Residency Aware Inter-VM Communication in Virtualized Cloud: Performance Measurement and Analysis

Qi Zhang[1], Ling Liu[1], Yi Ren[1,2] Kisung Lee[1], Yuzhe Tang[1], Xu Zhao[1], Yang Zhou[1]

[1]Georgia Institute of Technology, Atlanta, Georgia, USA, 30332
[2]National University of Defense Technology, Changsha, P. R. China, 410073

***Abstract*—*A known problem for virtualized cloud data centers is the inter-VM communication inefficiency for data transfer between co-resident VMs. Several engineering efforts have been made on building a shared memory based channel between co-resident VMs. The implementations differ in terms of whether user/program transparency, OS kernel transparency or VMM transparency is supported. However, none of existing works has engaged in an in-depth measurement study with quantitative and qualitative analysis on performance improvements as well as tradeoffs introduced by such a residency-aware inter-VM communication mechanism. In this paper we present an extensive experimental study, aiming at addressing a number of fundamental issues and providing deeper insights regarding the design of a shared memory channel for co-resident VMs. Example questions include how much performance gains can a residency-aware shared memory inter-VM communication mechanism provide under different mixtures of local and remote network I/O workloads, what overhead will the residence-awareness detection and communication channel switch introduce over the remote inter-VM communication, what factors may exert significant impact on the throughput and latency performance of such a shared memory channel. We believe that this measurement study not only helps system developers to gain valuable lessons and generate new ideas to further improve the inter-VM communication performance. It also offers new opportunities for cloud service providers to deploy their services more efficiently and for cloud service consumers to improve the performance of their application systems running in the Cloud.*

1. Introduction

Virtual machine monitoring (VMM) technology is pervasively deployed in Cloud based data centers to allow multiple virtual machines (VMs) to share the conventional hardware of a modern computer in a well isolated and resource managed fashion. On one hand, this enables VMs to be utilized as independent computing servers regardless whether they are co-located on a single physical host machine (local mode) or separated on two different host machines (remote mode). On the other hand, network I/O traffics between VMs are witnessed as one of the dominating costs for massive parallel computation jobs, such as Hadoop MapReduce jobs, in cloud based data centers. We argue that residence-aware inter-VM communication mechanisms present an opportunity to address the inter-VM communication inefficiency for data transfer between co-resident VMs.

Several recent research and development efforts have been made in building a shared memory based method to speed up the communication between co-resident VMs. Such residency-aware inter-VM communication mechanisms provide new opportunities to improve the network I/O performance between co-resident VMs from two perspectives: First, shared hardware resources among co-resident VMs, such as memory, can shorten the inter-VM communication paths among co-located VMs (i) by bypassing part of the native network stack in the operating system kernel on both sending and receiving VMs and (ii) by avoiding the involvement of VMM in co-resident VM communications. Furthermore, shorter communication paths can also reduce the possibilities for the network data to be corrupted or lost in transit.

Even though it is well recognized that such residency-aware inter-VM communication mechanisms hold the potential for speeding up the co-resident VM communication and thus the overall network I/O performance in Cloud based data centers [4,8,9,10,11,12], there lacks of in-depth measurement study with quantitative and qualitative analysis on challenging issues that are critical for real world deployment in operational Cloud based data centers. Here are some examples:

- How much performance gains can a residency-aware shared memory inter-VM communication mechanism provide under different mixtures of local and remote network I/O workloads?
- What is the overhead of supporting residence-awareness detection and communication channel switch to the remote inter-VM communication?
- What factors may exert significant impact on the throughput and latency performance of such a shared memory channel?

There are a number of reasons of *why* these questions are interesting and representative, and the answers to these types of questions demand for performance-based measurement study and analysis. First, most existing efforts [2,3,4,11,12] choose to use only co-resident I/O workloads to evaluate the effectiveness of their respective implementation. Very little efforts have been put forward to measure the effectiveness and efficiency of using shared memory channel in the presence of different mixtures of local and remote network I/O traffics. Second, existing implementations differ from one another in terms of what level of transparency they support. For example [2,5,6,] support only OS kernel transparency, [4] supports both OS kernel and VMM transparency but requires modifications in systems call and user library, [11] supports user-level and VMM level transparency but requires modifications of OS kernels, only [8,9,12] support all three levels of transparency. It is recognized that the capability of detecting residency-aware inter-VM communication and the capability of automatic switching between communicating through network stack and through shared memory channel play a fundamental role in providing all three level transparency and the support for

VM live migration and live deployment [10]. However, none has examined whether and how much the overhead of residence-aware VM detection and communication channel switch may impact on the performance of remote inter-VM communication. Finally, it is known that certain choices of communication configuration parameters may exert significant impact on the throughput and latency of such a shared memory channel, for example, different types of transportation protocols(e.g. TCP and UDP), different assignments of cores, message sizes, shared memory buffer sizes, to name a few.

In this paper we present an extensive experimental study, aiming at addressing a selection of fundamental issues outlined above. Another goal of conducting this measurement analysis is to gain first-hand insights regarding the design and implementation guidelines and better tradeoffs for developing a robust and customizable residency-aware inter-VM communication protocol based on shared memory structures. We believe that this measurement analysis not only helps system developers to gain valuable lessons and generate new ideas to further optimize the inter-VM communication performance. It also offers new opportunities for cloud service providers to deploy their services more efficiently with a higher Service Level Agreement (SLA) guarantee, and for cloud service consumers to improve the performance of their application systems running in the Cloud.

2. Overview and Background

2.1 Inter-VM Communication in Xen

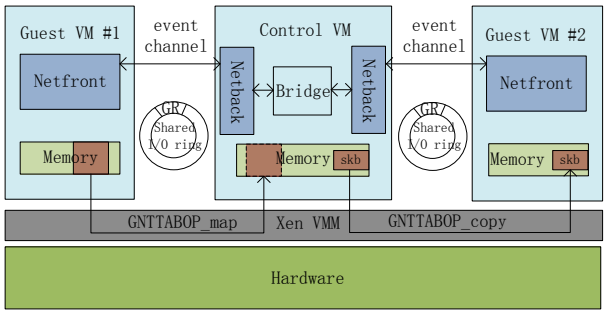


Figure 1. Xen architecture overview.

Xen is an open source x86 virtualization platform, which coordinates and manages multiple running VMs and commodity operating systems to share conventional hardware in a safe and resource managed fashion [1]. Virtual machines running on top of Xen can be divided into two categories: Control VM (dom0) and Guest VM (domU). Control VM is a privileged management domain that uses the interfaces provided by Xen VMM to guarantee the safe hardware access and reasonable manage shared resource allocation among Guest VMs. Guest VMs are unprivileged domains that running under control of dom0, and providing various services to the end users of the cloud.

Fig. 1 shows an architectural sketch of how inter-VM communication works in Xen VMM. For example, if Guest VM 1 sends a packet to Guest VM2, the frontend network driver (Netfront) in VM1 first places the grant table references (GR), which indicates the memory page that contains the packet data, into its shared I/O ring, which is a tunable shared memory buffer between Control VM and a Guest VM. Then it notifies the backend network driver (Netback) in the Control VM through event channel to fetch the GR. After getting the GR, Control VM uses the *GNTTABOP_map* interface provided by Xen VMM to map the memory page indicated by GR to its own memory, and put the data into a *sk_buf* structure, so that these data can be treated by the network stack in the Control VM. After discovering that the destination of this packet is a co-resident VM (i.e. Guest VM 2), the backend driver in Control VM gets the GR from the shared I/O ring of Guest VM 2, copies the *sk_buff* structure into the memory indicated by the GR of Guest VM 2 using *GNTTABOP_copy* interface, and finally notifies it through event channel.

2.2 Co-resident inter-VM Communication: Related Work

The host neutral abstraction of VMs supported by hypervisor technology by design treats all VMs as independent computing servers regardless whether communicating VMs are residing on the same physical host machine or not. Thus, the communication overhead between co-resident VMs can be as high as the communication between VMs located on two different hosts due to the long communication data path through the TCP/IP network stack. There are two alternative solutions to address this problem: First, we reduce the co-resident inter-VM communication latency by establishing a direct, shared memory I/O channel to allow the co-resident VMs to communicate without the involvement of VMM and virtual network driver. Second, we optimize CPU scheduling policies to make it communication aware. Most of existing works on co-resident inter-VM communication mechanisms, documented in the literature, have been based on shared memory mechanisms available in either Xen platform [1] or KVM platform [5]. The majority of the work, such as IVC [2], XWay [4], MMNet [8], XenVMC [9], XenSocket [11], XenLoop [12], have been engaged on Xen platforms to date. Nahanni [6,7] is the most known shared memory mechanism for KVM platform. Nahanni provides the shared memory API for co-resident VMs at systems calls and user library level for both host-to-guest and guest-to-guest communication. Thus, it does not support user-level transparency. Applications and library need to be modified to utilize the shared memory based inter-VM mechanism.

Table 1. Comparison of inter-VM communication mechanisms in Xen

	Implement layer	Protocol support	Open source	Transp-arency
IVC	User	TCP/UDP	No	N/A
XenXocket	Below socket	TCP	Yes	N/A
XWAY	Below socket	TCP	Yes	user
MMNet	Below IP	TCP/UDP	No	user/kernel
Xenloop	Below IP	TCP/UDP	Yes	user/kernel

Table 1 lists some important features of current implementations of co-resident inter-VM communication mechanisms in Xen. Given that Xenloop is released as an open source and it supports both TCP and UDP protocols

with three levels of transparency, in this measurement study, we choose XenLoop as the representative for measuring various performance parameters of shared memory based communication mechanisms for co-resident VMs.

3. Measurement Methodology

3.1 Measurement Goals

The ultimate goal of our measurement study is to identify the most fundamental requirements that should be considered in designing a co-resident inter-VM communication system with high performance, high availability and high transparency. Concretely, we want to provide insights and design principles for system developers to further improve the availability, the robustness and the performance of co-resident inter-VM communication mechanisms. In addition, we want to provide flexibility and customizability for cloud service providers as well as consumers to deploy co-resident inter-VM communication mechanisms for not only network I/O performance optimization but also VM live migration and live deployment support.

By high availability, we refer to availability, stability and reliability. From our experimental study, we observe that existing shared memory mechanisms fail to provide stable performance under some boundary conditions such as performance variations under different network protocols (TCP or UDP), the size of messages is extremely small or extremely large, the message arrival rate is normal or suddenly bursting, the number of co-resident VMs is relatively large. Also it is important to have fault tolerance built in, such as connection handling upon VM failure.

By high performance, we emphasize on understanding the multiple factors that may impact the network I/O performance. For example, how different assignments of cores to VMs may impact on the performance of co-resident inter-VM communication systems should be investigated, so that communicating VMs can be scheduled in a more efficient manner.

By high transparency, we advocate not only the OS kernel transparency and VMM transparency but also user level transparency. With OS kernel transparency, there will be no modification to either host OS kernel or guest OS kernel. With user level transparency, applications can take advantage of the co-resident intern-VM communication mechanisms without any modification to existing codes.

3.2 Measurement Plan and Metrics

As outlined in Section 2.3, most of existing works are developed independently and each evaluates the proposed approach using the network I/O workloads dedicated to co-resident VMs for throughput and latency comparison between the system powered with shared memory based inter-VM communication mechanism and the native system. In order to meet the above goals of our measurement study, we plan to focus our experiments and measurement analysis on the following three important categories of issues, which have not been studied in depth in previous works and are critical to both the development of next generation shared memory mechanisms and the deployment of residency-aware inter-VM communication optimization in practice.

- **Impact of Workload variation:** We want to understand how well does the residency-aware shared memory inter-VM communication mechanism perform under different mixtures of local and remote network I/O workloads. In addition, we want to measure how stable the performance of a system enabled by residency-aware shared memory mechanism is under varying workloads.
- **Overhead of Packet Interception:** To support user-level transparency, it is important to provide automated VM-residency detection and switching between local and remote inter-VM communication mode. Thus, we argue that it is important to measure and understand how much the overhead of supporting residence-awareness detection and communication channel switch may have on remote inter-VM communication performance.
- **Impact of Protocols and Core Assignments:** TCP and UDP are the most popular internet protocols in the Cloud, thus it is important that we understand how UDP and TCP protocol based applications behave under the co-resident inter-VM communication mechanism and the factors that may exert significant impact on the throughput and latency performance of such a shared memory channel. In addition, multicore computing is pervasively used in virtualized clouds, thus it is important to measure and analyze how the assignments of physical cores to VCPUs of co-resident communicating VMs may impact inter-VM communication performance.

Performance Metrics: In order to answer the above questions, the following metrics will be used in our measurement study. They are collected using Xenmon [13] and XenOprof [7].

- **Aggregate throughput (Mbps):** This metric measures the total throughput of all the network threads which generated by the benchmark in the VM, no matter the destination is a co-resident VM or a remote one. It references the overall network performance between the two communicating machines.
- **Transactions per second (#tran/sec):** This metric quantitatively measures the number of transactions between the client and server in a unit of time (second). A transaction is defined as the exchange of a single request and a single response. Based on this transaction rate, one can infer one way and round-trip average latency.
- **CPU utilization (%):** This is the CPU usage expressed as the percentage of the measurement interval.
- **CPU wait time (%):** This is how much time the VM spent waiting to run, or put differently, the amount of time the VM spent in the "runnable" state, but not actually running.
- **CPU block time (%):** This is how much time the VM stays blocked, or put differently, the amount of time the VM spent not needing/wanting the CPU because it was waiting for some event such as I/O.
- **VM switches per second (#/sec):** This metric is the number of VM context switch that has happened on all the physical cores in a unit of time (second). VM context

switch happens when a VM uses up its current CPU time slice or its execution is preempted by another VM.

- **CPU time per execution (μs/sec):** This metric measures the average CPU time in which a VM could execute once it is scheduled.

3.3 Experimental Environment

Our experimental environment includes two host machines, both of them are DELL Optilex 755 with a four core 2.4GHz Intel Core Quad Q6600 CPU(32 KB L1 cache and 8MB L2 cache), 3 GB 667MHz DDR2 memory, two 250 GB disk, and a Intel 82566DM-2 Gigabit network interface card. Xen-3.2.0 and Linux Xen Kernel 2.6.18.8-Xen are used as the underline system. Each VM is assigned 512MB memory and 1 VCPU, the default CPU scheduler (Credit Scheduler [14]) is configured.

Network traffics between two VMs are generated by netperf [13], which is a networking performance benchmark that provides tests for both unidirectional throughput and end-to-end latency. We modify the CPU entry in VM's configure file to manually pin the VCPU of the VM to a specific physical core.

4 Measurement Results and Analysis

In this section, we report three groups of measurement results and provide discussions on each in terms of guidelines for further improvement and better utilization of co-resident inter-VM communication mechanisms. The first group of experiments evaluates how mixtures of local and remote network I/O workloads may impact on the performance of the shared memory based protocol. The second group of experiments will evaluate the impact of the packet interception overhead for residence-aware VM detection may have on the performance of remote inter-VM communication. Finally, we analyze how the varying message sizes and different core assignment policies could impact the performance of UDP and TCP applications. .

4.1 Performance Impact of Workload Variation

In Cloud data centers, there are three types of inter-VM communication workloads: (i) VMs communicate with their local peers only, (ii) VMs communicate with their remote peers only, and (iii) VMs often have to communicate with both local VMs and remote VMs through network stack simultaneously to accomplish an application task (e.g., the shuffle task in some Reduce job may need to access outputs of different Mappers). However, none of the existing work has evaluated how shared memory based co-resident VM communication mechanism will perform under mixed workload (combination of local and remote communication) and pure remote workload. Thus, in this subsection, we setup experiments to measure and compare the performance of these two kinds of workload variations between a native system and a system with a shared memory inter-VM communication mechanism enabled. We generate different combinations of both local and remote inter-VM communication workloads using netperf, the percentage of remote communication in the mixed workload varies from 100% (5:0), 80% (4:1), 60% (3:2), 40% (2:3), 20% (1:4), and 0% (0:5) respectively.

Fig. 2 shows a comparison of the aggregate UDP throughputs measured under the varying mixed workloads for the native system that is running on Netfront/netback, and the system powered by the shared memory based co-resident inter-VM communication mechanism (Xenloop in this case). The message size used in this reported experiment is 1KB. We use 1:4 in X-axis to denote 20% remote and 80% co-resident inter-VM communications.

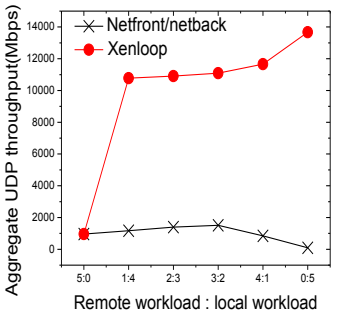


Figure 2. Aggregate UDP throughput of mixed workload

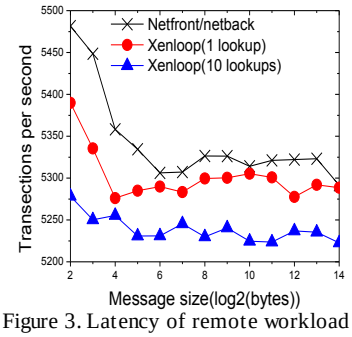


Figure 3. Latency of remote workload with various times of lookup operation

We discuss two observations: First, if the workloads are all between remote VMs, the aggregate throughputs of the two systems are almost the same, with Netfront/netback at 956.86Mbps and Xenloop at 956.11 Mbps. This indicates that the shared memory based co-resident inter-VM communication mechanism does not introduce visible performance degradation on remote inter-VM network workloads in this scenario, where the message size is fixed to 1KB. This motivates us to run next set of experiments to evaluate remote inter-VM communicating performance by varying the message size from 4 Bytes to 64K Bytes.

Second, with the increase of the percentage of co-resident inter-VM communication in mixed workload, aggregate throughput in Xenloop increases from 956.11Mbps to 13668.00Mbps as the percentage of local inter-VM communication workloads increases from zero percent to 100%. Interestingly, even when only 20% of the mixed workload is actually local between co-resident VMs, we observe that the aggregate network throughput is increased by orders of magnitude, 1027% from 956.11 Mbps to 10779.01Mbps. This indicates that once the shared memory inter-VM communication option is turned on, the overall inter-VM network I/O performance will largely benefit from it.

This set of experiments confirms our argument that using shared memory inter-VM communication is a promising way to improve the performance of inter-VM communications in Cloud data centers from two aspects. First, having the shared memory based inter-VM communication mechanism co-exist with traditional network protocols will not exert significant performance overhead on remote inter-VM communication. Second, even if there exists as high as 80% remote inter-VM communication, by turning on shared memory inter-VM

communication channel, one can benefit from the obvious performance gain in aggregate VM network throughput.

4.2 Packet Interception Overhead

Decision of switching network communicating to shared memory channel is made only when one VM detects that it is communicating with another co-resident VM. To support such user level and application level transparency, packet interception becomes mandatory to accomplish the task of this automatic detection (residency-awareness) in addition to automated protocol switch between local and remote inter-VM communications. A common way to provide seamless detection and switch is to have each VM maintain the information about its co-located VMs. Take Xenloop as an example, each VM maintains a hash table, called *vmid_mac_table*, which contains the *(vmid, mac)* pairs of all the co-resident VMs. VM will first look up mac address of intercepted packet in the *vmid_mac_table* to find out whether its communicating machine is a co-resident VM. There may exist collision chains in hash table when collision happens. This collision chain may lead to high overhead because a large number of lookup operations have to be performed for each out-going/in-coming packet in order to decide whether to use shared memory communication channel.

In the set of experiments reported in this section, we evaluate the amount of performance interference brought by packet interception in the shared memory inter-VM communication mechanism. We measure two cases: 1 lookup and 10 lookup (Fig. 3 and Table 2) to understand the performance impact of collision chain in the hash table. Due to the space constraint, we only report the results collected on remote workload. We generate remote communication workload by using netperf, with message size various from 4 bytes to 64K bytes. In this set of experiment, two guest VMs and a Control VM are running simultaneously on the same host, and scheduled by default Credit Scheduler.

Table. 2 demonstrates that packet interception could bring throughput degradation to remote inter-VM communications, and the reason is relevant to two factors: message size is very small and the number of lookup operations for each network packet is high. While the message size is smaller than 1KB, the packet interception operation injures 32.7% aggregated throughput of remote inter-VM communication by maximum when the number of lookup operation is 10 for each packet. However, as the message size grows bigger than 1KB, the throughput overhead caused by packet interception becomes less pronounced at the scale of bit throughput (Mbps), no matter the number of lookup operations is 1 or 10.

Fig. 3 measures the transaction per second throughput with varying message sizes. This allows us to analyze the per transaction latency overhead of remote inter-VM communication introduced by packet interception. There are two interesting observations. First, the native system where there is no packet interception and thus no lookup cost, has the highest transaction per second throughput for all sizes of messages, and Xenloop with 10 lookups in packet interception performs much worse compared to Xenloop with 1 lookup. Compare Fig.3 with Fig. 2, we conclude that the overhead of packet interception is less visible in terms of bit throughput (bandwidth in Mbps) but more pronounced when measuring the number of transactions per second throughput, which indicates that there is some overhead of packet interception on the latency of each transaction (sending and receiving a message of certain size) between two remote VMs. When the number of lookup operations for each packet interception grows to 10, the latency (round trip time per transaction) increases for all sizes of messages. This shows that packet interception can contribute to the network latency if not managed adequately.

The insight we gained from this set of experiments is two folds. First, the design of an efficient data structure to record and lookup for the co-resident VMs is an important issue for minimizing the overhead of packet interception used in the co-resident inter-VM communication mechanism. This is especially important because the mixed workloads are the norm in most of in a cloud data centers. Second, we learned that packet interception in co-resident inter-VM communication mechanisms could bring obvious performance overhead to remote inter-VM communication if the message size is very small (e.g. less than 1K Bytes). Thus, one possible strategy is to add heuristic based optimization in the decision of when to switch between shared memory based inter-VM communication protocol and conventional network protocol. For example, when the message size is very small from a sender VM to a receiver VM (e.g., client program is sending a request to a data server), we do not turn on the shared memory protocol. However, if a VM is sending data of large size to another co-resident VM, we should switch to the shared memory based inter-VM communication mechanism.

Table 2. Performance degradation brought by packet interception

Msg size	Native (base line)	1 lookup	Perf. decrease	10 lookups	Perf. decrease
4 bytes	5.58 Mbps	5.42 Mbps	3.0%	5.04 Mbps	9.7%
16 bytes	21.7 Mbps	18,7 Mbps	13.8%	16.4 Mbps	24.4%
32 bytes	44.0 Mbps	37.4 Mbps	15%	29.6 Mbps	32.7%
64 bytes	84.3 Mbps	74,9 Mbps	11.1%	58.3 Mbps	30.8%
1k bytes	934.1 Mbps	933.9 Mbps	0.2%	929.4 Mbps	0.5%
16k bytes	953.0 Mbps	951.1 Mbps	0.2%	951.1 Mbps	0%

4.3 Shared or Separate Cores on UDP Performance

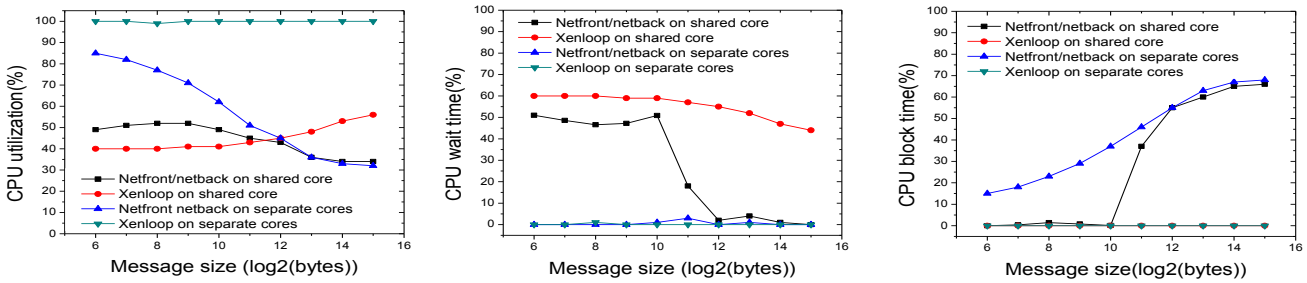
In this subsection, we focus on conducting a series of experiments to measure and compare the performance details of UDP workloads under the native Xen system and the system powered with shared memory co-resident inter-VM communication mechanisms such as Xenloop when the two co-resident communicating VMs use shared core and separate cores.

In this set of experiment, UDP workloads between co-resident VMs are generated by netperf. We conduct the experiments by varying the message size from 64 Bytes to 32K Bytes, and we also compare the UDP throughput between the co-resident VMs while they are running under different core assignment policies: executing on a shared core or running on two separate cores. Fig. 4 shows the throughput of UDP workload for both the native system and

Xenloop. Recall that each physical machine we use for the experimental setup has four cores (core0 – core3) and by default, the four cores are shared by all VMs (VM1, VM2, and Control VM). In shared core scenario, we assign VM1 and VM2 running on core0, and use default Credit Scheduler to schedule Control VM on all 4 cores. While in separate cores scenario, we assign VM1 running on core0 and VM2 running on core1, and also let Control VM to be scheduled by Credit Scheduler on all the four cores.

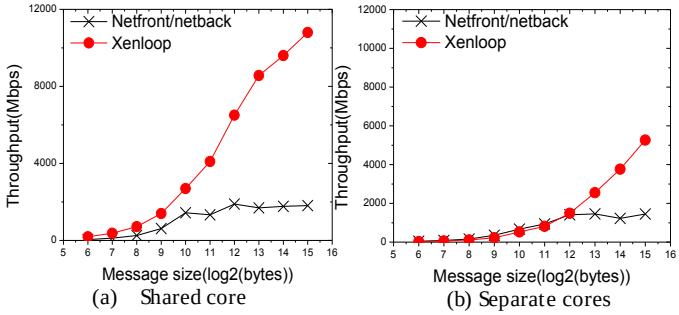
is released. (*ii*) for Xenloop, sending VM's CPU utilization is consistently 100%, because it needs neither to compete CPU resources with receiving VM nor to wait for responses from receiving VM, thus it can spend all the CPU time on sending data. *(iii)* However, for Netfront/netback, CPU waiting time of sending VM is high. But when message size increases to higher than 1KB, the CPU waiting time starts to drop, the CPU block time starts to increase. Also the CPU utilization of sending VM for Netfront/netback is dropping

Figure 5. Detail CPU metrics of sending VM



(a) CPU utilization (b) CPU wait time (c) CPU block time

From Fig. 4 we observe that the system communicating through Netfront/netback has relatively stable and consistent performance for co-resident inter-VM communication no matter the two VMs are sharing the same core or running on separate cores. However, when the shared memory based co-resident inter-VM communication mechanisms like Xenloop is measured, the UDP throughput will be better if two VMs are running on separate cores. Note that the y-axis in Fig.4 (b) uses a much higher scale than that in Fig. 4(a). In addition, while two VMs are running on a shared core, performance of Xenloop is higher than that of Xen native system with Netfront/netback only when message size is larger than 4K (=$\log_2 12$) Bytes. While in separate cores scenario, Xenloop performs better than Netfront/netback under all message size.



(a) Shared core (b) Separate cores

Figure 4. UDP throughput between co-resident VMs

To gain an in-depth understanding of the reasons behind the above observations, we use Xenmon to collect some detailed resource consumption based performance metrics. Due to the space limit, we report the measurement results of the two guest VMs in Fig. 5.

From Fig. 5(a), Fig. 5(b), and Fig 5(c), we observe three interesting facts when the two VMs are running on separate cores: (*i*) for both Xenloop and Netfront/netback, CPU waiting time of sending VM is lower than 10%. This indicates that by assigning each VM a separate core, the competition between sending VM and receiving VM on CPU as the message size increases under shared cores configuration. This shows once again that when transferring large data between co-resident VMs, the network performance will deteriorate as the message size grows. Thus, shared memory based inter-VM communication mechanisms present the opportunity to significantly enhance the network I/O performance in virtualized cloud datacenters.

Also, we measure the amount of VM context switches per second and the guest VM's CPU time per execution to find out why Xenloop does not perform well in small message scenario. Fig. 6 and Fig. 7 present the results, which show that although direct memory copy in shared memory based mechanisms such as Xenloop outperforms the operation of page mapping and memory copying in Netfront/netback, high frequency VM switches will result in serious performance degradation to co-resident inter-VM communication

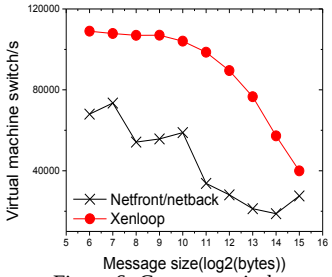


Figure 6. Context switches per second between VMs

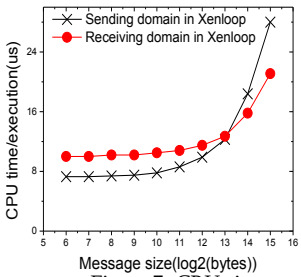


Figure 7. CPU time per execution

From this group of experiments, we learned that for UDP workloads, shared memory mechanisms like Xenloop perform clearly better than Netfront/netback when the message size is bigger (4KB or larger). A main reason is that VMs communicating through Xenloop will not be blocked by Control VM, which spends large amount of CPU time of page mapping and data copying when the message size is relatively bigger. This is also the reason why shared memory based mechanisms such as Xenloop does not do well in small message communication, when two VMs are running

in a CPU-competing mode (e.g., on the same core). One solution is to batch the notification from sending VM to receiving VM, which leads to less frequent VM context switches, and allows each VM execute a longer period of time whenever it is scheduled.

4.4 Shared and Separate Cores on TCP Performance

We below study how TCP workloads between co-resident VMs perform under varying message sizes and different core assignment polices (shared core and separate cores). The same configurations as those described in Section 4.3.

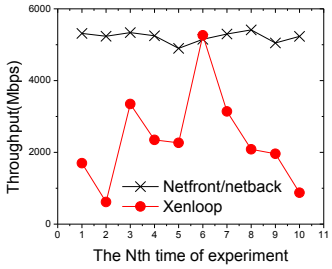


Figure 8. TCP throughput, message size = 1K Bytes, shared core

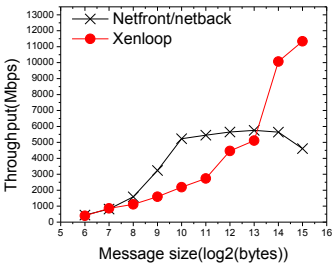


Figure 9. TCP throughput, separate cores

Fig. 8 shows the performance of TCP workload between co-resident VMs when the two VMs are running on a shared core. Each number is averaged over 10 runs. We observe that the throughput of TCP workload in Xenloop scenario is not only lower than that in Netfront/netback but also unstable. The situation improves somewhat in Fig. 9 where the two VMs are running on separate cores. We observe that the throughput of TCP workloads in Xenloop scenario is lower than that in Netfront/netback when the message size is smaller than 8K Bytes and only outperforms that in Netfront/netback when the message size is larger than 8KB. We use XenOprof to zoom into what is happening inside the system to gain a better understanding of the problem. For the convenience of description, we denote the point which represents the highest Xenloop performance in Fig. 8 HIGH_CASE, and denote the lowest of that LOW_CASE.

We first measure the execution time of functions in both HIGH_CASE and LOW_CASE, and list some of the functions whose execution time varies dramatically between these two cases. Table 3 presents that in sending VM, the execution time of *tcp_ack()*, which is called whenever a segment with an ACK arrives, in LOW_CASE is nearly 9 times as high as that in HIGH_CASE. In Xenloop, the arrival of ACK packet will be notified to sending VM through event channel. According to Credit Scheduler, frequent notifications to sending VM will make it preempt the receiving VM which is running on the same core. Therefore, frequency of context switches between the two communicating VMs in LOW_CASE should be much higher than that in HIGH_CASE. This can be demonstrated by Table 3 in that the execution time of functions such as *csched_schedule()* and *do_event_channel_op()* in LOW_CASE is about 9 times as high as that in HIGH_CASE.

Table 3. Execution time of functions

Function	Exe time (%) LOW	Exe time (%) HIGH	Module
tcp_ack	2.87	0.33	Xenolinux,
csched_schedule	0.39	0.03	Xen
do_event_channel_op	0.5	0.06	Xen

Then, we measure the execution time of functions while the two communicating VMs are running on separate cores. Table 4 presents the execution time of top functions in sending VM with Xenloop when message size is 1K Bytes and 32K Bytes respectively. Table 4(a) reflects that, with mechanisms like Xenloop, the most CPU consuming function in sending VM is *allocate_buffer_ring()*, which is a routine in netperf benchmark to allocate a circular list of buffer for either send or receive operations, when message size is 1K Bytes. This is because sending VM has to hold its network data in pre-allocated buffer until the corresponding ACK arrives, but the receiving VM process these small-size messages in a very low efficient manner. Thus, sending VM in Xenloop has to allocating extra buffers to accommodate new coming requests from application level frequently. Table 4(b) shows that time spend on *allocate_buffer_ring()* decreases from 9.74% to 3.99%, and time spend on data copy functions increases. This is because compared with Xenloop, performance benefit gained by batch operations in Netfront/netback diminished for large-size messages. Thus, in this scenario, large-size message communication in Xenloop outperforms that in Netfront/netback.

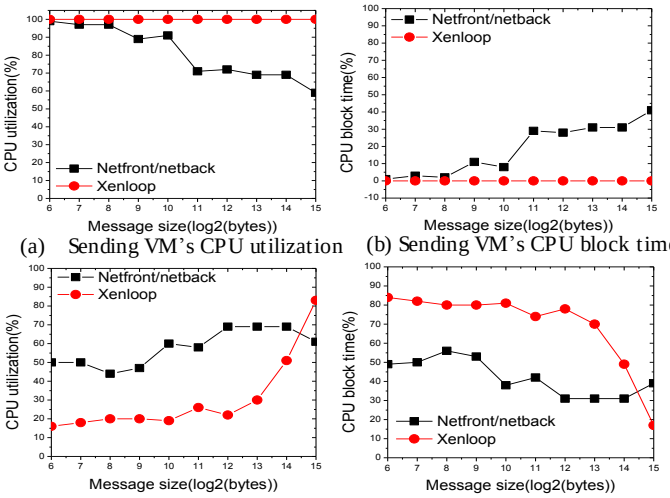
Table 4. Execution time of functions under difference message size

(a). Execution time of functions when message size is 1K Bytes

Function	Exe time (%)	Module
allocate_buffer_ring	9.74	netperf, sending VM
sha_transform	7.42	Xenolinux, sending VM
__copy_from_user_II	2.80	Xenolinux, sending VM

(b). Execution time of functions when message size is 32K Bytes

Function	Exe time (%)	Module
skb_copy_bits	21.52	Xenolinux, sending VM
__copy_from_user_II	19.11	Xenolinux, sending VM
allocate_buffer_ring	3.99	Xenolinux, sending VM



(a) Sending VM's CPU utilization (b) Sending VM's CPU block time

(c) Receiving VM's CPU utilization (d) Receiving VM's CPU block time

Figure 10. Detail CPU metrics of sending & receving VM

We also measure the CPU metrics of both sending VM and receiving VM when they are running on separate cores. Fig. 10 shows two interesting facts. First, sending VM incurs high CPU utilization in Xenloop, while in Netfront/netback, the CPU utilization of sending VM decreases as the message size increases. Recall Figure 9, Xenloop only outperforms Netfront/netback when the message size exceeds 4KB. However, when the message size is small, the CPU resource is limited for sending VM. Thus, Xenloop is not able to gain sufficient performance benefit compared to NetFront/netback. To verify our observation, we run another set of experiments which allocate sending VM two separate cores and receiving VM one core and control VM shares four cores, we observe that Xenloop performs better than the case where sending VM has one separate core. Due to space constraint, we refer readers to [15] for further detail. Second, compared with Netfront/netback, CPU utilization of receiving VM in Xenloop increases much faster as the message size increases. Also using Xenloop, Control VM is no longer the bottleneck in co-located VM communication, and receiving VM is able to work much more efficiently with larger amount of data in shared memory.

From this set of experiments, we observe two important lessons: First, for TCP workloads, it is important to use a careful configuration that can avoid co-resident VMs competing CPU resources, and offers more CPU resource to sending VM for large message size. Second, when the inter-VM communication message size is small, there is no incentive to switch on shared memory based inter-VM communication channel. Moreover, current shared memory based mechanisms improve co-located inter VM communication performance at the expense of more CPU resources, thus not switching to shared memory based mechanism when CPU intensive applications are running in either sending or receiving Guest VMs. These precautions can reduce the frequency of context switch between VMs and minimize the large amount of buffer allocation requests in applications when the two co-resident VMs are running under Credit Scheduler.

5. Conclusion

We have presented the measurement and analysis of co-resident inter-VM communication performance using local and remote workload variations, UDP vs. TCP protocol with varying sizes of messages, under shared core and separate cores. We also studied the overhead of packet interception, a core function for providing VM co-residency detection in shared memory based inter-VM communication scheme. To the best of our knowledge, this is the first in-depth measurement and analysis on critical performance properties of residency aware inter-VM communication optimization. We believe that this measurement study not only helps system developers to gain valuable lessons and generate new ideas to further improve inter-VM communication performance by designing the next generation shared memory mechanisms. It also offers new opportunities for cloud service providers to deploy their services more efficiently and for cloud consumers to improve the performance of their application systems running in the virtualized Cloud.

Acknowledgement: This work is partially funded by grants from NSF CISE NetSE program and SaTC program and Intel Science and Technology Center on Cloud Computing.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, L. Pratt, and A. Warfield, "Xen and the art of virtualization", ACM SIGOPS Operating Systems Review, vol. 37, no. 5, pp. 164-177 ACM, 2003.

[2] W. Huang, M. Koop, Q. Gao, and D.K. Panda, "Virtual machine aware communication libraries for high performance computing", Proc. of the 2007 ACM/IEEE conference on Supercomputing (SC '07). ACM New York, NY, USA. Article No. 9. 2007.

[3] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-aware virtual machine scheduling for I/O performance", Proc. of the 5th ACM SIGPLAN/SIGOPS international conference on Virtual Execution Enironment(VEE '09), Washington DC, USA, 2009, pp. 101-110

[4] K. Kim, C. Kim, S.I. Jung, H.S. Shin, and J.S. Kim, "Inter-domain socket communications supporting high performance and full binary compatibility on Xen", Proc. of the 4th ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments (VEE '08), New York, NY, USA, March 2008, pp. 11-20

[5] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux Virtual Machine Monitor", Proc. of the Linux Symposium (OLS '07), Ottawa, Canada, vol. 1, pp. 225-230, June 2007.

[6] A.C. Macdonell, "Shared-Memory Optimization for Virtual Machines", Ph.D. thesis, University of Alberta, 2011.

[7] A. Menon, J.R. Santos, Y. Turner, G.J. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the xen virtual machine environment", Proc. of the 1st ACM/USENIX international conference on Virtual Execution Environments(VEE' 05), Chicago, Illinois, USA, June 2005, pp. 13-23

[8] P. Radhakrishnan and K. Srinivasan. "MMNet: an efficient inter-vm communication mechanism", Proc. of Xen Summit. Boston, 2008.

[9] Y. Ren, L. Liu, X. Liu, J. Kong, H. Dai, Q. Wu, and Y. Li, "A Fast and Transparent Communication Protocol for Co-Resident Virtual Machines", Proc. of the 8th IEEE International Conference on Collaborative Computing (CollaborateCom'12), Pittsburgh, Oct. 2012.

[10] Y. Ren, L. Liu, Q. Zhang, Q. Wu, J. Guan, and H. Dai, "Shared Memory based Co-located VM communication Optimization: Design Choices and Techniques", Technical Report, Feb. 2013, School of Computer Science, Georgia Institute of Technology.

[11] X. Zhang, S. McIntosh, P. Rohatgi, and J. L. Griffin, "XenSocket: A high-throughput interdomain transport for virtual machines", Proc. of the ACM/IFIP/USENIX 2007 International Conference on Middleware (Middleware '07). Springer-Verlag. pp. 184-203.

[12] J. Wang, K. Wright, and K. Gopalan. "XenLoop: A transparent high performance inter-VM network loopback", Proc. of the 17th International Symposium on High Performance Distributed Computing (HPDC '08). ACM New York, USA, 2008, pp. 109-118.

[13] D. Gupta, R. Gardner, and L. Cherkasova, "Xenmon: Qos monitoring and performance profiling tool", HP Lab. Technical Report, 2005.

[14] D. Ongaro, A. L. Cox, and S. Rixner. "Scheduling I/O in virtual machine monitors", Proc. of the fourth ACM SIGPLAN/SIGOPS international conference of Virtual Execution Environment (VEE' 08), Seallte, Washington, USA, Mar. 2008, pp.1-10.

[15] Qi Zhang, Ling Liu, Yi Ren, "Residency Aware Inter-VM Communication in Virtualized Cloud Data Centers: Performance Measurement and Analysis", Center for Experimental Research in Computer Systems (CERCS), Georgia Tech, USA, Feb, 2013