Providing Network Performance Isolation in VDE-based Cloud Computing Systems

Vijeta Rathore[1)], Jonghun Yoo[1)], Jaesoo Lee[1)] and Seongsoo Hong[1), 2), 3)]

[1)] School of Electrical Engineering and Computer Science

[2)] Department of Intelligent Convergence Systems,
Graduate School of Convergence Science and Technology

[3)] Advanced Institutes of Convergence Science and Technology
Seoul National University, Republic of Korea
email:{vijeta, jhyoo, jslee, sshong}@redwood.snu.ac.kr

Abstract—**In a cloud computing system, virtual machines owned by different clients are co-hosted on a single physical machine. It is vital to isolate network performance between the clients for ensuring fair usage of the constrained and shared network resources of the physical machine. Unfortunately, the existing network performance isolation techniques are not effective for cloud computing systems because they are difficult to be adopted in a large scale and require non-trivial modification to the network stack of a guest OS. In this paper, we propose a performance isolation-enabled virtual distributed Ethernet (PIE-VDE) to overcome such difficulties. It is a network virtualization software module running on a host OS. It intends to (1) allocate fair share of outgoing link bandwidth to the co-hosted clients and (2) divide a client's share to the virtual machines owned by it in a fair way. Our approach supports full virtualization of a guest OS, ease in wide scale adoption, limited modification to the existing system, low run-time overhead and work-conserving servicing. Experimental results show the effectiveness of the proposed mechanism. Every client received at least 99.5% of its bandwidth share as specified by its weight.**

Keywords-Network performance isolation; cloud computing; Virtual Distributed Ethernet (VDE); proportionally fair resource allocation

I. INTRODUCTION

Cloud computing has become an important computing paradigm in the Internet as it evolves to an effective means for seamless information retrieval and service provision over the Internet. It has already been widely deployed in data centers for web-hosting, data processing and utility computing. Often, such cloud data centers are constructed at a huge scale and requested to service a high churn of clients. This incurs serious technical challenges to the cloud providers in terms of efficient hardware resource sharing, fair resource allocation, and reliable and secure services. System virtualization technology which was originally developed in 1960's for mainframe computers has been brought up to directly address many of such technical issues. System virtualization enables multiple existing operating systems, often referred to as a guest OS, to share a physical machine by providing them with an illusion of exclusive access to imaginary hardware called a virtual machine (VM). As a result, it has been successfully used in enterprise and desktop computing domains for the cost-effective sharing and maintenance of computing resources.

System virtualization is often subdivided into CPU, memory and I/O virtualization since different types of resources require distinct techniques. Virtual Machine Monitors (VMMs) such as Xen and VMware ESX server provide mechanisms for CPU, memory and disk virtualization. In case of network, virtual networks such as Virtual Distributed Ethernet (VDE) [1] provide multiple abstracted networks from the same physical network infrastructure. Particularly, VDE realizes the packet forwarding, switching and routing functionalities of the virtual network fabric.

Unfortunately, ensuring the network performance of cloud computing applications has not been of much concern until recently although network performance metrics such as throughput, delay and packet loss rate show large variations with the bandwidth usage pattern of other co-residing VMs [2]. The fundamental steps in this direction are the minimization of network performance interference and providing adequate network resources. While the latter necessarily involves hardware changes at a huge scale, the former can possibly be put into effect with software modifications. With the increasing scale and complexity of the network, the performance isolation has become essential for optimal and fair usage of the constrained and shared network resources.

In this paper, we address the network performance isolation problem in the context of cloud computing. The network performance interference in the cloud network occurs when clients do not get the required amount of bandwidth due to excessive network bandwidth usage by other clients. Some users of the Amazon EC2 servers have reported of pronounced variations in delay and throughput due to "noisy neighbors" who run highly network intensive loads [3]. As performance of cloud computing applications is highly affected by the network performance, it is vital to ensure a client's network bandwidth requirement. The client can either specify the network requirement in the form of shares, or if not directly specified it can be interpreted by the cloud provider from its computing requirements.

In the literature, there are some techniques for network performance isolation, such as Transmission Control Protocol (TCP)-based techniques, Quantized Congestion Notification (QCN) [4], and Class of Service (CoS) technologies [5]. However, these are not quite effective for the cloud computing network as explained below.

A TCP-based solution, Seawall [6], provides a network performance isolation solution based on end nodes. It sets up TCP-like tunnels between each pair of communicating VMs or applications that nest the transport layer protocol and control flow rate while ensuring fair resource usage on every network link. In order to avoid performance issues associated with nested TCP control loops such as deteriorated throughput and increased delay in lossy networks [7], it modifies the network stack to defer congestion control to Seawall's shim layer. The need for changes to the network stack of the guest OS precludes this solution from working on the full-virtualization systems.

The QCN is a congestion notification protocol for cloud data center networks. This protocol also has some shortcomings. First, it works only within a layer-2 domain. Second, it necessitates hardware or software implementation on every network node, making it less flexible to changes and difficult to implement in a huge network.

The CoS technologies, namely, 802.1p Layer 2 tagging, Type of Service and Differentiated Services (DiffServ), provide quality of service among classes of traffic. They either fail to suit the huge scale of the cloud networks or cannot be effective due to lack of standard policies among network providers.

In this paper, we propose a software mechanism to provide performance isolation among the clients in a cloud network. In order to suit the huge scale of a cloud data center and optimal resource utilization, our design principles include full virtualization support, ease in wide scale adoption, limited modification to the existing system, low run-time overhead and work-conserving servicing. The mechanism is embedded in VDE at the end nodes of the network. We propose an enhanced version of the open source VDE switch we call Performance Isolation Enabled-VDE (PIE-VDE) switch to exercise the mechanism. It intercepts the traffic originating at the node, segregates it per VM and sends it to the outgoing link in a work-conserving manner such that clients get fair share of network bandwidth. It uses the VTRR proportionally fair scheduling policy [8] to multiplex bandwidth among the clients. The overhead is minimal since VTRR has an $O(1)$ time-complexity. The solution works for all network and transport layer protocols owing to the protocol agnostic nature of VDE. Also, it is scalable since it involves software modification only on the end nodes.

For validation of our mechanism, we have implemented it and performed thorough tests. The results confirm that network performance isolation among the clients is achieved by the mechanism.

This paper is organized as follows. Section 2 gives a background of the VDE to aid in understanding the solution mechanism and its implementation. Section 3 describes the target system architecture and gives the problem description. Section 4 provides the details of the solution mechanism. Section 5 validates the mechanism through a set of experiments. Section 6 concludes the paper.

II. BACKGROUND

We briefly give an overview of the VDE and its architecture as a background to our work.

A. *Overview of VDE*

VDE is an open source layer-2 virtual distributed network. We have chosen to provide our network performance isolation mechanism in VDE as it has been used by open source cloud computing platforms including Eucalyptus [9] to build private networks for a cluster of nodes.

VDE provides an Ethernet-compliant virtual network to connect VMs, applications and real machines.VDE is a virtual network with its parts being completely built in software. It is distributed in the sense that parts of the same VDE network can run on different physical machines. It supports the Ethernet protocol and is able to forward, send and route plain Ethernet packets. It provides interfaces to connect with VMs, connectivity tools as well as virtual interfaces of real systems.

B. *Architecture*

VDE is component-based software. Its components have the same functionality as the hardware components in the modern Ethernet network. The two main components are the VDE switch and the VDE cable.

The main functionality of VDE switch is to switch Ethernet packets between the ports. It manages dynamic association between physical addresses and ports using a hash table. The VDE cable is used to interconnect two VDE switches. It consists of three software components: two VDE plugs, one at each end of the cable, and an interconnection tool. The VDE plug is a program connected to a switch to convert all the traffic to a standard stream connection. The interconnection tool bi-directionally connects the streams of the two plugs.

C. *Operation of VDE*

The VDE switch associates a file descriptor to each connected port. It polls the file descriptor for any packet arrival events. When a packet arrives from a connected VM or a process, the switch receives the packet and reads the destination physical address. It then looks up the egress port in the hash table and forwards the packet to the port.

The VDE switch has two modes of operation: switch mode and hub mode. The switch mode is used for switching packets to only specific ports while the hub mode is used to broadcast packets to all the connected ports. It also supports retransmission of unsuccessfully sent packets by maintaining a separate queue. The user can enable or disable the retransmission as needed.

III. PROBLEM FORMULATION

In this section, we describe the target system architecture and formulate our problem.

A. *Target System Architecture*

The target cloud system comprises of a set of physical computing nodes. A physical node runs a host OS on which a VMM is executed. The VMM is used by the cloud provider to instantiate VMs according to clients' computing requirements. Hosted VMMs are needed in order to run VDE switch and VDE cable processes on the host OS.

The cloud network is realized using VDE as depicted in Fig. 1. Each physical node has a VDE switch that interconnects the VMs on the node and provides connection with other physical nodes. The switches are connected via cables according to the needed topology. In Fig. 1, all pairs of switches are connected.

B. *Problem Definition*

For the cloud computing system described above, our approach aims to provide network performance isolation among clients. Clients are individuals or organizations that buy computing, network and storage resources from the cloud providers. A contract is signed between a client and a cloud provider about the services received by the client.

A given physical node services a set of clients denoted by $\{c_1, c_2, \ldots, c_m\}$. Client c_i owns a set of VMs denoted by $\{v_1^i, v_2^i, \ldots, v_n^i\}$. On a particular node, client c_i's network requirement is expressed by its weight w_i. It expects a fraction of bandwidth proportional to this weight. For given outgoing link bandwidth B, the bandwidth entitled to client c_i is simply given by:

$$B_i = B \cdot \frac{w_i}{\sum_{j=i}^{m} w_j} \qquad (1)$$

A client's share of bandwidth B_i is in turn divided among the VMs owned by it as:

$$B_i = b(v_1^i) + b(v_2^i) + \cdots + b(v_n^i) \qquad (2)$$

where $b(v_j^i)$ is bandwidth used by VM v_j^i.

Our approach intends to (1) ensure proportionally fair allocation of bandwidth among the clients and (2) divide a client's share of bandwidth among the VMs owned by it in a fair way. Fairness criterion for VMs of a client is subject to the application. In most practical cases, VMs belonging to a client have similar network requirements and are given equal weights. In cases where VMs have varied requirements, different weights are assigned. In that case, the VMs should be provided with proportionally fair shares of bandwidth.

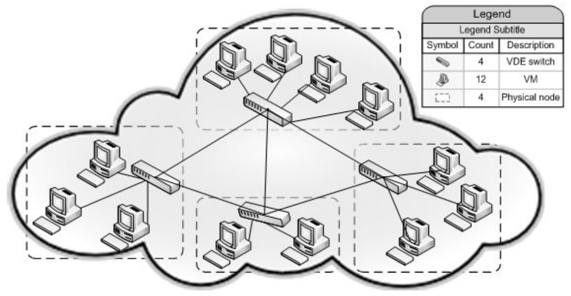

Figure. 1. Target cloud computing system architecture.

IV. THE PROPOSED APPROACH

We propose a software approach to solve the problem of network performance isolation in the cloud network. Our approach is based on a two-level link-bandwidth scheduling where the first level scheduling is done among clients and the second among VMs owned by each client. Using the specification of the clients and their VMs, our approach first generates a schedule of clients according to a proportionally fair scheduling algorithm. We particularly adopt the VTRR algorithm for its low time-complexity and improved fairness as compared to other algorithms. At the next level, our approach makes a schedule in terms of VMs. If VMs belonging to a client have equal weights, then a round-robin scheduler is used; otherwise, a proportionally fair scheduler is used in a nested fashion.

We realize our mechanism as an enhancement to the legacy VDE switch. We name it performance isolation-enabled VDE (PIE-VDE). The mechanism observes the design principles stated in Section 1. It is based on the VDE switch alone and does not need any modification to the guest OS or its applications, thus supporting full-virtualization. The change to the VDE switch is incremental and can be easily adopted through a software patch. The mechanism does not perform static bandwidth reservations and multiplexes the bandwidth share among the VMs in a work conserving manner.

We extend the legacy VDE switch by adding four components to it: (1) a VTRR scheduler, (2) a packet dispatcher, (3) per-client schedulers and (4) per-VM packet queues.

The VTRR scheduler computes a client schedule from the given set of clients and the associated weights. VTRR is a linear-time proportional share scheduler that combines the round-robin and proportionally fair scheduling approaches. A client schedule is a sequence of clients such that the number of each client appearing in the sequence is proportional to its weight. The same sequence is repeated as long as the clients and their weights remain unchanged. Thus, VTRR scheduler is invoked only once when such a change occurs.

For each client in the schedule, the packet dispatcher invokes the corresponding per-client scheduler to pick a packet and transmits it to the outgoing link. When a client has no packet to transmit, the packet dispatcher simply skips the client and continues to the next one in the schedule. This ensures the work-conserving property of our mechanism.

A per-client scheduler schedules multiple per-VM queues that belong to the same client. It is instantiated when a new

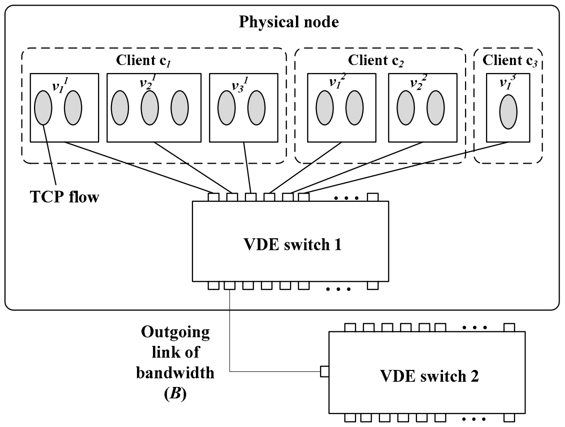

Figure 2. Cloud computing system with VDE network.

client is allocated to the physical node. It is either a round-robin or proportionally fair scheduler depending on the weights of VMs in the client. When a request arrives from the packet dispatcher, it returns a packet from a queue according to its scheduling policy.

Finally, a per-VM packet queue is given to each VM. It is instantiated when a new VM is created on the physical node. Packets transmitted by a VM are inserted at the tail of the associated packet queue.

We illustrate our mechanism through an example shown in Fig. 2. A physical node services three clients c_1, c_2 and c_3. They own three, two and one VMs and their weights are given by 5, 3 and 1, respectively. All VMs owned by the same client are given equal weights. The applications running on the VMs establish TCP flows such that all the flows pass through the outgoing link. In total, client c_1, c_2 and c_3 establish seven, four and one flows, respectively.

In the legacy VDE, there is no mechanism for allocating network bandwidth among clients. Thus, the outgoing link bandwidth is allocated to each flow by the TCP congestion control mechanism which maintains max-min fairness among the flows. Each client obtains a fraction of the bandwidth proportional to its number of flows. Specifically, client c_1 gets about a half of the bandwidth, c_2 gets a third and c_3 gets only a twelfth of the bandwidth instead of the due shares according to the weights.

In the proposed mechanism, the VTRR scheduler makes schedule (c_1, c_2, c_3, c_1, c_2, c_1, c_2, c_1, c_1) and creates per-client schedulers and per-VM queues as shown in Fig. 3. According to the schedule, the packet dispatcher invokes a per-client scheduler which in turn returns a packet from the associated queues in the round-robin manner. As a result, each client receives the outgoing link bandwidth in proportion to its weight.

V. VALIDATION

We have implemented the solution mechanism in the open source VDE switch version 2.2.3. Among four components we have added, the packet dispatcher runs in a separate thread whereas the other ones run in the main thread of VDE switch.

We performed experiments to compare the performance isolation capability of the legacy VDE and PIE-VDE. The experimental setup is shown in Fig. 4. We used a physical machine whose hardware consists of a 64-bit Intel Core2 Duo processor and 2GB of RAM. On the physical machine, the Linux 2.6.32 kernel was run as a host OS and KVM was used as a VMM. On the host OS, six VMs were instantiated and they were connected with each other via two VDE switches. The switches were connected though a couple of VDE plugs and a *dpipe* connection tool. The six VMs were owned by two clients, c_1 and c_2 as shown in Fig. 4. Their weights were given by 1 and 2, respectively. Thus, Client c_2 expects to get twice the bandwidth used by client c_1. That is, $B_2 / B_1 = 2$.

In order to generate network traffic between VMs, we ran *Iperf* version 2.0.4 which is a network performance benchmark tool [10]. It generated a burst of TCP traffic from a source VM to a destination VM. Source and destination relationship between VMs is shown in Table I. It then reported the average throughput received by each flow.

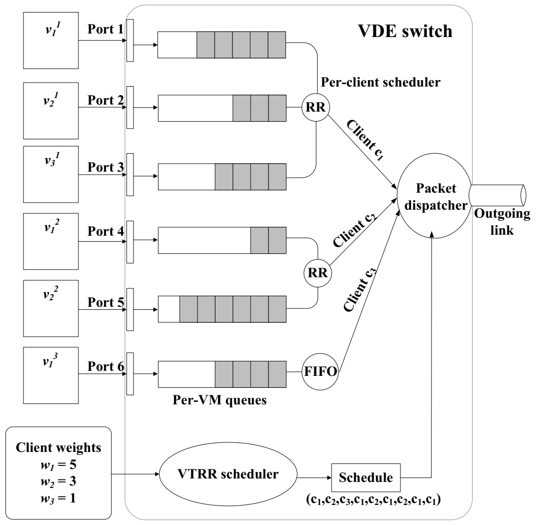

Figure 3. Component structure of the proposed mechanism.

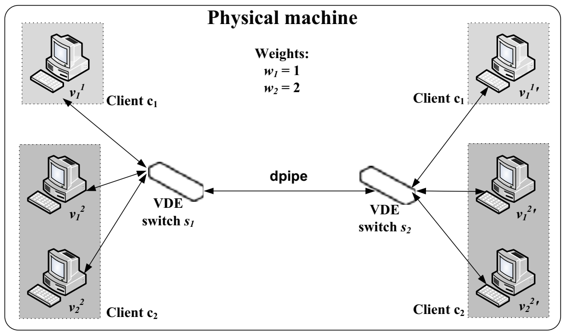

Figure 4. Experimental setup.

TABLE I. FLOWS ESTABLISHED FOR OUR EXPERIMENT

Flows	*Client*	*Source VM*	*Destination VM*
F1	c_1	v_1^1	$v_1^{1\,\prime}$
F2	c_2	v_1^2	$v_1^{2\,\prime}$
F3	c_2	v_1^2	$v_2^{2\,\prime}$
F4	c_2	v_2^2	$v_1^{2\,\prime}$
F5	c_2	v_2^2	$v_2^{2\,\prime}$

TABLE II. ACHIEVED THROUGHPUT IN MB/S

	Flows					*Clients*	
	F1	***F2***	***F3***	***F4***	***F5***	c_1	c_2
VDE	17.0	13.8	14.5	14.5	13.0	17.0	55.8
PIE-VDE	22.9	13.4	11.7	9.41	11.7	22.9	46.2

Table II shows the experimental results. We can observe that PIE-VDE allocates the link bandwidth to clients according to their weights whereas the legacy VDE gives nearly equal amount of bandwidth to each flow. In PIE-VDE, total bandwidth was 69.1 Mbps and the expected share of bandwidth for c_1 and c_2 were 23.0 and 46.1 Mbps, respectively. The actual share of bandwidth received by c_1 and c_2 were 99.5% and 100.2% of the expected ones, respectively. B_2 / B_1 was 2.01 in PIE-VDE whereas it was 3.28 in the legacy VDE. We can also observe that PIE-VDE provides each VM owned by a client with equal share of bandwidth in a fair way.

VI. CONCLUSION

We have proposed a mechanism to provide network performance isolation in a VDE-based cloud computing system. It achieves proportionally fair allocation of network bandwidth among co-residing clients on a physical node and also provides fairness among the VMs owned by a client. Our approach is distinctive as it focuses on the client abstraction which is the main entity in a cloud computing contract. It is based on the virtual switch and hence does not incur any modifications to the guest OS and the existing network infrastructure. We have conducted several experiments to validate the approach. The results showed the effectiveness of the proposed mechanism. Every client received at least 99.5% of its bandwidth share as specified by its weight.

In the future, we plan to extend the approach for the receiving end in order to make the mechanism complete and more effective. The future distributed computing systems will be larger in terms of size and complexity, thus software solutions for providing performance guarantees will become extremely important.

ACKNOWLEDGMENT

The work reported in this paper is supported in part by the Technology Innovation Program (No. 10036495) funded by the Ministry of Knowledge Economy (MKE, Korea), by Digital Media Communications R&D Center, Samsung Electronics, Co. Ltd (No. 0421-20110016), by Advanced Institutes of Convergence Technology (AICT), Institute for Green Smart Systems and by Automation and Systems Research Institute (ASRI).

REFERENCES

[1] R. Davoli, “VDE: virtual distributed Ethernet,” Proc. of IEEE/Create-Net Tridentcom 2005, Trento, Italy, pp. 213-220, May 2005.

[2] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, “Enforcing performance isolation across virtual machines in Xen,” Proc. of the ACM/IFIP/USENIX 7th International Middleware Conference, Melbourne, Australia, vol. 4290, no. 7, pp. 342-362, Lecture Notes in Computer Science, Springer, November 2006.

[3] http://alan.blog-city.com/has_amazon_ec2_become_over_subscribed.htm.

[4] R. Pan, B. Prabhakar, and A. Laxmikantha, “QCN: quantized congestion notification,” http://www.ieee802.org/1/files/public/docs2007/au-prabhakar-qcn-description.pdf, May 2007.

[5] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, “Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification,” Proc. of Internet Meas. Conf. 2004, Sicily, Italy, pp. 135–148, October 2004.

[6] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, “Seawall: performance isolation for cloud datacenter networks,” 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10), 2010.

[7] C. Kiraly, G. Bianchi, and R. L. Cigno, “Solving performance issues in anonymization overlays with a L3 approach,” University of Trento Information Engineering and Computer Science Department Technical Report DISI-08-041, Ver. 1.1, September 2008.

[8] J. Nieh, C. Vaill, and H. Zhong, “Virtual-time round-robin: an O(1) proportional share scheduler,” Proc. of USENIX Annual Technical Conference, Berkeley, CA, USA, pp. 245–260, June 2001.

[9] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The Eucalyptus open-source cloud-computing system,” the 9th IEEE Int’l Symp. on Cluster Comp. and the Grid, 2009, (CCGRID '09), Shanghai, China, pp.124-131, May 2009.

[10] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, “Iperf 1.7.0 - the TCP/UDP bandwidth measurement tool,” http://dast.nlanr.net/Projects/Iperf, 2004.