Network Optimization in the Open Stack Cloud

Pramod Gavali
Student,
Department of Computer Engineering
S.K.N.C.O.E, Savitribai Phule Pune University
Pune, India

S. P. Pingat
Assistant Professor,
Department of Computer Engineering
S.K.N.C.O.E,Savitribai Phule Pune University
Pune, India

ABSTRACT
Cloud systems software requires a VM placement engine that decides where to place the virtual machine in the environment. The placement engine in the cloud platform such as OpenStack considers multiple constraints when launching a new instance, including required compute and memory resources. This placement mechanism does not considers network requirements of VM In this paper, we propose the solution to optimize the network resources which is easily integrated in the OpenStack placement engine. Our solution keeps the track of the the traffic following within the physical network devices and the appropriate action is taken to optimize the network resources including migration of existing VM.

General Terms
VM- Virtual Machine

OpenStack- Open Stack cloud software

CPU- Central Processing Unit

RAM- Random Access Memory

CN- Cloud Network

Keywords
Cloud, Network optimization, Virtual Machine , Container, Placement, Migration

1. INTRODUCTION
The OpenStack[1] cloud computing platform has sophisticated mechanisms for managing the CPU, memory, and storage resources. However, Openstack lacks behind managing the network resources. The OpenStack uses the logic that launch instances such as Virtual Machines (VMs) to the physical server is called scheduler or placement engine. The default placement engine only considers CPU and memory resources. It does not consider network requirements. Other cloud computing platforms e.g. Eucyliptus have similar limitations regarding network awareness. A lack of network optimization in the cloud management platforms risks performances issues in the data plane and a violation of the Quality of Service (QoS) [2]. For example, there could be a congestion in shared resources on servers, which leads to increase the job queue and results into QoS violation.. In the past there has been substantial progress regarding theoretical solutions and designs for addressing the network optimization. But none of them are practically implemented in the existing cloud platforms. In this paper we analyzes the network optimization problem and design a solution that can easily integrated in the existing cloud computing platform -Openstack.. Section 2 explains the exiting work to address network optimization problem. In Section 3 captures the mathematical model . Section 4 introduces the Openstack architecture and reviews the functionalities of its existing default scheduler. In Section 5, we analyze the solutions for optimizing network resources during VM placement. Later, we propose the design of our prototype implementation network optimization in OpenStack. Section 6 Finally, concludes this paper

2. RELATED WORK
This section briefly describes the efferent techniques already designed for cloud network optimization

2.1 NETDEO
The NetDeo[3] is designed for improving the better utilization of the existing network resources and infrastructure .Without changing existing network architecture and routing protocol intact, it aims at reducing network bandwidth requirement by optimizing placement of VMs in the cloud. This mechanism has lower up-front cost and immediate effect.

2.1.1 Advantages
- Applicable in the dynamic environment where the servers are getting added and removed from the environment very frequently
- Continuous and incremental optimization of the cloud network
- Performs good in Tree[10][11] and FatTree [11] network topology

2.1.2 Disadvantages
- The traffic agent needs to be installed on the all servers that keep on sending the traffic information for given server . Its very difficult to install and maintain the traffic agent software if the number of server in the environment are very huge
- Does not perform well in Bcube [9] topology
- Performance degrades when number of hosts are increased

2.2 MAPLE
A MAPLE[4] is a network-aware Virtual Machine placement technique that uses estimate of the effective bandwidth required for compute hosts to ensure that the cloud network performs within targets specified in the SLA for the customer application. It allocates network resources to make perfect balance between efficiency of resource utilization and the performance requirement. The MAPLE is designed to provide a network-aware VM placement policy used to launch the set of VMS on different servers. The MAPLE does require the a prior reservation of bandwidth.

2.2.1 Advantages
- Good performance in case of deploying multiple VM at a time

- Less QoS violation compared to NetDEO[3] and Oktupus[5]

2.2.2 Disadvantages

- Not applicable in dynamic environment where the server are upgraded very frequently
- Applicable only for tree topology.

2.3 Oktopus

This Oktopus[5] is designed for multi-tenant cloud provider that accepts the network bandwidth requirement from the tenant and try to reserve the required network bandwidth while deploying the tenet application in the cloud. The user specifies the network bandwidth requirement as *Virtual Network* . It provides the abstraction of the virtual network by which it determines the trade-off between the network bandwidth guarantees offered to customer and the cloud provider's profit.

2.3.1 Advantages

- Virtual Network profiles allows to mentions the QoS requirement
- Physical Network sharing ratio is increased in multi-tenant environment

2.3.2 Disadvantages

- Not applicable for dynamic environment where the infrastructure is frequently upgraded
- Does not perform VM migration

2.4 Traffic Aware Virtual Machine Placement Problem(TVMPP)

The TVMPP[6] is a network performance problem. The traffic matrix among the virtual machines and the cost matrix among the compute host machines is considered as the input for this algorithm. The output of this algorithm is the optimal solution that suggests which compute host the VMs should be placed in order to improve the cloud network. The aggregate traffic rates dictated by every network devices like switches and routers. The TVMPP is NP-hard problem and propose a heuristic approach to solve the TVMPP efficiently for large environment. The algorithm follows a novel two-step approach: it first makes partitions of VMs and compute hosts in the clusters. Then it finds matching VMs and compute hosts within the cluster and subsequently at individual level.

2.4.1 Advantages

- Good performance in varying and heterogeneous traffic condition
- Bcube topology is benefited more with TVMPP

2.4.2 Disadvantages

- Worst performance in Tree network topology
- Not applicable for dynamic environment where the infrastructure compute and network devices get upgraded continuously. The NetDeo[3] is designed for improving the better utilization of the existing network resources and infrastructure .Without changing existing network architecture and routing protocol intact, it aims at reducing network bandwidth requirement by optimizing placement of VMs in the cloud. This mechanism has lower up-front cost and immediate effect.

3. PROBLEM DEFINITION AND FORMULATION

The network optimization problem is defined as a set of service nodes {n1, n2, ..., nN} on a collection of networked server systems {s1, s2, ..., sM}, where N and M are respectively the total number of nodes and servers in the system.

The traffic load between two servers nx and ny is defined as the product of their traffic , path length, and inverse path reliability:

$$TLoad(n_x, n_y) = D_{xy} \times L_{ij} \times R_{ij}^{-1}$$

The traffic load of a service node is defined as the quadratic mean of the traffic loads between the service/node and all its communicating pairs:

$$NodeLoad(n_x) = \sqrt{\frac{1}{Nx}\sum_{y=1}^{Nx} TLoad\left(n_x, n_y\right)^2}$$

where Nx is the number of service nodes communicating with node n_x

4. OPENSTACK ARCHITECTURE

4.1 Overview

A cloud computing platform manages pools of compute, storage, and networking resources to offer them on demand, e.g., as Infrastructure-as-a-Service[9] (IaaS). This paper focuses on the OpenStack [1], which is widely used cloud platform by cloud service providers. It is an open source cloud platform that comprises of a set of interconnected development projects.. The main components are implemented in Python, and they are accessible through REST API. The web interface is also used to manage different components. The OpenStack architecture is extensible and flexible. Figure. 1 shows a simplified overview on the OpenStack software architecture.

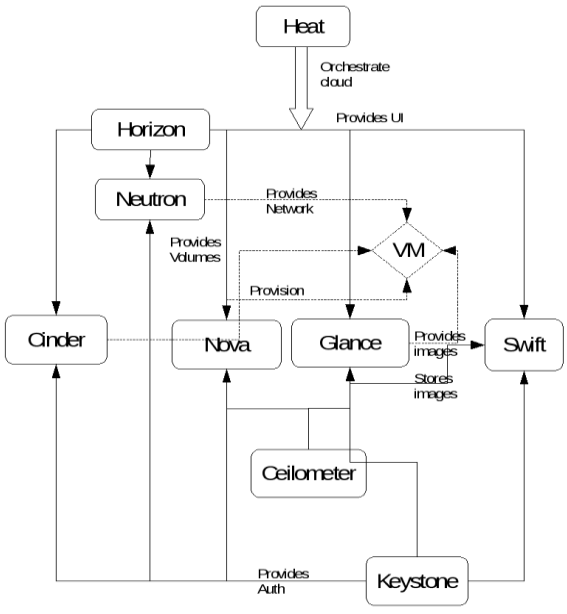

Fig. 1: Open Stack Architecture Overview

A cloud computing environment can be built by subset of these components, and systems can also be extended by other components that are not officially part of the OpenStack platform.

4.2 VM Placement

"Nova" is the components of OpenStack that manages the hypervisors on the physical server and controls the execution of VM. The "scheduler" controls the compute resources on the hypervisors of the available hosts. If a new VM/instance is launched, the scheduling logic/placement engine selects the most suitable physical server to host the new instance. The default placement policy in Open Stack is the "filter" scheduler [8][11]. It uses a 2-step process of "filtering" and "weighting" to make decision on which server a new VM should be launched.

4.3 Existing drawbacks and Challenges

The existing filter scheduler determines a set of acceptable compute hosts by filtering and weighting. Each time it selects a host to launch a new instance, the scheduler consumes some resources so that subsequent evaluations can adjust accordingly. It suffers from few limitations as below:

No Network-aware filter[10]: There is not filter the select the host based on the available network bandwidth on the host

Static data structure: The VM scheduler maintains the static own view of the resource allocation on each nodes, This view is not frequently updated dynamically .

5. PROPOSED DESIGN

Figure. 2 shows the proposed architecture of the systems that could add network awareness in the VM placement

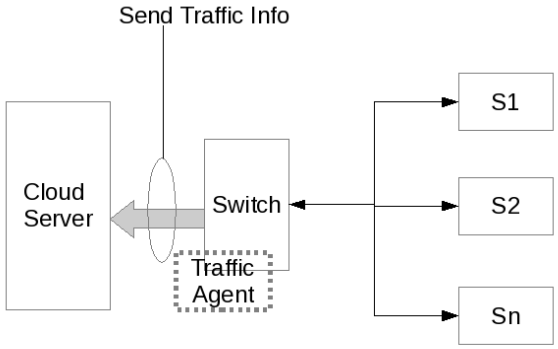

Fig. 2: Proposed Design

Basically, the traffic agent is installed on the centralized system , like router or switch . The router or switch is most suitable candidate as the all the traffic goes through it . This helps the traffic agent inspect the packets based on sampling mechanism and send the traffic data to the cloud server We can make use of exiting technologies like sFLow[8] which can be installed on the switches . The sFlow works on sampling techniques that inspect the packets on the configured interval. It keeps on sending the sampled pack to the Collector . In our case the collector is nothing but the cloud server which collects the packets form the switch and make analysis of which link is heavily used and which VM are talking each other over the physical network. Based on this information the appropriate CN optimization can be performed .

5.1 sFlow collector

The slow collector is the process running on the cloud server that keeps listening the UDP packets form the sFlow agent. The collector keeps records of the more recent traffic information and make analysis of the traffic identifying top N talkers. The list of top N talker is sent to VM placement engine to run a VM placement algorithm

5.2 Algorithm

Algorithm 1 is the main algorithm used for network aware virtual machine placement in the cloud. It contains following steps,

- Step 1: It collects traffic information using GetTrafficInfo() function.
- Step 2: Find the list of target physical machine to accommodate new VM. The function FindTargetPM() is used.
- Step 3: Select the VM to be migrated using function SelectVMNode().
- Step 3.1: The improvement in objective is calculated using. AcceptMove() If the move is not accepted, continue step 3.1.
- Step 3.2: Explores neighborhoods by exchanging different server on the candidate server to the original server.

Algorithm 1: The Netwok-Aware VM placement Algorithm

Input: *p_scheme_in* - current placement scheme
e_in - Initial system thermal energy
max_trial- Maximum number of trials to run
rate- sampling rate
max_packet- maximum packets to be received in one trial
Output: *p_scheme_out* resultant placement scheme
e_out resultant thermal energy

```
1  cnt:=0;
2  while cnt <= max_trial do
       /* Step 1 : Get traffic information */
3      traffic_info:=CollectTrafficInformation(rate,max_packet)
       /* Step 2: Find target Physical Machine having capacity to to
          accommodate new machine */
4      target_pm_list:=FindTargetPhysicalMachine()
       /* Step 3: Select VM that causing network performance issue */
5      vm_node:=SelectVMNode() /* Evaluate Trial Moves */
6      foreach target_pm in target_pm_list do
           /* Step 3.1 Try -1 Displacement Neighborhood */
7          p_scheme_out_temp:=MoveNode(target_pm,vm_node)
           /* calculate the output energy saved */
8          e_out:=e_in - CalcalteEnergy(p_scheme_out_temp)
           /* Step 3.2 Accept move */
9          move_accepted:=AcceptMove(e_out,e_in)
10     end
       /* Step 4 Handle Accepted moves */
11     if move_accepted then
12         p_scheme_out:=p_scheme_out_temp + p_scheme_in
13     end
       /* increment trial counter */
14     cnt:=cnt + 1
15 end
```

Algorithm 2: Collect traffic information using packet sampling

```
function CollectionTrafficInformation(rate,max_packets)
    packet_list:=[];
    Total_packets:=0;
    Total_samples:=0;
    skip:=NextSkip(rate);
    /* process all the incoming packets */
    while max_packet <=0 do
        packet:=ReceivePacket();
        exclude_packet:=ParsePacket()
        if not exclude_packet then
            AssignDestinationIterface(packet)
            skip−;
            Total_packet++;
            if skip = 0 then
                skip:=NextSkip(rate);
                Total_samples++;
                packet_list.append(packet);
            max_packets−;
    return packet_list
end function
```

Algorithm 3: Collect traffic information using packet sampling

```
function FindTargetPhysicalMachine()
    target_pm_list:=[];
    pm_list:=GetAllPhysicalMachine();
    /* iterate over all physical machine and find the one that has capacity */
    foreach target_pm in pm_list do
      has_cpu_and_mem_capacity:=CheckCapacity(target_pm)
      if has_cpu_and_mem_capacity then
          has_network_bandwidth:=
          CheckNetworkBandwidth(has_cpu_and_mem_capacity)
      /* Add physical machine into target list if it has cpu, memory and
      n/w capacity                                                    */
      if has_cpu_and_mem_capacity and has_network_bandwidth then
          target_pm_list.append(target_pm)
    return packet_list
end function
```

6. TEST ENVIRONMENT AND RESULTS

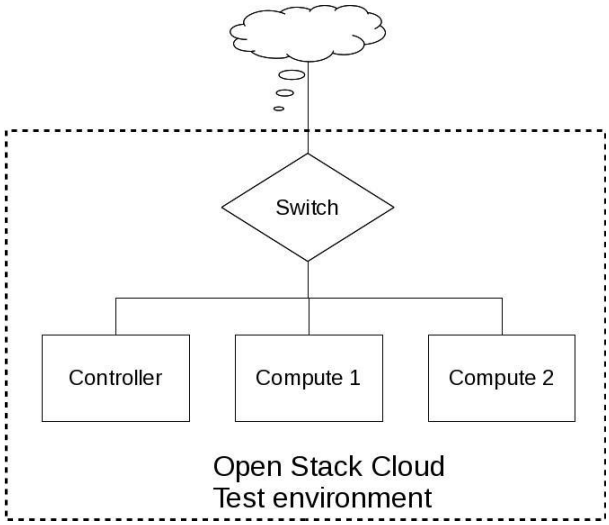

Fig. 3: Open Stack Cloud Test Environment

Figure. 3 shows the test environment for open stack cloud . It contains 1 controller and 2 compute servers, each one has 16 CPU and 64 Gb memory. Each compute host is capable of hosting 20 VMs. The server are connected to the 1 Gbps switch. The following graphs shows the bandwidth utilization with and without Network aware VM placement.

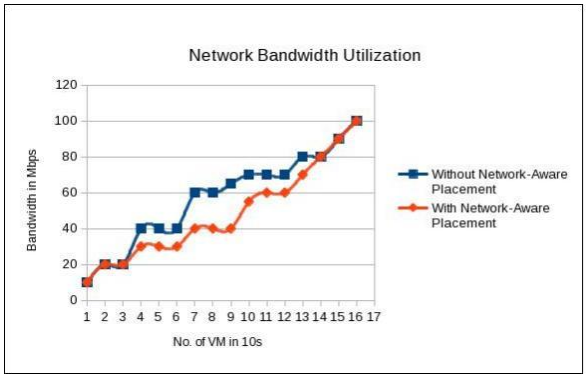

Fig. 4: Graph Network Bandwidth utilization – NetworkAware vs Without network Aware VM placement

If the number of VMs less them 40, the Network-aware placement does not make any difference. If the no. of VMs are greater than 40 and less the 150, the network aware placement makes difference. The VM migration can be performed to save the bandwidth. If no. of VMs are more than 150. The environment becomes more congested; hence there is much less opportunity to migrate the VM.

7. CONCLUSIONS AND FUTURE WORK

Adding the network awareness in the VM placement significantly optimizes the cloud network. The proposed design is easily integrated in the OpenStack cloud platform. The proposed solution eliminates the drawback of existing solutions covered in section 2 At the writing of this paper, the full implementation and the result testing is not complete.

8. REFERENCES

[1] Kevin Jackson , Cody Bunch "OpenStack Cloud Computing Cookbook Second Edition Kindle Edition"

[2] M. Steiner, B. G. Gaglianello, V. Gurbani, V. Hilt, W. Roome, M. Scharf,and T. Voith, "Network-aware service placement in a distributed cloud environment," in Proc. ACM SIGCOMM, 2012."

[3] Zhenyu Wu, Guofei Jiang, and Haining Wang,Yueping Zhang, Vishal Singh, "Automating Cloud Network Optimization and Evolution," IEEE journal VOL. 31, NO. 12, DECEMBER 2013

[4] David Breitgand, Amir Epstein, Alex Glikson, "Network Aware Virtual Machine and Image Placement in a Cloud," 9th CNSM and Workshop at IFIP DECEMBER 2013

[5] H. Ballani, T. Karagiannis, P. Costa, and A. Rowstron, "Towards predictable datacenter networks," in Proc. ACM SIGCOMM, 2011, pp. 242–253.

[6] X. Meng, L. Zhang V. Pappas, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in IEEE INFOCOM, San Diego, CA, USA, March 2010.

[7] Fie Xu ,Fangming Liu, " Heterogeneity and Interference-Aware Virtual Machine Provisioning for Predictable Performance in the Cloud" IEEE Transaction OCT 2015Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. .

[8] P. Leitner and J. Cito, "Patterns in the Chaos a Study of Performance Variation and Predictability in Public IaaS Clouds," in Proc. of WWW, May 2015.

[9] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, "Managing Performance Overhead of Virtual Machines in Cloud Computing: A Survey,State of Art and Future Directions," Proceedings of the IEEE, vol. 102, no. 1, 2014. Heterogeneity of Public Clouds," IEEE Transactions on Cloud Computing, vol. 1, no. 2, 2013

[10] K. LaCurts, S. Deng, A. Goyal, and H. Balakrishnan, "Choreo: Networkaware task placement for cloud applications," in Proc. ACM IMC, 2013.

[11] B. Hu and H. Yu, "Research of scheduling strategy on OpenStack," in Pro. CLOUDCOM-ASIA, 2013.