Improving Network I/O Virtualization Performance of Xen Hypervisor

Shikha R. Thakur[#1], R. M. Goudar[*2]

[#]*Department of Computer Engineering, MIT Academy of Engineering, Alandi (D) Pune (M.S.), India*

***Abstract*— Virtualization technology is the backbone of Cloud Computing. Virtualization provides efficiency, flexibility and scalability in cloud computing. Virtualization in cloud computing can be done through different virtualization platform such as VMware, Kvm, UMLinux, VirtualBox, Xen. Xen is an open source hypervisor; a virtualization tool for cloud computing that is widely used among cloud providers. Since, Xen yields poor throughput for network I/O virtualization. To overcome this problem; number of hardware and software enhancement solutions are proposed. Packet aggregation mechanism is one of the solutions that can improve the performance of driver domain based model of Xen. Packet aggregation mechanism results in increased throughput at a cost of maximized packet delay and jitter. Here is the proposed self-adaptive buffering jitter control mechanism that dynamically tunes the aggregation to achieve best trade-off between throughput and delay. It finds the mean release time of a container according to dynamic traffic load. Thus, an aggregated model of Xen would improve performance resulting in strong foundation of virtualization for cloud providers.**

***Keywords*— Xen, Network I/O virtualization, Cloud Computing, Packet aggregation, Delay and jitter, Adaptive buffering**.

I. INTRODUCTION

This Cloud [1] is an abstraction to hardware resources that maintain and manages itself. Computing that enables accessing of virtualized resources and services needed to perform functions with dynamic user demands and changing needs is termed as cloud computing. Cloud computing provides three types of services, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Virtualization is a key technology to implement infrastructure services. Virtualization is an isolation to each separate user. Implementing virtualization provides flexibility, scalability and effectiveness to the cloud.

Tools such as Kvm, UMLinux, VMware, VirtualBox, and Xen can be used to implement virtualization in cloud. Xen hypervisor, driver domain based model is an open source virtualization platform. Xen [10, 11] is a hypervisor providing services that allow multiple virtualized operating systems to execute on a single computer hardware concurrently. Xen hypervisor provides a strong foundation of virtualization to cloud providers. Hypervisor is a software layer that creates runs and manages virtual machines (VMs). Hypervisor layer lies between physical and operating system. Hypervisor were first implemented for computing intensive application and not for the network intensive application. Thus, a hypervisor exhibits poor network I/O virtualization performance [8]. Network I/O virtualization is essential to provide connectivity to the virtual machines. However, current implementation of VMMs does not provide high enough throughputs especially when the applications running on the different virtual machines within the same physical machine are I/O intensive (web services, video servers, etc.). Network intensive applications are among the applications dominating the cloud based data centres today.

To improve networking performance, it is necessary to make the networking performance of the VM scale up at line rates. Packet aggregation mechanism [2] used to maximize the throughput to scale up a networking performance. Experimental evaluation of packet aggregation has been done and maximized delay and jitter is observed. For network intensive interactive communication, there must be smooth traffic. In real applications, the traffic is dynamic and is according to the dynamic user needs .The smoothness of the traffic is measured in terms of its delay and jitter. Limited buffer size jitter regulators can be used to minimize delay and jitter. Thus, an algorithm is proposed to find a release schedule for dynamic traffic rate with optimal jitter. In this paper, the aim is to minimize this delay and jitter to obtain desired throughput of system with post implementation of packet aggregation.

Remainder of the paper is brief in section. Section II gives the related work to improve network I/O virtualization. Section III gives theoretical analysis. Section IV gives design details and Section V gives brief idea about packet aggregation and focuses on proposed algorithm, self-adaptive buffering and its design details. Section VI brief about experimental evaluation and result analysis. Last section concludes the paper and mentions the future scope.

II. RELATED WORK

An Several research works [9, 10, 12] have been dedicated to the I/O performance improvement. The detailed study of research has been done and is stated as follows:

A) Large Receive Offload (LRO) [18] receives multiple TCP packets and passes it as a single larger packet to the upper layer of the network. The CPU overhead is lowered and a performance improvement is expected. An improvement at a rate of 200% is expected after implementing LRO with five

1Gb/s NICs. LRO is nothing but an aggregation mechanism, performed in the device driver of the receive side only, in the context of TCP packets reception. In virtualization, LRO can be used to transfer packets from the hardware i.e. NIC to the driver domain. Henceforth, the grant reuse as well as proposal are the ones that operating between the driver domain and the virtual machines. .LRO reduces the physical interrupts that are required to transfer packets from physical layer to driver domain. System lacks at reducing virtual interrupts.

B) XenLoop [19] is an inter-VM loopback channel that allows direct communication among VMs in the same physical machine without the involvement of unprivileged domain i.e. Dom0. Whenever two guest VMs within the same physical machine have an active communication of packet traffic, bypassing the standard data path via Dom0, VMs set up a bidirectional inter-VM data channel between themselves. An improvement from a factor of 1.5 to 6.2 is obtained in bandwidth improvement with XenLoop over the native netfront-netback corresponding to the transferred message size. It reduces the memory call among virtual machines VMs running on a single physical machine.

C) XenSockets [13] is a unidirectional communication pipe between two VMs. Based on UNIX socket implementation, It implements an inter VM communication mechanism. The implementation of XenSockets makes use of statically shared memory buffers. Instead of the traditional Xen page flipping mechanism, It uses shared memory for message passing. For example, For a message size of 4 Kb, XenSockets yields XenLoop with a throughput of 5800 Mbps. Throughput of Xen hypervisor increases greatly and is considerable. Xensocket does not support VM migration and exhibits poor transparency.

D) In [4], the authors observed that with numerous Vms sharing a single CPU, the latency experienced by each VM to obtain its CPU time slices increases and that the CPU access latency dominates the round trip time between two VMs, graduating the progress of TCP connections between Vms. To solve this problem, the authors proposed a solution Known as vFlood in which they did a small modification to the sending VMs TCP stack that essentially offloads congestion control functionality to the driver domain. The driver domain handles congestion control on behalf of the VM, therefore ensuring the compliance of TCP semantics. The throughput achieved by vFlood is almost 5 times higher than that of the vanilla Xen. For large packet transfers, vFlood improves TCP throughput by 20% to 40%. vFlood improves Xen hypervisor performance [16,17] giving higher throughput. The throughput is increased at a cost of maximized delay and jitter.

E) In [20], authors proposed optimization to minimize the I/O virtualization overhead: efficient interrupt coalescing for virtualization and virtual receive side scaling (RSS). Here, an adaptive multi-layer interrupt coalescing scheme for network I/O virtualization is used to dynamically throttle interrupt frequencies. To parallelize the backend driver and to adopt RSS to bind each thread of the backend driver to corresponding virtual CPU, virtual receive side scaling is implemented. Efficient interrupt coalescing for virtualization can mark ably minimize CPU utilization.

F) In [21] authors investigated the performance of four different algorithms for dynamically adjusting the playout delay of audio packets in an interactive packet-audio terminal application, for varying network delays and dynamic traffic. Experimental results indicate that an adaptive algorithm which explicitly adjusts to the sharp, spike-like increases in packet delay can achieve a lower rate of lost packets for both a given average playout delay as well as a specified maximum buffer size. The four algorithms differ in calculating the mean estimate of delay and variation in delay. Formula for calculating playout time is same for all the algorithms.

III. THEORETICAL ANALYSIS

All Packet buffering technique [2] introduces an additional delay to the traffic. In a system without aggregation, the mean delay and jitter of a packet varies as a function of the input rate. The delay in this system will remain constant for the drastic increase in forwarding throughput [2]. Whereas, an aggregated system, consider the following scenarios:-

Assumption: - As it is known each TCP Packet size is 65535 bytes, which is a large size for any realistic data. The Maximum Transmission Unit (MTU) that can be transferred is 1500 bytes and the container size is considered as 4096 bytes as it is one page size of shared memory.

1) When forwarding throughput reaches to its ace, there would be increase in packets average delay due to additional waiting time until the container reaches to its timeout or maximum accommodating size is reached.

2) At low input rates, containers are transferred to destination after the timeout has expired. At this rate, containers spend a long time to get full. Thus the packet delay would be increased.

3) At high input rate, the containers reach their maximum size rapidly before the timeout expire which triggers the container transfer. Henceforth, containers will fill fast and will generate small waiting packet delays. At the same time, there could be more memory call and virtual interrupts that is considerable In order to implement packet aggregation mechanism efficiently, it is necessary to tune the aggregation mechanism with the dynamic packet traffic. Here is the proposed self-adaptive jitter regulating method that can adapt the container size with the incoming traffic dynamically to achieve best trade-off between delay and throughput. Next part of this section will detail the proposed system model to minimize delay and jitter.

IV. Design

A. Platform Choice

Sr. No.	Requirements	
	Hardware	Software
1.	Intel 3rd Gen Core i5 processor	Xen hypervisor 4.1
2.	6GB RAM	Ubuntu 12.10 Desktop Version
3.	500GB or more HDD	Oracle Java
4.		Kepler Eclipse IDE 4.3
5.		Apache Jmeter 2.10
		Hazel Cast (Shared Memory)

B. Modules of the System

In the proposed system, there are two main modules "Fig. 1" - packet aggregation and self-adaptive jitter regulator algorithm. An aggregation mechanism is used to improve performance of Xen hypervisor. As it improves performance there is one disadvantage of aggregation is that it increases delay and jitter, thus to minimize delay and jitter second module i.e. self-adaptive jitter regulation mechanism has been implemented.

Fig. 1. Modular diagram for improving I/O performance

C. Architecture of Proposed System

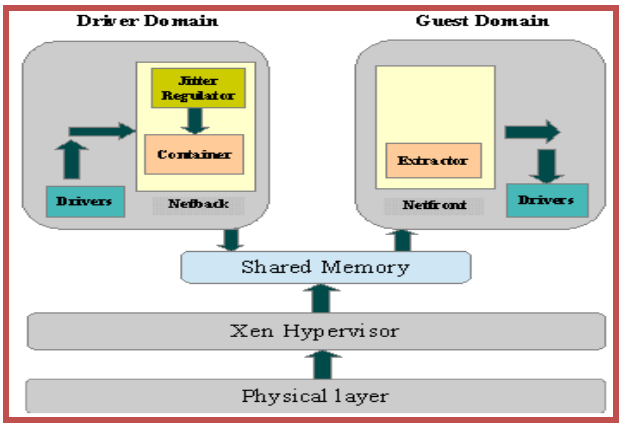

Fig. 2. Architecture of proposed system

V. Implementation Details

This proposed system is composed of two modules, packet aggregation and self-adaptive jitter mode. Packet aggregation is again divided into two sub-modules i.e. Container and Unloader. Second module implements algorithm for minimizing delay and jitter. The implementation consists of two parts:

a) It is coded to prove the correctness of algorithms. Thus it consists of simulation and the coding is done in java.

b) Once correctness is proved, the spilt device drivers can be modify and the algorithm can be embedded into it.

A. Packet Aggregation

Consider a scenario in which driver domain is communicating with virtual machines. The basic concept behind aggregation mechanism is to buffer packets in a container of fixed size and then to transfer it at once when the container size is full or buffering time of container is timeout. Container generation algorithm buffer packets and transfer container to shared memory from driver domain. Extractor will extract the packets from container in FIFO order. Author Manel Bourguiba and his team has implemented and experimentally evaluated aggregation mechanism [2] which shows that it improves throughput affecting delay and jitter.

B. Self-Adaptive Jitter Model

Consider a scenario in which a system consists of a driver domain and N number of VMs running simultaneously that communicates through shared memory. At the arrival of packet from the physical layer to the driver domain, aggregation mechanism generates a container and waits for the next packet to be queued until the maximum size is reached or timed out. Thus, the arrival rate of two packets that is to be transferred to the shared memory generates the delay and is equal to the sum of delay between N packets transferring to driver domain.

Foundation of jitter regulation is consisting of jitter regulators that use a limited-size buffer in order to minimize delay and jitter. Here, is the proposed method of finding playout time of a container. Playout time is the release time of container that can be calculated by obtaining mean of releasing time of each packets arriving to container. This idea can be clear through the following algorithm.

1) Notations:

P_i - Playout time of i^{th} packet

P_j - Playout time of subsequent packets

T_0 - Time out of container

α- is a linear weighting factor and is 0.99

D_k – Mean estimate of delay

V_k - Variation in delay i.e. Jitter

2) Algorithm:-

After connection is setup, consider first packet coming to Dom0 from physical line.

Step 1: Calculate playout time for the first packet; it would be maximum for the first time which could be equal to the timeout of the container.

$$Pi = To$$

For this condition, mean of delay and variation in delay will be zero.

Step 2: For subsequent packet, the playout time is given as:

$$Pj = Dk + (4 * Vk)$$

Step 3: Di and Vi can be find by the given formula:

$$Dk = (\alpha * Dk\text{-}1)$$

$$Vk = (\alpha * Vk\text{-}1) + (1 - \alpha)$$

Step 4: For each N subsequent packets Pj, the release time of container will be adjusting accordingly and thus container will release at a calculated mean time when it is time out or overflow.

Step 5: As one container will release; another container will be allocated and then repeat step 1 to 4 until the connection for a DomU is terminated.

IV. Experimental Evaluation and Results

Experimental test bed consists of a PC/Laptop of configuration 3rd Gen Ci5 Core Processor/ 6GB RAM/1 GB NIC/500GB HDD. Consider one system as Dom0. For setting up a Dom0 machine, it requires a Xen hypervisor to be installed first. Here the host operating system used is Ubuntu Desktop version 12.10. Any host operating system supporting Xen hypervisor can be installed. Configure Virt manager to have two or more guest operating system (DomU). Create a network bridge between DomUs by following installation guide of Xen.

Experimental performance is simulated. TCP load to the system can be provided by a GUI tool called apache Jmeter 2.10.This tool provides heavy as well as light load to the java server and get the response from the server and accordingly makes different graphs for various performance metrics. Here, two guest operating system is considered; for each of two TCP load of 500 samples is applied. First this test case is applied to only aggregated mechanism i.e. aggregation mechanism without self-adaptive jitter algorithm. This generates graph for throughput "Fig. 5" and gives average throughput of 4.187 KB/s. The same test sample when is applied to aggregation with self-adaptive jitter regulation system an improved throughput graph "Fig. 6" is obtained with average throughput of 13.349 KB/s.

Table -1 Experiment Result

Performance Metric	Packet aggregated	Jitter Regulated packet aggregated
Throughput (KB/S)	4.18781	13.3497

In given two figures graph drawn in green indicates throughput. Observing results in Fig 5 and Fig 6, it is concluded that the throughput obtained in aggregation mechanism with self-adaptive jitter regulator is improved as compared to the results obtained from aggregation mechanism.

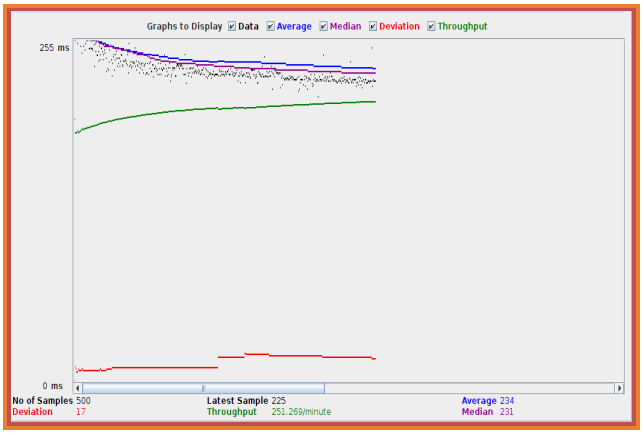

Fig. 5. Throughput graph of aggregation mechanism

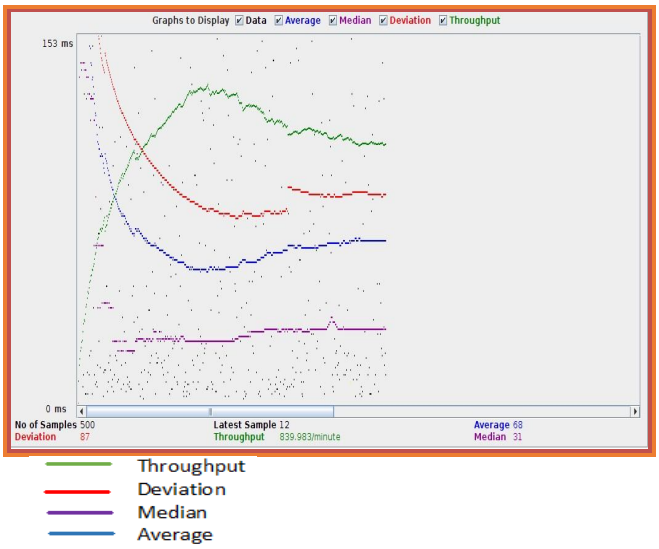

Fig. 6. Throughput graph of aggregation mechanism with self-adaptive jitter regulator.

V. Conclusion And Future Scope

Driver Domain based Xen hypervisor model is an effective, scalable, flexible and most widely used virtualization foundation for cloud service providers. Since, Driver domain based model of Xen exhibits poor I/O networking performance; it is overcome by the proposed aggregation mechanism. In this paper, aggregation mechanism overcomes the network I/O communication problem. It reduces number of memory calls per byte transfer and thus increases the throughput. Since it buffers the packets in a container of fixed size which introduces packet delay and jitter while communication when traffic is low. Henceforth, second part of implementation is for minimizing the introduced delay and jitter. Adapting the size of container with respect to the dynamic incoming traffic will reduce delay and jitter to achieve the best tradeoff between throughput and delay. Thus, packet aggregation mechanism improves the performance of driver domain based I/O virtualization in Xen hypervisor.

The proposed algorithm would be implementing in java to prove the functionality of the algorithm. Thus, the future work is to implement packet aggregation in source code of Xen hypervisor. The modification can be done in netfront.c and netback.c file located at Linux/Drivers/net/Xen/netfront.c and netback.c.

References

[1] GManel Bourguiba, Kamel Haddadou, Ines El Korbi, Guy Pujolle, "Improving Network I/O Virtualization for Cloud Computing," IEEE Transactions on Parallel and Distributed Systems, 25 Feb. 2013.

[2] M. Bourguiba, K. Haddadou, and G. Pujolle, "Packet Aggregation Based Network I/O Virtualization for Cloud Computing",Elsevier Computer Communications, Vol. 35, no. 3, pp 309-319,2012.

[3] David Hay , Gabriel Scalosub," Jitter regulation for multiple streams", 13th Annual European Symposium on Algorithms,2005.

[4] S. Gamage, A. Kangarlou, R. Kompella, and D. Xu, "Opportunistic Flooding to Improve TCP Transmit Performance in Virtualized Clouds", Proc. ACM Symp. Cloud Computing (SOCC' 11), 2011.

[5] K.K Ram, J.R. Santos, Y. Turner, A.L Cox, and S. Rixner, "Achieving 10Gb/s using safe and transparent Network Interface Virtualization", Proc. ACM SIGPLAN/SIGOPS Conf. Virtual Execution Environments (VEE' 09), 2009.

[6] V. Ramaswami, "From the matrix-geometric to the matrix exponential", Queueing Systems Theory Appl., vol. 6, pp. 229- 260, June 1990.

[7] M. Dobrescu, N. Egi, K. Argyraki, B.G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, Routebricks : exploiting parallelism to scale software routers, Proc. ACM SIGOPS Symp. Operating systems principles (SOSP 09), 2009.

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R.Neugebauer, I.Pratt, and A. Warfield, "Xen and the art of virtualization", Proc. ACM Symp. Operating Systems Principles (SOSP' 03), Oct. 2003.

[9] Xing Pu , Ling Liu , Yiduo Mei , Sankaran Sivathanu , Younggyun Koh , Calton Pu, Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments, Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, p.51-58, July 05-10, 2010.

[10] Mukil Kesavan , Ada Gavrilovska , Karsten Schwan, Differential virtual time (DVT): rethinking I/O service differentiation for virtual machines, Proceedings of the 1st ACM symposium on Cloud computing, June 10-11, 2010.

[11] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, M. Williams, Safe hardware Access with the Xen virtual machine monitor, in: Proceedings of the First Workshop on Operating System and Architectural Support for the on Demand IT Infrastructure, OASIS 2004.

[12] Jeremy Sugerman , Ganesh Venkitachalam , Beng-Hong Lim, Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, Proceedings of the General Track: 2002 USENIX Annual Technical Conference, p.1-14, June 25-30, 2001.

[13] X. Zhang, and Y. Dong, "Optimizing Xen VMM based on Intel Virtualization technology", Proc. International Conference on Computer Science and Software Engineering, 2008.

[14] Scot Rixner, Network Virtualization: Breaking the Performance Barrier, Queue, v.6 n.1, January/February 2008.

[15] Paul Barham , Boris Dragovic , Keir Fraser , Steven Hand , Tim Harris , Alex Ho , Rolf Neugebauer , Ian Pratt , Andrew Warfield, Xen and the art of virtualization, Proceedings of the nineteenth ACM symposium on Operating systems principles, October 19-22, 2003.

[16] Yiduo Mei , Ling Liu , Xing Pu , Sankaran Sivathanu, Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud, Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, p.59-66, July 05-10, 2010.

[17] Scot Rixner, Network Virtualization: Breaking the Performance Barrier, Queue, v.6 n.1, January/February 2008.

[18] A. Menon, and W. Zwaenepoel, "Optimizing TCP Receive Performance", Proc. USENIX Annual Technical Conference (USENIX' 08), 2008.

[19] J. Wang, K. Wright, and K. Gopalan, "XenLoop: A Transparent High Performance Inter-vm Network LoopBack", Proc. ACM Symp. High Performnce Parrallel and Distributed Computing (HPDC'08), 2008.

[20] Yaozu Dong, Dongxiao Xu, Yang Zhang, Guangdeng Liao, "Optimizing Network I/O Virtualization with Efficient Interrupt Coalescing and Virtual Receive Side Scaling ", Proc. IEEE International conference on cluster Computing, p.26-34, Sept. 26-30, 2011.

[21] Ramjee, R. Dept. of Comput. Sci., Massachusetts Univ., MA, USA Kurose, J. Towsley, D. Schulzrinne, Henning "Adaptive playout mechanisms for packetized audio applications in wide-area networks", INFOCOM '94. Networking for Global Communications., 13th Proceedings IEEE, Page(s): 680 - 688 vol.2, 1994.