0 Mbit/sec. Furthermore, if AMQP is considered to be the only way to communicate with the Services, how do the users issue commands? The answer is definitely the API service. This service is an HTTP service (a WSGI application known in Python). The API service listens for the coming REST commands via the HTTP service and interprets them into AMQP messages for those services. Similarly, responses from the services come in via AMQP and the API service turns them into valid HTTP responses in the format the user has requested. Open Stack currently uses EC2 (API) and Open Stack. However, it is not only API that communicates with the services. Services can also communicate with each other.
Fig. 4. Virtual bridge status

```
root@IPL213:~/home/openstack# ovs-vsctl show br-int
OFPT_FEATURES_REPLY (xid=0x1): ver: 0x1, dpid: 00007e2e3d4a0940
n_tables: 255, n_buffers: 256
features: capabilities: 0xc7, action: 0xfff
1(tap556ebc58-c8): addr: fe: 16: 3e: af: ff: 54
  config: 0
  state: 0
  current: 10MB-FD COPPER
2(tap215779b8-aa): addr: fe: 16: 3e: 6e: 4a: 81
  config: 0
  state: 0
  current: 10MB-FD COPPER
  config: 0
  state: 0
  current: 10MB-FD COPPER
5(tap9f9a565a-e1): addr: fe: 16: 3e: ba: d2: bc
  config: 0
  state: 0
  current: 10MB-FD COPPER
  config: 0
  state: 0
  current: 10MB-FD COPPER
7(int-br-eth1): addr: 8a: bc: e5: 63: 0b: cb
  config: 0
  state: 0
  current: 10MB-FD COPPER
LOCAL(br-int): addr: 7e: 2e: 3d: 4a: 00: 40
  config: PORT_DOWN
  state: LINK_DOWN
OFPT_GET_CONFIG_REPLY (xid=0x3): frags=normal miss_send_len=0
```

Fig. 5. Iperf bandwidth test

```
root@openstack-virtual-machine:~/home/openstack# iperf -s -D
root@openstack-virtual-machine:~/home/openstack# Running Iperf Server as a daemon
The Iperf daemon process ID: 3846
```
Figure 8 shows the bandwidth allocation after implementing the method proposed in this thesis between the nodes IPL2 1 2 and IPL2 1 3. Compute may need to communicate with Network and Volume to get the necessary resources. If we are not careful with how we organize the source code, all of this communication will experience problems. Therefore, for our initial article, let us dive into the Service and RPC mechanism will be using the Python unites notation for the modules, methods and functions. Specifically, `nova.compute.api`: APL run the instance connects to the run instance method of the API class in the `/novalcompute/api.py` file. With the exception of the API service, each Nova service must have a related Python module to handle RPC command marshaling. For example: The network service has `lnovalnetworklapi.py`. The compute service has `lnovalcomputelapi.py`. The scheduler service has `./nova/scheduler/api.py`. These modules are usually just large collections of functions that make the service do something. However, sometimes they contain classes that have methods to do something. It all depends on if we need to intercept the service calls. We will touch on these use cases a little later.

5. Conclusion

Through this paper, our objective was to implement, evaluate and demonstrate of VNE strategy in Open Stack. And that was done by first conducting research on related cloud computing and virtualization moreover by gaining an understanding of the Open-Stack Operating system more specifically Dashboard (Horizon) and Compute (Nova) architecture and its scheduler component. Later, a working test bed built using Open-Stack was set up and used to implement a scheduler for initial VMs nodes. Finally, the test bed was used to validate and conduct experimental evaluation to showcase its performance.

Customization and modification of the Open Stack Horizon, by customizing the GUI interface for the users to input the nodes and links intend to be embedded in Open Stack, intensive reading and comprehends for Open Stack source code, this all was required along with a good commend of Python and Django to accomplish this task (Customizing Open Stack Dashboard). We also used debugging tools such PDB to capture the data flow and parameters handling in the fabric of the Open Stack.
Fig. 7. Bandwidth between VMs in IPL212 and 213

Fig. 8. Bandwidth between IPL 212 and IPL 213 after implementing VNE

References


Received November 16, 2016