ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 3, March 2015

Survey of Approaches to Improve Inter Virtual Machine Communication Efficiency on Xen Platform

Shivaraj Sankh[1], Prof. Varsha Priya JN[2]

Computer Engg., and information technology Veermata Jijabai Technological Institute (VJTI), Mumbai, India[1,2]

Abstract: Virtual Machines (VMs) are building blocks of today's cloud computing infrastructure. VMs provide isolation across applications and services while sharing a common hardware platform. At the same time network intensive applications, such as web services/database applications, are being consolidated into single physical platform. This leads to Network I/O workloads which are dominating in many data centers. Though strict isolation between co-resident VMs ensures security and a lot of research interest is dedicated to strengthen this feature, which undermines the potential communication channels and limits maximum achievable communication throughput between co-resident VMs. Virtual Network Interface (VNI) based communication serves the purpose of transparency however traversal through entire network stack degrades the performance when communicating VMs are co-located. Data integrity is also compromised as data might travel unprotected via an insecure path where it could be altered or intercepted. Recently proposed Inter Virtual Machine Communication (IVMC) methods for co-located VMs include shared memory, customized libraries or API. Though shared memory based approaches seems like the obvious solution, they have certain issues regarding security and transparency. None of these initiatives take security considerations into account. Unfortunately none of these solutions meet all the requirements of IVMC.

Keywords : IVMC, Virtual Machine Communication, Shared Memory Communication

I. INTRODUCTION

Virtualization has become integral part of modern Data Centers, as it enables sharing of underlying hardware re-sources and provides well defined boundaries to each application. Virtualization provides consolidation of different virtual machines on a single physical machine for power saving, migration of virtual machine for load balancing etc. Virtualization optimizes resource utilization by providing full control of resource allocation to administrator. Hypervisor or Virtual Machine Manager (VMM) is hardware allocation and management software.VMM allows each virtual machine to access the resources like CPU, disk, memory, network etc. At the same time VMM isolates different virtual machines from each other. Virtualization classification can be done by the technique with which hardware is emulated to the guest operating system (OS). They are as follows:

A. Full Virtualization
VMM controls the hardware resources and emulates it to guest OS without any modification. Almost complete simulation of the actual hardware is done by VMM, which typically consists of a guest OS, to run unmodified. One of the ways deployed to do this is binary in which non virtualizable instructions are replaced with new sequences of instructions that have the intended effect on the virtual hardware.
Ex - VMWare Workstation, VirtualBox, Kernel Virtual Machine (KVM[1]).

B. Paravirtualizaion
In para virtualization a hardware environment is not completely simulated by VMM but certain changes are made in guest operating system to adapt it to run in virtual environment. The guest OS is modified to change non virtualizable privileged instructions with hypercalls to the VMM. Thus VMM provides API to guest OS to access the hardware.
Ex - Xen

C. Hardware-assisted virtualization
Hardware-assisted virtualization improves the efficiency of hardware virtualization. It involves employing specially de-signed CPUs and hardware components that help improve the performance of a guest OS.
Ex - Intel-VT, AMD-V with hypervisors like Kernel Virtual Machine (KVM).

II. BACKGROUND

A. Xen Architecture[2]
Xen architecture is a 3-layer architecture, Hardware layer, Xen VMM layer, Guest OS layer. Hardware layer include all the available hardware resources. Xen VMM is a virtualization layer which emulates hardware resources to guest OS. Guest operating system layer contains all guest OS's installed.

In Xen terminology, guest operating OS is called as domain. One domain with highest privileges has direct access to all the hardware resources. This domain is called as domain-0 or dom0. Xen is booted in dom0 automatically. All other domains Don't have access to any resource directly. These domains are called as domain-U or domU. DomU need to access the hardware resources through dom0 using Xen API.

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 3, March 2015

B. Xen Network Interface
Xen virtualization provides near native machine performance through the use of para-virtualization. Xen provides virtualized network devices to each guest OS, instead of real physical network interface cards. The actual network drivers can either execute within Dom0 or within Isolated Driver Domains (IDD), which are essentially driver specific to virtual machines.

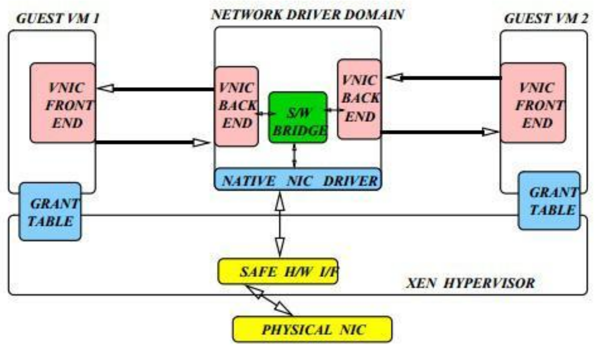

Fig. 1. Split Netfront-Netback driver architecture in Xen

Communication between Dom0 and DomU using split network-driver architecture which is shown in Figure 1 enables multiplexing of physical network card. The driver domain hosts the backend of the split network driver, called netback, and the DomU hosts the frontend, called netfront. The net-back and netfront interacts using high-level network device abstraction. The primary use of the grant table in network I/O is to provide a fast and secure mechanism for unprivileged domains (DomUs) to receive indirect access to the network hardware via the privileged driver domain. They enable the driver domain to set up a DMA based data transfer directly to/from the system memory of a DomU rather than performing the DMA to/from driver domain's memory with the additional copying of the data between DomU and driver domain. The grant table can be used to either share or transfer pages between the DomU and driver domain.

C. Inter Virtual Machine Communication obstacles [3]
Following are the major obstacles to efficient inter-VM communication:

1) Communication via TCP/IP network stack: Xen plat-form enables transparent communication across VM boundaries using standard TCP/IP sockets. This results in a significant performance penalty when communicating Virtual machines are on same host. Packet transmission and reception involves traversal of TCP/IP network stack and invocation of multiple Xen hypercalls.

2) CPU scheduling without communication awareness: If the CPU scheduler is unaware of communication requirements of co-located VMs, then it might make non-optimal scheduling decisions that increase the inter-VM communication latency. For example, the Xen hypervisor has two schedulers: the simple earliest deadline rst (SEDF) scheduler and the Credit scheduler. The SEDF scheduler makes each VM specify a required time slice in a certain period; a (slice, period) pair represents how much CPU time a domain is guaranteed in a period. The SEDF scheduler preferentially schedules a domain with the earliest deadline. It requires finely tuned parameter configuration for meeting VMs' performance requirements. On the other hand, the Credit scheduler is a proportional share scheduler with a load balancing feature for SMP systems. The credit scheduler is simple but provides reasonable fairness and performance guarantee for CPU-intensive guests.

3) Absence of real-time inter-VM interactions: Another problem with current virtualization platforms is the lack of support for real-time inter VM interactions. For example, after one VM transmits a packet to another co-located peer VM, the peer VM must be scheduled as quickly as possible to guarantee lowest possible message latency. Also when the peer VM is scheduled, the CPU scheduler within the peer VM needs to ensure that the incoming message is processed in a timely manner. The unpredictability of current VM scheduling mechanisms makes it hard to provide any form of timeliness guarantees. Since the hypervisor lacks knowledge of timing requirements of applications within each VM, it cannot meaningfully provide timing guarantees for interactions between co-located VMs. This semantic gap is the root cause of poor support for real-time inter-VM interactions.

III. INTER-VM COMMUNICATION TECHNIQUES
Two co-located VMs in virtualized environment communi-cates through their virtual network interfaces. Each packet has to traverse through an intermediate driver domain using multiple hypercalls which results in low network throughput. This overhead can be reduced with simple use of shared memory channel between the communicating domains for exchanging network packets, which require fewer hypercalls and bypasses part or whole of the default network data path. This section follows description of techniques employing shared memory communication:

A. XenSockets[4]
XenSocket uses a shared memory buffer between communicating VMs to completely bypass the network interface stack. Receiver VM allocates 128 KB pool of pages and asks the Xen hypervisor to share those pages to the sender VM. Circular buffer is used to reuse the pages. Sender writes the data to the one way communication pipe, receiver can the data immediately. Efficient data transfer algorithm which uses one shared control variable that indicates number of bytes available for writes in a circular buffer. Sender and receiver maintain local read/write offsets into the circular buffer. A new XenSocket type socket family is used to communicate across the shared memory channels. Network applications/libraries have to use this new socket type in order to take advantage of XenSocket. The core operations on XenSocket are bind() and connect(). These are mandatory steps for the sender and receiver to negotiate the control information and setup the shared memory. After the two functions are successfully called, the

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 3, March 2015

communication channel is set up and ready to exchange data packets. XenSocket provides a shared memory communication mechanism that exports a new socket-like interface. Xensocket does not provide automatic discovery of co-location and migration of communicating VMs.

B. XWay[5]
XWay provides an accelerated communication path between co-located VMs. Xway uses shared memory channel and pro-vides full binary compatibility for applications communicating over TCP sockets. No modification is required in applications
using TCP sockets in order to use the shared memory channel. As shown in Figure 2, XWay is composed of three layers:

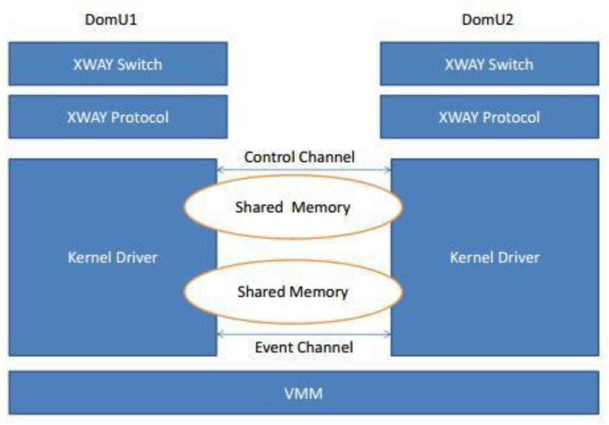

Fig. 2. Xway Architecture

Switch, protocol and driver. To reduce the development effort as much as possible, a virtual socket, called the XWay socket, is introduced. The XWay switch layer transparently switches between TCP socket and XWay protocol. It chooses which lower layer protocol should be called whenever a message is sent. At the very first packet delivery attempt, XWay switch determines whether the destination domain resides in the same physical machine by consulting a static file that lists all co-located VMs. XWay then creates a shared memory channel, and binds it to the XWay socket. In order to set up the shared memory channel and an event channel, the two VMs have to exchange the grant reference (or identity) of the shared memory and the port number of the event channel. This is normally done through another control channel. All the subsequent packets from the same connection are redirected to the shared memory channel. Otherwise, the switch layer simply forwards the requests to the TCP layer. XWay protocol layer supports TCP socket semantics for data send/receive operations such as blocking and non-blocking mode I/O. XWay defines a virtual device and a device driver to represent XWay socket.

XWay achieves high performance by providing direct shared-memory communication channel between co-located VMs. However, XWay can only support TCP communication, and requires significant changes to the Linux kernel code

C. Inter Virtual Machine Communication (IVC)[6]
IVC enables shared memory regions through the Xen grant tables. IVC consists of two components: a user space communication library and a kernel driver. The client uses IVC user library to start the connection setup. Once connection is setup, client process gets notification of the result. Internally, the IVC user library allocates a shared memory region. The user library asks the kernel driver to allow the access of particular region. The IVC library on the destination VM then maps the communication buffer to its own address space through the kernel driver. Communication establishment follows a client-server model, and communication uses producer/consumer circular ring buffer. IVC supports automatic discovery of peers on the same node. An IVC backend driver is run in the privileged Domain-0. Each parallel job that intends to use IVC is assigned a unique magic ID by the cluster administrator. All registered domains with the same magic ID form a local communication group in which processes can communicate through IVC. When a computing process initializes, it notifies the IVC library of the magic ID through a function call. This process is then registered to the backend driver. Assignment of this unique magic ID across the cluster can be provided by batch schedulers though the paper does not provide additional details A communication coordinator is introduced to handle migration via dynamically creating and tearing down IVC connections as the VM migrates. The communication coordinator keeps lists of channels and outstanding packets that are being actively used on the same host. IVC kernel driver gets a callback from the Xen hypervisor once the VM is about to migrate. It then notifies the IVC user library to stop writing data into the send ring to prevent data loss during migration; after that, the IVC kernel notifies all other VMs on the same host through event channels that the VM is migrating. IVC then tells user programs that the IVC channel will be torn down due to migration. Finally, IVC un maps the shared memory pages. Once migrated, IVC will be available to peers on the new host.

D. MMNet[7]
MMNet differs from other approaches in that it avoids copying of data across VM kernels by mapping the entire kernel address space of one VM into the address space of its communicating peer VM with read only access. MMNet eliminates data copies and hypervisor calls in the critical path by mapping in the entire kernel address space of the peer VM. Relaxation of memory isolation between VMs compromises security aspect. Although a bug in one VM cannot corrupt the peer VM’s memory because of the read-only mapping, it could potentially crash the peer VM This allows MMNet to make its link-layer driver a loadable kernel module without affecting the behavior of other system functions. MMNet implements and exports a standard Ethernet interface. MMNet interjects at the link-layer by updating routing tables. Since this is the lowest layer in the protocol stack, most of the OS subsystems and all the user applications can work transparently.

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 3, March 2015

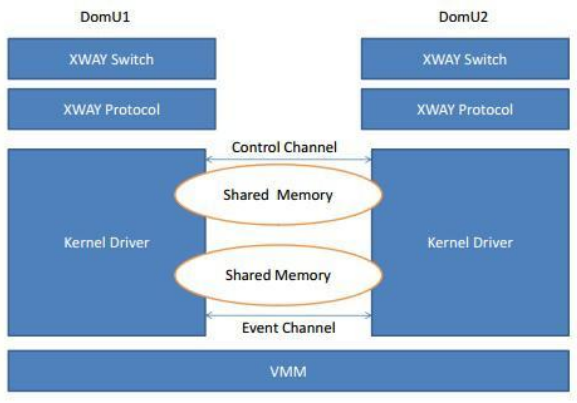

Fig. 3. Xenloop

VM discovery does not require a coordinating MMNet module in Domain 0. Each joining VMs write to global XenStore directory to advertise its presence. Each member VM watches the directory for membership updates.

When a VM is created, new MMNet connection channels are established dynamically between the new VM and other co-located VMs and new IP routing table entries are added to route traffic to the MMNet Ethernet interface. Within a VM, running applications can seamlessly switch to (or from) the MMNet connection path by updating the IP routing tables appropriately.

VM destruction will trigger removal of the routing table entries and tear down of the MMNet channel. Within the MMNet channel, data is encapsulated in a scatter-gather (SG) array. The shared meta-data segment which contains pointer to SG array and length is similar to Xen I/O rings which uses circular buffer consumer/producer algorithm.

Translation between sk buff and SG array happens before packet send and after packet receive, similar to the mechanism in netfront but without copying and page flipping. Security can be a major concern for MMNet when the communicating VMs cannot fully trust each other.

E. Xenloop[8]

Xenloop provides direct communication channel between two co-located VMs without the intervention Domain-0, along the data path. Xenloop can be used with existing network applications without any code modification, compilation or linking again.

There is no need to modify guest OS code or Xen hypervisor code to use it with Xenloop since it works as a self contained Linux kernel module. Guest VMs with Xenloop can automatically detect and setup/close XenloopChannels as needed. Xenloop also supports migration of Guests without disrupting ongoing network communications, as well as it supports switching between the standard network channel and the Xenloop channel. Using a handshake protocol dynamic connection setup is done between communicating co-located VMs. Figure 3 shows the core of Xenloop module which is a high-speed bidirectional inter-VM communication channel which is further divided into three components.

Two unidirectional first-in-first-out (FIFO) data channels and one bidirectional event channel. The two FIFO channels are set up using the inter-domain shared memory facility. Event channel notifies the presence of data channel to endpoints using 1-bit event notification mechanism. The Xenloop module contains a guest-specific software bridge that is capable of intercepting every outgoing packet from the network layer in order to inspect its header to determine the packet's destination. Netfilter hook mechanism provided by Linux is used to perform packet interception.

IV. CONCLUSION

We discussed major obstacles as well as different approaches towards efficient Inter Virtual Machine Communication. Shared memory approach improve throughput and reduce latency as it bypasses the network stack and works with fewer hypercalls. MMNet avoids shared memory with direct memory reference however it undermine the security aspect by isolation relaxation. Though lot of research interest is dedicated towards IVMC, Security aspect is largely neglected. Synchronization, live migration support and application transparency need to be embedded in a single techniques with enhanced security features.

REFERENCES

[1] Avi Kivity, Yani Kamay, Dor Laor, Uri Lublin, and Anthony Liguori, KVM: the Linux Virtual Machine Monitor, http://www.kernel.org. Ot-tawa, Ontario Canada. June 27th-30th, 2007.
[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neuge-bauer, I. Pratt, and A. Warfield, ”Xen and the art of virtualization,” in SOSP ’03: Proceedings of the nineteenth ACM symposium on Operating systems principles. New York, NY, USA: ACM, October 2003, pp.164-177
[3] Jian Wang, Survey of State-of-the-art Inter-VM communication Mecha-nisms, Research Proficiency Report Binghamton University, 2009.
[4] Suzanne McIntosh, IBM T. J. Watson Research Center, ”XenSocket: Interdomain transport for VMs”, Xen Summit 2007.
[5] Kangho Kim, Cheiyol Kim, Sung-In Jung, Hyu-Supn Shin, and Jin-Soo Kim. Inter-domain socket communications supporting high performance and full binary compatibility on Xen. In Proceedings of the fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pages 11-20, 2008.
[6] W. Huang, M. Koop, Q. Gao, and D.K. Panda. Virtual machine aware communication libraries for high performance computing. In Proceedings of SuperComputing, Reno, NV, Nov. 2007
[7] Kiran Srinivasan Prashanth Radhakrishnan. Mmnet: An efficient inter-vm communication mechanism. In Proceedings of Xen Summit, 2008
[8] Kiran Srinivasan Prashanth Radhakrishnan. Mmnet: An efficient inter-vm communication mechanism. In Proceedings of Xen Summit, 2008.
[9] Jian Wang, Kwame-Lante Wright, and Kartik Gopalan. Xenloop: A transparent high performance inter-VM network loopback. In Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC), pages 109-118, 2008.