OpenStack on Servers Without Hard Drives

Andre Tättar
University of Tartu
Institute of Computer Science
Estonia, Tartu
Email: andre.tattar@ut.ee

***Abstract*—The goal of this project is to create a private cloud which could operate on servers without hard drives. That means each server would have a motherboard, a CPU, some RAM and a network interface. This would allow for a very scalable and cheap system. The point to worry about is how will it affect performance. What is the bottleneck? How important is storage space for servers?**

I. Introduction

The goal of this paper is to develop an infrastructure for a private cloud with very limited resources. The objective is to create an infrastructure such that there would be one main server node that serves the filesystem, operating system and configurations for other servers over the network. These headless servers would then have OpenStack running on them, which would allow for easy cloud operations with easy management and a friendly interface to operate Virtual Machines (VMs).
The hypothesis is that the solution will work, because the kernel and all of the neccessary software required to operate a working Debian 8.6 server only requires less than 200 MB of RAM, which will all be stored in RAM, so operating system should not do much IO over network. Additionally OpenStack compute nodes require about 200 MB of software and all of that also can be fit into RAM easily, which would leave more than enough RAM to run 10 or more small Virtual instances. The assumption is that not a lot of IO will be done.
This solution will have drawbacks and some tests will be run to figure out exactly what are the bottlenecks of this kind of a system. Additionally, we will try to answer the question: can this infrastructure handle 100 small VMs, which will be used by students in System Administration course.

II. Related Work

This project is a continuation of two Bachelor theses, both supervised by Artjom Lind. The objective is to combine the two theses and test the resulting setup.

A. Building and Configuring a Custom Private Cloud Using Consumer Hardware [1]

This paper was done by the author this paper last year and was about building, configuring and testing a private cloud using consumer hardware [1]. The paper firstly describes clouds, which more precisely refers to cloud computing - using resources that one can connect to over a network. These resources could be anything from VMs to services like mailhost hosted somewhere. Generally clouds benefit from less IT management, accessibility, high computing power, lower costs, scalability and availability. Private clouds are described as clouds that are managed, built and used by one to many organizations and access to such clouds are usually limited to organizations themselves and their partners. Compared to public clouds, private clouds have greater reliability, flexibility, efficiency and improved data security, but also have disadvantages over public clouds - more maintenance, more difficult scalability and greater cost.
The paper discusses different cloud softwares, but OpenStack is chosen as the cloud computing software. OpenStack is an open source software that can manage small to large sized compute, storage and networking resources, all managed by a dashboard. It is highly scalable and flexible. The author uses Ubuntu 14.04 LTS as the operating system and KVM as the hypervisor. All choices are explained.
OpenStack has many components, here I will provide a brief overview, but do read the original thesis for more information.

- Keystone - Identity Service - Responsible for authentication and authorization for other OpenStack services, used as a common unified API. Can be used with existing authentication services like LDAP. Each OpenStack service must be registered with Keystone.
- Glance - Image Service - Stores and manages virtual machine stock disk images. Accepts and handles API requests for disk or server images and metadata definitions. Supports storage of image files on different file systems like nfs and object storage (OpenStack Service).
- Nova - Compute Service - Manages computing services in OpenStack. On controller it has many services like Nova-api, which is a link between the dashboard (Horizon) and other Nova compute services. Additionally provides vnc-proxy and some other services. On compute servers, nova-compute is run, which is responsible for communicating with hypervisor and controller and managing/running/destroying compute instances.
- Neutron - Networking Service - Provides network connectivity as a service between OpenStack services, most importantly Nova compute services (VMs). It implements the Neutron API, which enables users to define and create networks, subnets and ports that OpenStack can use and attach compute instances to networks, using virtual networking. Supports Layer 2 and Layer 3 networking.

- Horizon - Dashboard - a graphical web-based interface of OpenStack and links togeter all other OpenStack services in order to view, create and manage all the cloud interfaces.

The main point of the thesis is about networking and how to satisfy the OpenStack requirement of two network interfaces for servers, which was solved by using VLANs. Additionally, results for network performance and storage read/write speeds are tested. The thesis states that OpenStack is a viable option on consumer hardware.

B. Fault-tolerant networking using Linux based systems and consisting of used hardware [2]

The second thesis was done by Anders Martoja and was about fault tolerant networking using Linux based systems on obsolete hardware [2]. Basically he created a fault-tolerant network with consumer grade hardware that was reliable and used free open source software. The most important part for this paper was that he described how to achieve headless servers, so each server only had a network interface, CPU, motherboard and RAM, so no storage at all. He achieved this with additional services, which he set up. The services are as follows:

- BIND - Berkeley Internet Name Domain - open source software that enables one to publish their Domain Name System (DNS) information on the web, and to resolve DNS queries for one's users.
- DHCP - protocol that hands out IP addresses to computers connected to the DHCP server. DHCP must be configured with a pool of IPs, and DHCP server leases these IPs out. All the neccessary information is given out along with IP, such as a subnet, gateway, nameserver etc.
- TFTP - Trivial File Transfer Protocol - file transfer protocol used for network booting diskless servers. Author uses thtpd-hpa, as HPA's implementation of TFTP is more stable and more portable. Functionality includes read/write to/from server, but lacks security features, so using a firewall is a must.
- Apache2 - open source HTTP web server software. Highly configurable and able to work with many operating systems and environments.
- iPXE - implementation of Preboot eXecution Environment (PXE) with enhancements - iPXE is an open source network boot firmware. Provides booting from a web servers and networks.
- NFS - Network File System - distributed file system, which allows servers to mount file systems over the network. The kernel sees the file system like an internal storage. NFSv4 is used in the thesis.
- iptables - command line program used to configure firewalls on many Linux systems. Iptables supports many rules, like filtering, routing, chaining, masquerading, port forwarding etc.

III. Foreseeable problems

The resources for servers are even more limited in the current setup than they were in the first thesis [1]. Let's analyze the resources:

- RAM - Low amount of RAM for running virtual machines. This is a limiting factor for how many VM-s can be run on one server. Being without a hard drive should not affect RAM availability much, because for the system everything "feels" like the storage is there. For this setup about 6.7 GB RAM was available for our servers. Additionally, fresh Debian installations require 180 MB RAM, but headless servers require around 250 MB.
- 1 NIC - OpenStack requires two or more network interfaces to function optimally. But because additionally NFS will run over the same network (probably one additional VLAN), there will be much more traffic, because all the compute nodes are also stored over NFS. Expectation is that a lot of traffic will be added to the network. If network traffic drops a lot, we hope to fix that issue by adding more network interfaces for NFS sharing server and increase throughput by bonding those interfaces together.
- Storage - We only have a couple of SSDs to work with and we don't have one SSD for every server. Storage space could become a limiting factor and also speed, because data transfers will be limited by network interface speeds, which will be 1GbE. In comparison, the SATA cables have a speed of 6 Gbit/s and less hops - network traffic goes through a switch to the NFS serving node and then through the SATA cable, which will also increase latency.

IV. Setups

Two different OpenStack clouds will be set up. One is with SSDs and the other one will be headless. The first cloud will have a controller node and one compute node, where all of OpenStack's neccessary components will be on controller and only hypervisor and compute service will be on compute node. Headless setup will additionally have one additional server only serving the filesystem, operating system, configurations and compute instance qcow files.

A. SSD setup

This setup is almost identical to the setup described in [1], only that Ubuntu 16.04 LTS will be used and new OpenStack release called Newton will be used. Networking will be the same, except that the network will be served over 100 Mbit/s links. Internal communications could be made faster very easily, by using a 1GbE router and that is why we do not consider internal communication as a drawback here. We will do additional tests for CPU and RAM usage. I describe what are meant on the CPU cycle graphs

- us, user : time running un-niced user processes
- sy, system : time running kernel processes

TABLE I
RAM USAGE OF CONTROLLER NODE - CASE 1 IS IDLE, CASE 2 IS VNC AND CASE 3 IS LAUNCHING AN INSTANCE

Case	Total	Used	Free	Cached	Available	Swap
1	6.7G	4.4G	1.1G	1.3G	2.0G	676M
2	6.7G	5.5G	259M	1.0G	973M	692M
3	6.7G	5.6G	187M	1.0G	858M	639M

TABLE II
5 ITERATIONS OF CPU USAGE OF CONTROLLER

Tick	us	sy	ni	id	wa	hi	si	st
1	11.6	0.7	0.0	87.5	0.2	0.0	0.0	0.0
2	12.6	0.7	0.0	86.7	0.0	0.0	0.0	0.0
3	13.2	0.8	0.0	85.6	0.3	0.0	0.1	0.0
4	12.2	0.7	0.0	86.9	0.2	0.0	0.1	0.0
5	12.4	0.8	0.0	86.7	0.2	0.0	0.0	0.0

- ni, nice : time running niced (low-priority) user processes
- id, idle : time spent in the kernel idle handler
- wa, IO-wait : time waiting for I/O completion
- hi : time spent servicing hardware interrupts
- si : time spent servicing software interrupts
- st : time stolen from this vm by the hypervisor

In Table I there is given the RAM usage of an OpenStack controller. The OpenStack is in its normal state, working and handling 3 virtual machines with one compute node. Test 2 is when vnc is used over horizon and test 3 is when there is a lot of traffic from the Horizon web interface. Additionally, CPU usage is shown for when OpenStack is running and Horizon is not used in Table II, where CPU is on idle around 86-88 % of the time, and in Table III the CPU usage is shown when Horizon is being used, where CPU is idle for around 61 to 75 % time. We can see that the user processes are increased by a lot and also system kernel and software interruptions have increased. Moreover, the web frontend is a bit slow and laggy and pages take a long time to load. That might be because Apache and Horizon require a lot of CPU power and also a fair amount of RAM. This problem might be solved by adding additional RAM/CPU power to the controller (upgrading controller hardware) or by putting Horizon and Apache on a different node, so it would not interrupt other processes on controller.

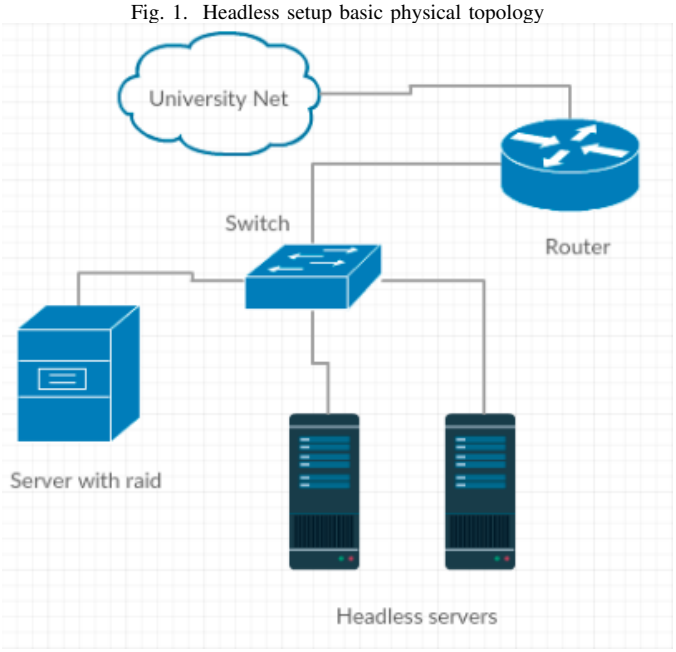
B. Headless setup

The headless setup was not set up by me, but by Anders Martoja and Artjom Lind. It has the current latest Debian 8.6 as the operating system. In Figure 1 there is the basic topology of a headless setup. The OpenStack would go on headless servers. There are currently 11 headless servers running.

TABLE III
5 ITERATIONS OF CPU USAGE OF CONTROLLER WHEN USING HORIZON

Tick	us	sy	ni	id	wa	hi	si	st
1	28.9	1.3	0.0	69.2	0.2	0.0	0.4	0.0
2	22.8	1.3	0.0	75.6	0.2	0.0	0.2	0.0
3	36.0	1.5	0.0	61.9	0.2	0.0	0.4	0.0
4	36.9	1.3	0.0	61.1	0.3	0.0	0.6	0.0

Fig. 1. Headless setup basic physical topology



This is where problems were encountered, Debian manuals on OpenStack are clearly out of date. Firstly, I tried to set up the controller node on headless servers, except that from the first service it was already troublessome. The documentation suggested to install mysql-server, which is mariadb on Ubuntu and mysql on Debian, but documentation suggested that this installs mariadb on Debian. This was however a small issue. Next problems were encountered with setting up the first service - Keystone. The documentation suggested commands which were not even available. After many tries to fix the problem, decision was made to setup Debian compute instance for the SSD setup.
The same Debian version was used in the SSD setup and another node was added with an SSD. Problems arose with the compute node aswell. Even after fixing the problems, new ones came up.
In the end, Debian Sid was booted from chroot, but got broken and it did not yield good results. There is much work to be done with this, learned a lof about OpenStack and other services while debugging.

C. Unforeseeable problems

Two main problems came out when testing or setting up the SSD setup and the headless setup.

- Controller - The controller is a special type of server, which will also be headless and will have all the OpenStack neccessary components running, which control the operations of OpenStack compute nodes and many other services. All the OpenStack services when only running one compute node require around 5 GB of RAM. When only one VM is running, it increases when instances are added or when compute nodes are added. A solution

might be to buy better hardware for the controller or distribute some OpenStack services to other nodes.
- Difficulties with Debian - OpenStack documentation has poorly written manuals for a Debian setup and it definately is not up to date. More work is required to set up compute OpenStack services on Debian servers.

V. Conclusion

In conclusion, there are two things to be done. Firstly, the controller is a special type of server for OpenStack and responsible for a lot of services. The controller should be separate from the headless setup with its own SSD, additionally with a better CPU and with more RAM. Secondly, more work is required when using Debian as the operating system. Because OpenStack on Debian 8.6 proved to be a harder task than imagined, I suggest trying Debian Stretch, which has the same versions for nova-compute and neutron. Also Debian on a headless setup could be switched for Ubuntu, on which it is easier to setup OpenStack, or maybe try Debian Stretch more. Although trouble came up with Debian, it is definately possible to set up OpenStack on Debian, but it requires more time because there are a lot of configuration choices and fixes. The right set up just takes time.

References

[1] A. Tättar, "Building and Configuring a Custom Private Cloud Using Consumer Hardware." University of Tartu Bachelor thesis, 2016. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?language=en.

[2] A. Martoja, "Fault-tolerant networking using linux based systems and consisting of used hardware." University of Tartu Bachelor thesis, 2016. http://www.tuit.ut.ee/sites/default/files/tuit/atprog-courses-bakalaureuset55-loti.05.029-anders-martoja-text-20160531.pdf.