An Evaluation of Multiple Communication Interfaces for Virtualized SMP Clusters

Muhammad Atif
Department of Computer Science
College of Engineering and Computer Science
The Australian National University
Canberra, ACT, 0200, Australia
muhammad.atif@anu.edu.au

Peter Strazdins
Department of Computer Science
College of Engineering and Computer Science
The Australian National University
Canberra, ACT, 0200, Australia
peter.strazdins@anu.edu.au

Abstract

Clusters with multiple CPU nodes are becoming increasingly popular due to their cost/performance ratio. Due to its many potential advantages, interest in using virtualization on these systems has also increased. Although several studies on the applicability of Xen for high performance computing have been made, most overlook the issue of multiple network interfaces. In this paper, we present an update to the state of art of Xen and give a comprehensive performance evaluation of the various network configurations that can be implemented using multiple gigabit ethernet (GigE) interfaces. We introduce new Xen network configurations, which enable the Xen guests to efficiently utilize the available network infrastructure compared to the default Xen network configurations. The evaluation of these configurations show 10-50% improvement in the NAS Parallel Benchmark suite compared to the default configurations. For these new configuration on multiple SMP nodes, the results also indicate that the need for fast intra-domain communication mechanisms is not compelling. We also detail the MPI implementations in the case of multiple GigE interfaces and their impact on a virtualized environment.

Keywords Virtualization, SMP Clusters, Communication Interfaces, Xen, MPI

1. Introduction

Cluster computing has recently seen an evolution from single processor systems to multi-core SMP systems. Commodity-off-the-shelf (COTS) SMP systems are increasingly becoming popular due to their low cost/performance ratio.

This trend has coincided with the revival of virtualization technology. The virtualization technology is receiving widespread adoption mainly due to the potential benefits of server consolidation and isolation, flexibility, security and fault tolerance. Virtualization also offers other benefits, which include development/testing of applications, live migration and load balancing. Virtualization has also generated considerable interest in High Performance Computing (HPC) community mainly for the reasons for high availability, fault tolerance, cluster partitioning and balancing out conflicting user requirements (7; 14). Another potential benefit is that the concept of live migration of virtual machines (VMs) can be utilized to maximize the throughput of a compute farm and better turnaround times of the submitted jobs. In the context of an HPC job, this means migrating a process seamlessly from one physical node to another physical node.

Despite all these advantages, the HPC community is still wary of the performance disadvantages associated with virtualization especially in the case of network I/O (8) and disk I/O (14), which are critical for many HPC applications.

This paper investigates the use of virtualization for SMP/multicore clusters with multiple network interfaces, where one can effectively partition the cluster to one CPU per virtual machine (VM). This will not only make cluster administration easier, but will enable better scheduling of jobs, as the minimum grain of migration will be a process.

In this paper, we present an update to the state of art of Xen with special emphasis on HPC applications and analyze the inter- and intra-domain communication characteristics in the current version of Xen. Using micro- and application-level benchmarks, we give a performance evaluation for a virtualized cluster for network configurations which utilize multiple Gigabit Ethernet (GigE) interfaces. These configurations include VMM by-pass and various Xen network bridge configurations, some of which are novel, using two popular implementations of MPI, namely OpenMPI (OMPI) and MPICH. Due to differences in their use of multiple interfaces and routing mechanisms, we show that different configurations are optimal for each.

The rest of the paper is organized as follows: Section 2 gives a brief introduction to the Xen hypervisor and its I/O capabilities. We will discuss related work in Section 3. In Section 4, we will detail the network configurations that were compared and tested. Section 6 will present and analyze experimental results. Section 7 contains the conclusions and future work.

2. Background

Xen is an open source Virtual Machine Monitor (called Hypervisor or VMM) that enables running multiple operating systems on a single machine (6). Xen gives users the options of running either in fully virtualized mode or para-virtualized mode. In full virtualization, the guest operating system is presented with the full abstraction of underlying physical hardware. In para-virtualization, the guest domain is presented with hardware which is similar but not identical to the physical hardware. The hypervisor runs in the most privileged ring and provides necessary mechanisms (e.g. virtualizing resources) for controlling and running the guest operating

3rd Workshop on System-level Virtualization for High Performance Computing (HPCVirt'09) 31 March 2009, Nuremberg, Germany.

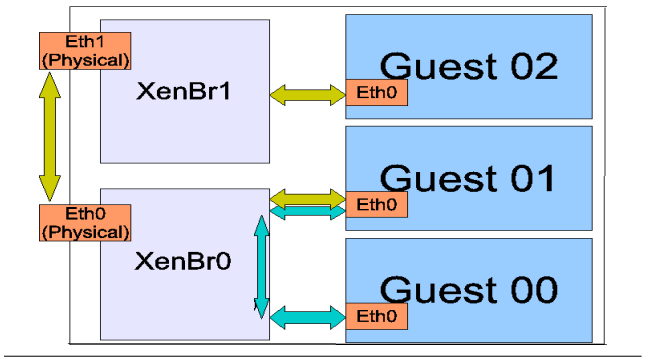

Figure 1. Xen bridge architecture.

systems. Xen also requires a special privileged guest domain called *domain 0* to control the guest domains e.g. start, shutdown and migrate the guest operating systems across the infrastructure.

Xen follows a split device driver model for I/O device virtualization. The native devices are run under the *Isolated Device Domain* (IDD) which is essentially *domain 0*. Domain 0 (Dom0) provides a back-end driver for the guest domains (also called domUs). The guest domains use their front-end drivers to communicate with the back-end drivers provided by Dom0. In the case of network I/O[1], the physical device can be multiplexed, so that it can be used by a number of guest domains. Such network interfaces are called virtual interfaces (vifs) for the guest domains. When a packet arrives from the outside world, it is handled by the ethernet driver of Dom0 and is eventually sent to a software bridge called XenBridge(N), where 'N' is the ethernet interface connected to the bridge[2]. The bridge distributes the packet just like a normal switch to the destination guest domain. If two VMs are co-located and are attached to the same bridge, then a packet destined from one VM to another co-located VM is routed through this software bridge. In the case where co-located VMs are attached to different Xenbridges, the packets are routed through the physical switch. The high-level bridge routing mechanism is shown in Figure 1.

Xen also gives the user a facility to by-pass the driver domain (Dom0) and export the device directly to guest domain. For network interfaces, this method provides better performance compared to virtual interfaces. A major drawback of this method is that the guest domain cannot be migrated from one VMM to another.

3. Related Work

A lot of research work has been done in the performance evaluation of Xen or similar VMMs, Most of the current research focuses on fault tolerance and the viability of using Xen in high performance computing (HPC).

Nagarajan et al. (5) utilized Xen 3.0.2 for fault tolerance in a cluster environment. They evaluated the NAS benchmarks on a compute cluster and concluded that compute clusters can be successfully virtualized with the Xen VMM. The paper found that for class C benchmarks, virtualization was quite competitive on a 16 node dual core cluster. However, each node was only running a single virtualized guest domain, which resulted in the under-utilization of compute resources. As we show in later sections, this does not represent SMP cluster and Xen network performance in the case of intra and inter domain communication.

[1] The back-end driver is called Netback and front-end driver is called Netfront.

[2] From Xen 3.2, XenBridge(N) has been changed to Eth(N). To avoid confusion we will use the older convention.

Strazdins et. al (12) evaluated a number of Gigabit ethernet network configurations for performance enhancement of SMP clusters under Xen virtualization. Channel bonding and VMM bypass were considered, with comparisons being made to equivalent native Linux configurations. The best performance came from configurations using Xen virtualized hosts with VMM-bypass for network I/O. It was also concluded that the intra-node communication between Xen guests on the same node is an order of magnitude slower than the native shared memory transport. This counteracts the advantages of VMM-bypass for configuring an SMP cluster with a Xen guest on each CPU. It was suggested to implement a shared memory transport for network communication for the guests sharing same physical host. However, this conclusion was reached only from the consideration of micro-benchmarks.

Numerous attempts have been made to enhance the network performance of Xen. Among them Xensocket (15), Xway (9), IVC (8) and Xenloop (13) are notable. Xensocket provides a one way communication socket between two guest domains on the same physical host using a grant table mechanism. Xensocket is particularly useful for the applications which are aware of being hosted on the same Xen virtualization platform. The biggest drawback of Xensocket is that its implementation is not binary compatible with other socket implementations and it does not support migration of the operating system. Xenloop and Xway are attempts to provide binary compatibility with native socket implementations. Xenloop uses netfilter to steal packets destined towards the co-located domain and uses grant table operations to deliver the packet. Our testing of Xenloop revealed that Xenloop's performance for OSU and NAS Benchmarks was equal to the default Xen mechanism. Xway is a similar open source attempt, but it requires certain changes to Xen hypervisor itself. To our knowledge, no one has been able to replicate the efforts of XWay team.

Huang et. al. (8) implemented an inter-VM communications library (IVC) to support shared memory communication between guest operating systems on the same physical host by implementing a VM-aware MPI based on MVAPICH2 (called MVAPICH2-ivc). The results show that inter-VM communication mechanism yielded approximately 11% better performance with the NAS benchmarks as compared to native Xen implementation. Some micro-level benchmarks comparing Xen and native Linux environment were also carried out in the paper. However, performance results across cluster nodes were not given, and in any case IVC is specific to MVAPICH2 and hence Infiniband.

4. Network Configurations

In this section we discuss the various network configurations that were tested for determine the viability of utilizing Xen for HPC applications. It is assumed that on each node, there are (at least) one CPU and one physical network interface for each guest domain, and that each guest domain will host a single application process.

Each guest domain is provided with two virtualized GigE interfaces, eth0 and eth1. Eth0 is normally reserved for cluster management tasks, whereas eth1 is used for MPI communication. All the eth0's for the guest domains are virtualized and attached to XenBridge0 (also called Xenbr0). XenBridge0 is connected to the physical eth0 of Dom0. Eth1 is configured in a different manner as discussed below.

The first configuration is called *Exported Interfaces* or *VMM bypass*. In this configuration we export the PCI bus to the guest domain via the PCI-back mechanism, and connect this to the guest's eth1. This configuration enables all the guest domains to have a dedicated, non-virtualized GigE interface for MPI communication. This mechanism is proven to give high performance in latency and bandwidth (12). Currently Xen provides no mechanism to migrate a guest domain which has a bypassed interface.

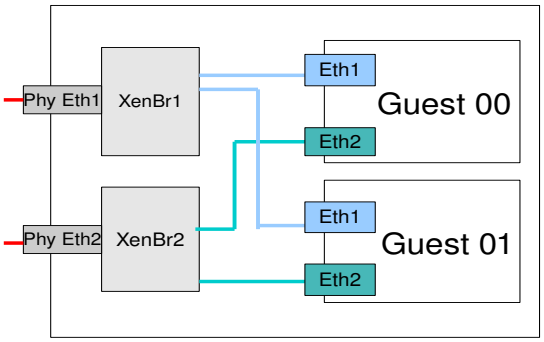

Figure 2. Xen multiple shared bridges configuration.

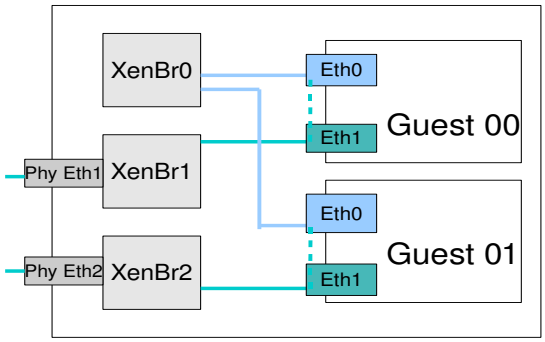

Figure 3. Xen shared-separate bridge configuration.

The second configuration is the *Shared Bridge* configuration. In this configuration, the eth1 interface of the guest domains is connected to XenBridge1, which is in turn connected to the physical eth1 of Dom0. This means that all the guest domains on the same VMM are using only one physical GigE interface, and hence the bandwidth is divided between the guests. The advantage of using this configuration is that intra-domain communication should be relatively fast, since it goes through XenBridge1.

We also use a variant of Shared Bridge called *Multiple Shared Bridges* as shown in Figure 2. In this configuration, the eth1 interface of all the guest domains is attached to XenBridge1 as in Shared Bridge. In addition, the guest domains are also provided with eth2 interface, which is connected to XenBridge2. XenBridge2 is connected to the physical eth2 of dom0. The eth1 and eth2 interfaces are on different subnets and hence in the case of OMPI this creates the effect of Shared bridges for intra-domain communication and utilization of two interfaces for inter-domain communication.

The third configuration is the *Separate Bridge* configuration. This configuration is similar to Exported Interfaces, except the network interfaces are virtualized. This means that the eth1 of each guest domain is connected to a different Xen Bridge, which is connected to a distinct GigE interface on Dom0. This results in each guest domain having a dedicated but virtualized GigE interface for MPI communication. The disadvantage of using this configuration is that intra-domain communication is not as fast as the Shared Bridge configuration because communication is routed through physical switches.

The last network configuration is *Shared-Separate*, which is mix of the Shared Bridge and Separate Bridge configurations. In this configuration we utilize both the interfaces (eth0 and eth1) for MPI communication as shown in Figure 3. Eth0 of each guest VM is connected to XenBr0 and eth1 of each VM is connected to a distinct Xen bridge (Separate bridge). We set the routing tables of each guest VM so that the network packets destined towards co-located VM are routed through eth0, resulting in use of the Shared Bridge (XenBr0), whereas packets destined for outside VMM are routed through eth1. This effectively results in the utilization of a software bridge intra-domain communication and the utilization of distinct virtualized network interfaces (vifs) for inter-domain communication.

To our knowledge the Separate Bridge and the Shared-Separate Bridge configurations have never been presented or tested by the HPC community.

We do not consider configurations involving channel bonding as this has been shown to yield poor performance (12). Our subsequent experiments have shown this to be still the case.

A native Linux configuration will be used as a performance baseline. Here, the same number of processes as guest domains are allocated to Dom0, and the system is configured to use exactly the same number of network interfaces. Communication for intra-node processes is carried out through shared memory. For inter-node communication, the MPI implementation will determine how the GigE interface are used, as is explained below.

5. MPI Implementation Differences

We discuss two popular MPI-2 compliant implementations, namely OpenMPI (OMPI) (3) and MPICH2 (1). These two implementations are fundamentally different from each other, especially in terms of architecture and routing issues. Both of these implementations claim to have highly configurable and efficient TCP and shared memory communication infrastructure.

One noticeable difference is that in OMPI the basic unit of routing is a process, not an IP address (4). In the case of TCP, this routing mechanism enables every OMPI process to use all the network interfaces in a round-robin fashion. During the initialization, OMPI determines the interfaces which are routable and then the MPI messages are striped across the network utilizing all of the routable interfaces. This theoretically can achieve the aggregate bandwidth of all individual network interfaces. This of course is limited by the PCI bus speed and other hardware limitations. This means that for the native Linux configuration, all processes on the same node share all the network interfaces.

Channel bonding in particular is not preferred in the case of OMPI as it will almost always yield poor results (2). This is due to the fact that channel bonding requires packets to be delivered in order, whereas OMPI can effectively utilize all the available interfaces to stripe and send the message. Especially in the case of large messages, OMPI can write part of the message to user-space without worrying about the order of the packets. This enables OMPI to yield better bandwidth compared to channel bonding.

In the case of MPICH, the default routing mechanism only allows for one GigE interface. However, through the use of multiple IP addresses for each node and routing tables, messages coming into different processes on the same node can be routed through separate interfaces (12).

6. Results

In this section, we present the performance of the various network configurations. All the results are for OpenMPI (OMPI) unless specified otherwise.

6.1 Experimental Setup

We primarily use a 2×4 cluster (2 nodes, with 4 CPUs each) for our experimentation. Each node consists of two SMP dual-core 2.2 GHz AMD Opteron processors with a 2-way 64 KB level 1 data cache and an 8-way 512 unified L2 cache, and 4 GB of RAM. The nodes have an IWILL DK8-HTX 815 motherboard, with 800 MB/s HyperTransport links. The motherboard has in-built dual

Intel 82541GI/PI GigE controllers. External NICs can be connected to two slots on the same 64-bit 100 MHz PCI-X bus, and to one slot on a third 32-bit 33 MHz PCI 2.2 bus. For external NICs with dual Ethernet ports, this permits up to 8 Gigabit Ethernet interfaces on this motherboard. The in-built dual Intel GigE controllers are configured as Eth0 and Eth1. The nodes also have a Pro/1000 MT NIC (Intel 82546GB chips) with dual interfaces; these are configured to eth2 and eth3 utilizing PCI-X bus. A Pro/1000 MT NIC with an Intel 82541PI chip is configured to eth4.

We also use an 8×2 cluster, with dual core AMD Opteron processors with similar specification as that of 2×4 machines. The Eth0 and Eth1 are the in-built GigE interfaces. The Eth2 of each machine is connected to 32-bit 33 MHz PCI bus.

The system software is based on the Ubuntu Hardy distribution, with Xen 3.3 compiled from source using GCC 4.2.4. The kernel of Dom0 and domU is Linux 2.6.18.8. We use the OpenMPI 1.2.2 and MPICH2 1.0.7 implementations. For microbenchmarks, we use the OSU benchmarks (11) for latency, bandwidth and bidirectional bandwidth. For application-level benchmarks we use the NAS Parallel Benchmarks 3.2 (10). For Exported Interface configuration we used Xen 3.1.4 as we were not able to bypass the PCI buses to Guest VMs on Xen 3.3.0 due to some issues. However we do not expect any performance difference between the two implementations as we are utilizing a paravirtualized environment.

All instances of domUs have only one VCPU, which is pinned to a distinct CPU. In the case, where number of guest VMs equal the number of physical CPUs, the Dom0s are not pinned to any specific CPU, as we found that pinning VCPUs for Dom0 actually reduced performance.

All tests involving native Linux were run with processor affinity, similarly pinning each process to a particular CPU. The native Linux case utilizes kernel 2.6.24.21, supplied by Ubuntu Hardy distibution.

6.2 Inter-domain communication

By default, the OSU benchmarks determine the communication behavior (sustainable latency, bandwidth and bidirectional bandwidth) for one communicating pair of processes only. We modified the benchmarks to enable them to span multiple pairs. In the case of inter-domain communication, each communicating pair has two processes, each on a different node of a 2-node cluster. We conduct our test on as many pairs as (enabled) network interfaces (1 to 4 pairs).

For a clearer comparison between the network configurations discussed in Section 4, we have summarized the latency results in Tables 1 and 2 [3]. These tables present latency at 1 byte and 4 MB message size respectively. Bandwidth and bidirectional bandwidth results are summarized in Tables 3 and 4. These results show average sustainable bandwidth results for messages sizes between 4 KB and 4 MB.

Figures 4, 5 and 6 give more detailed results for the bandwidth benchmark. The detailed data for the bi-directional bandwidth benchmarks is similar. The Shared-Saperate and Multiple-Shared bridge configurations were not tested as they are not applicable to point-to-point communication benchmarks.

Results for native Linux using OMPI and MPICH firstly indicate a noticeable difference in the performance of the two MPI implementations, with OMPI generally performing slightly better than MPICH.

In the case of 1-pair communication, there is no significant difference between any of the network configurations, native or virtualized. The exception is that the Shared and Separate Bridge configurations are slightly slower for bi-directional bandwidth where the overhead of virtual interfaces begins to be felt.

For the two pair communication, the performance of all the network configurations is still comparable for the latency and bandwidth benchmarks, except that the Shared Bridge configuration falls behind as expected because it is using only one GigE interface. For the Shared and Separate Bridge configurations, the Dom0 kernel remained considerably busy (approximately 30%). As each machine has four CPU cores, Dom0 had two CPU cores at its disposal therefore its performance is competitive.

Table 1. Summary of Latency (μ Sec) at 1 Byte

Config	1 Pair	2 Pairs	3 Pairs	4 Pairs
Linux-OMPI	125	94	114	123
Linux-MPICH	106	104	124	123
Exported Interfaces	125	112	80	110
Separate Bridges	125	129	125	160
Shared Bridge	126	125	128	149

Table 2. Summary of Latency (MB/Sec) at 4 MB

Config	1 Pair	2 Pairs	3 Pairs	4 Pairs
Linux-OMPI	109	199	199	248
Linux-MPICH	109	218	202	240
Exported Interfaces	109	197	326	408
Separate Bridges	109	161	190	194
Shared Bridge	108	124	136	126

Table 3. Avg. Bandwidth (MB/Sec) for message size ≥4K

Config	1 Pair	2 Pairs	3 Pairs	4 Pairs
Linux-OMPI	109	165	300	397
Linux-MPICH	105	186	305	366
Exported Interfaces	105	183	312	394
Separate Bridges	102	182	177	142
Shared Bridge	102	119	105	96

Table 4. Avg. Bi-Bandwidth (MB/Sec) for message size ≥4K

Config	1 Pair	2 Pairs	3 Pairs	4 Pairs
Linux-OMPI	124	296	337	408
Linux-MPICH	124	273	350	361
Exported Interfaces	121	183	370	415
Separate Bridges	115	183	177	168
Shared Bridge	115	119	105	111

For three pair configurations, the performance gap between native Linux and the configurations utilizing the Xen bridge mechanism becomes quite visible. The Separate Bridge is almost 2 times slower than the native Linux. It is however 1.75 times faster than the Shared Bridge. We observed an increased number of cache misses for the Shared Bridge, as compared to the Separate Bridge configuration. Exported Interfaces out-performs native Linux; this is due to the fact that it offers a better parallelization of the processing of the TCP/IP stack, as explained in (12).

For four pair communication, both the bridge configurations perform poorly. The Shared Bridge is approximately 3.5 times slower than native Linux, whereas the Separate Bridge configuration is 2.5 times slower. This is due to all the CPUs being required for the domUs and no dedicated CPU left for Dom0.

From the experiments above, we can conclude that latency and bandwidth are affected by a factor of two or more if the Xen bridge mechanism is utilized. It is clear that using Separate Bridge mechanism for vifs is better as it gives at least 50% improvement over the conventional Shared Bridge mechanism. The reduced bandwidth in

[3] As OSU Latency benchmark is essentially a ping-pong benchmark; therefore the results of latency at 4 MB are shown in MB/Sec

4-pair case compared to 3-pair shows that Xen's netback-netfront implementation is highly CPU intensive and Xen will always benefit from having at least one CPU spare for inter and intra-domain communications.

However the OSU benchmarks only give half the picture. For a mix of scientific applications we decided to run NAS benchmarks over the 2×4 and 8×2 compute clusters, as discussed in Sections 6.3 and 6.4.

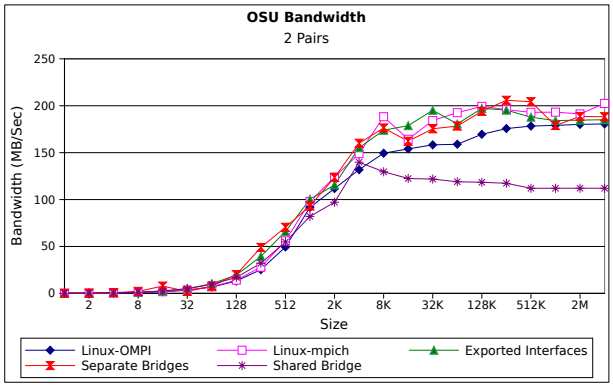

Figure 4. OSU 2 pair bandwidth benchmarks

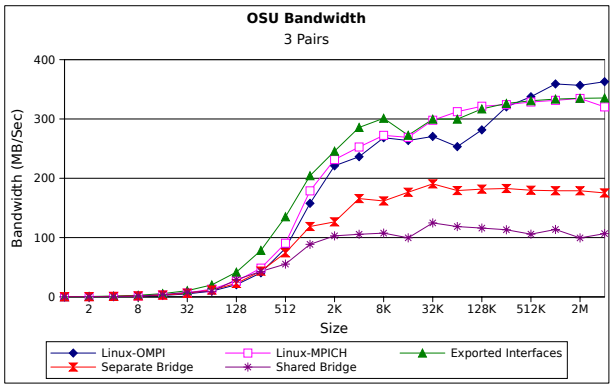

Figure 5. OSU 3 pair bandwidth benchmarks

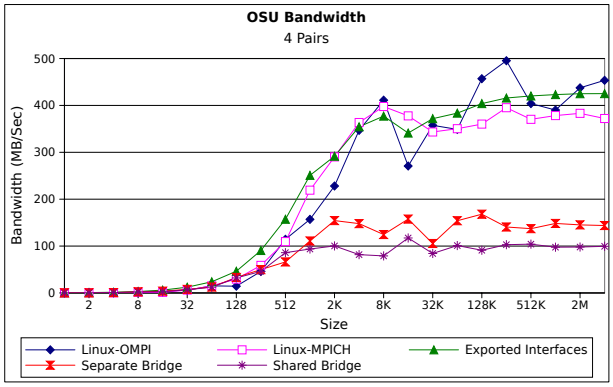

Figure 6. OSU 4 pair bandwidth benchmarks

6.3 Intra-domain Communication

For these experiments, we have two or more guests located on the same VMM and study the performance of communication between processes on these guests. We thus used one node of the 2 × 4 cluster discussed in Section 6.1, and are thus limited to 4 guests, as we do not want to over-subscribe the CPUs.

In the case of the OSU benchmarks, we can have up to two communicating pairs. We tested Shared Bridge, Separate Bridge and native Linux configurations using OMPI. The results of the OSU Bandwidth results are shown in Figure 7 and 8. The summary is provided in Tables 5.

Table 5. Avg. bandwidth (MB/Sec) for message size ≥4K

	Bandwidth		Bi-Bandwidth	
Config	**1 Pair**	**2 Pairs**	**1 Pair**	**2 Pairs**
Linux-OMPI	1250	1804	1163	1456
Exported Interfaces	87	196	82	127
Separate Bridges	84	94	85	98
Shared Bridge	344	300	366	338

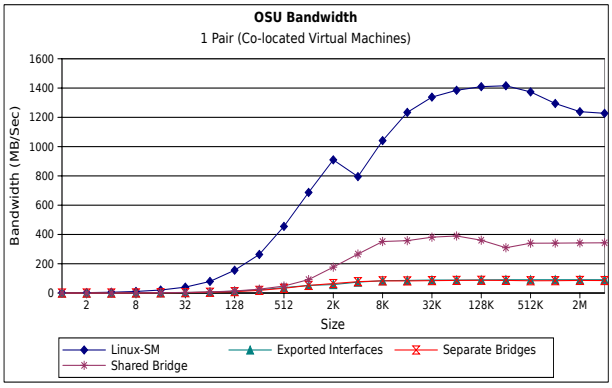

Figure 7. OSU bandwidth (1-pair, co-located VMs)

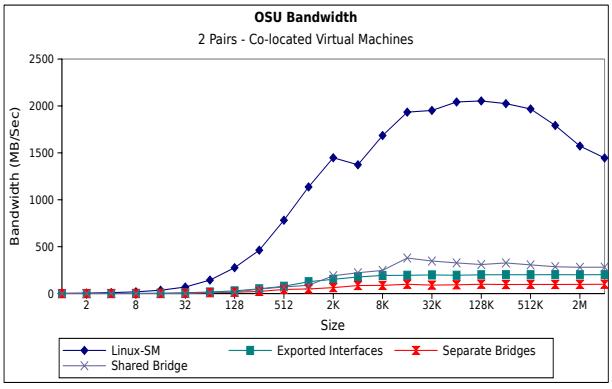

Figure 8. OSU bandwidth (2-pair, co-located VMs)

For 1-pair bandwidth and bidirectional bandwidth benchmarks, Linux shared memory outperforms the Shared Bridge by 50%. The Separate Bridge configuration is an order of a magnitude slower, as routing is done by the physical switch. In the 2-pair configuration we see this gap increasing compared to the 1-pair configuration. Exported Interfaces and Separate Bridges actually see improvement due to the use of two physical GigE interfaces but the Shared Bridge configuration sees a decline, asserting the fact that Xen's page flipping mechanism is highly CPU intensive and does not scale well.

To evaluate scientific applications, we evaluated the system with the NAS parallel benchmarks. Normalized results with respect

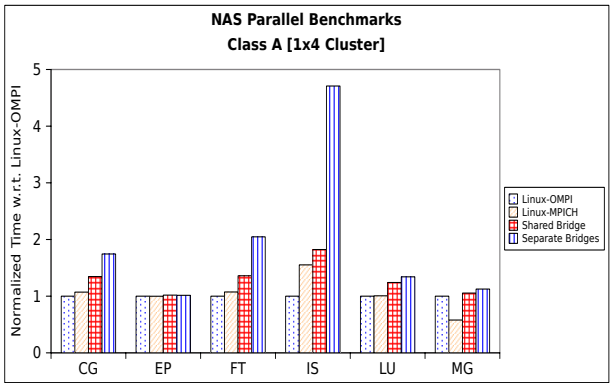

Figure 9. NAS parallel benchmarks on a 1×4 cluster.

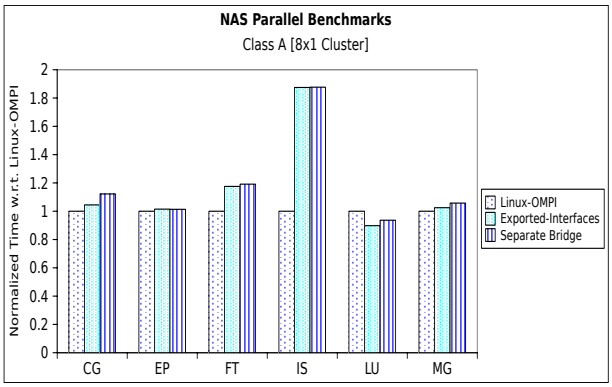

Figure 11. NAS parallel benchmarks, Class A on a 8×1 cluster configuration

to native Linux with OMPI are shown in Figure 9. The results of the NAS parallel benchmarks (NPB) assert the fact that, for communication intensive applications, Xen's software bridge fails to match the native shared memory transport. For applications which require less communication bandwidth like EP, LU and MG, the performance of Xen bridge is quite competitive. In the case of IS, the performance of Xen bridges is quite bad, which is due to the high communication rate in the benchmark.

We also tested the Exported Interface configuration. Its performance was generally between that of the native and Xen bridge configurations.

6.4 Inter-Intra Node Communication Mix

We employed the NAS parallel benchmarks (NPB) to evaluate impact of above mentioned network configurations on applications spanning multiple Xen hosts.

A 2×4 cluster will expose the impact of the configurations for the situation where the amount of inter- vs intra- node communication is roughly balanced. With one process (one guest) allocated per CPU, it also presents the situation where no CPU can be dedicated to Dom0.

In order to evaluate the impact on larger clusters where inter-node communication is more dominant, we also used the 8-node cluster. We similarly tested one process (one guest) allocated per CPU ('8×2 cluster'). However, to create the situation where a CPU is dedicated to Dom0 to support communication, we also tested one process (guest) per node ('8×1 cluster'). A comparison between the two would thus expose the CPU overheads involved in Xen's network communications.

As OMPI and MPICH employ different mechanisms for routing of multiple interfaces, we tested all the network configurations with OMPI and MPICH. The results of MPICH with Shared Bridge, Separate Bridge and Exported Interfaces were almost identical to OMPI and therefore are not represented. We were not able to run Shared-Separate-Bridge-OMPI because of the fact that OMPI determines the routable interfaces through its own mechanism and does not rely on the routing table information(4). The results of NPB, Class A on the 2×4 cluster configuration are shown in Figure 10.

We see that MPICH and OMPI behave in a slightly different fashion. In case of native Linux, OMPI generally has an advantage over MPICH, as we saw in the micro-benchmarks. However, we noticed that OMPI is unable to determine the fastest interface and utilizes all the routable interfaces. This in particular causes degradation in the case where one interface is faster than the other.

Exported Interfaces gives excellent performance (only 5% slower at an average compared to native Linux), despite the fact that the routing of packets is done through a physical switch. This indicates that the advantage of native Linux shared memory transport for intra-node communication becomes relatively small once inter-node communication is introduced.

The Shared Bridge configuration gives reasonable performance. However,due to the attached CPU overheads, this configuration results in a performance degradation of less than 40%, as compared to the native Linux.

The slow performance of Separate Bridges (OMPI and MPICH) is a surprise. This could be due to two factors. Firstly, the Xen netfront-netback infrastructure is CPU intensive not only for intra-domain communication but also for the communication with the outside world. Secondly, the inter-VM communication of co-located domains with Separate Bridges is routed through the physical switch; however, this factor does not seem to have hurt Exported Interfaces.

The Shared-Separate Bridge configuration gives better results compared to other two bridge configurations and is at an average only 8-10% slower than the native Linux configurations on all benchmarks except the IS benchmark. The reason for better performance is the fact that this configuration utilizes Shared Bridge for the communication with co-located domains and Separate Bridge for communication with domains on other VMMs, hence the VMs end up using two network interfaces instead of only one.

The 2×4 configuration results assert that the netback-netfront implementation of Xen is CPU intensive. As no CPU is available to Xen to handle packet delivery therefore this results in Xen stealing the CPU times of the guest domains, resulting in loss of precious CPU cycles as well as high number of cache misses.

To test the above assertion, an 8×1 cluster configuration was used to see if the availability of a CPU for inter-domain communication, coupled with the lack of intra-domain communication, would result in improved performance of the virtualized configurations. The results are shown in Figure 11. Note that here, there is no distinction between Shared and Separate interface, and the results of MPICH are not represented as they were similar.

We noticed that the benchmarks run was overall 30-35% faster than the 2×4 configuration run. This is primarily due to having single process per node, which reduces the effects of memory and hypertransport bandwidth limitations. With the exception of IS, the virtualized configurations were much closer to native Linux than on the 2 × 4 cluster, confirming the benefit of having a CPU available to support Xen communication.

The results for the 8×2 cluster configuration are shown in Figure 12. As expected, we see that Exported Interface configuration performs almost equal and at times better than native Linux. This suggests that if live migration of virtual machines is not required

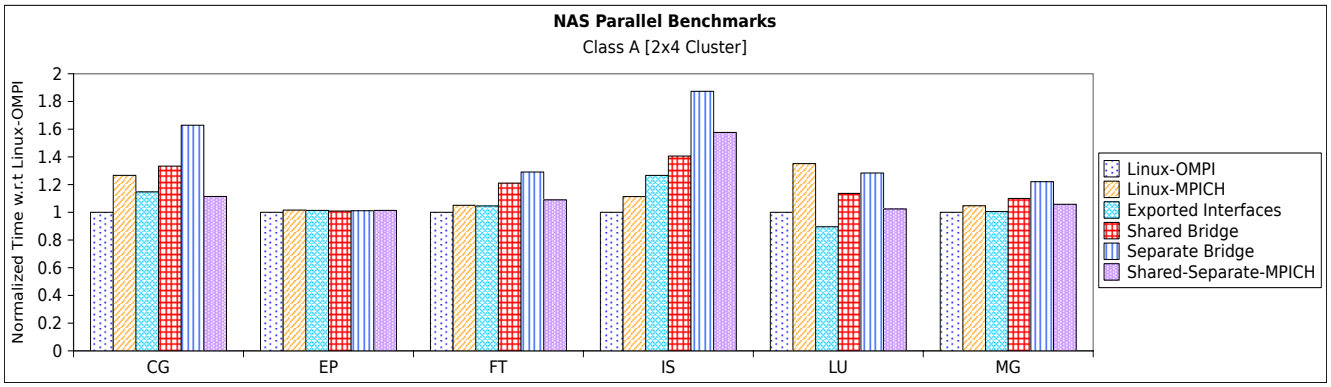

Figure 10. NAS parallel benchmarks on a 2×4 cluster

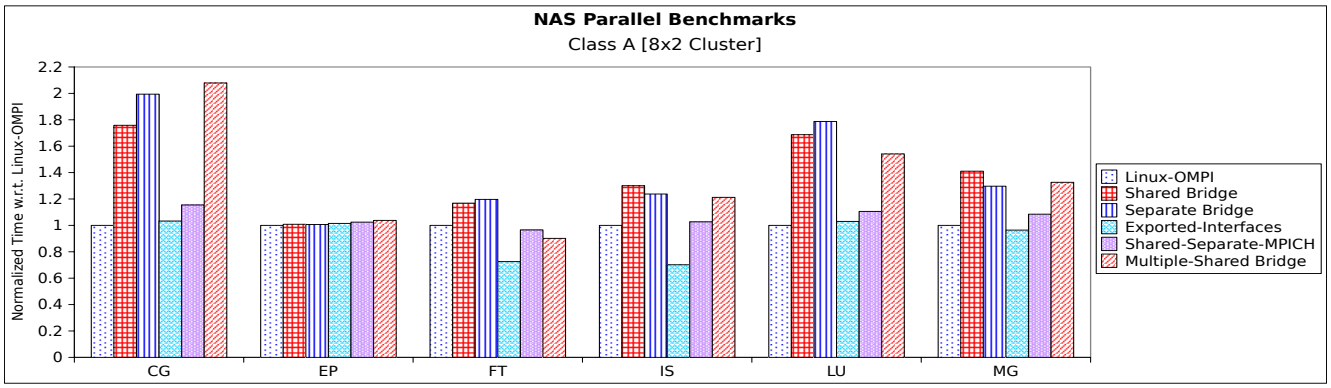

Figure 12. NAS parallel benchmarks, Class A on a 8×2 cluster

in a compute cluster, then virtualization can be beneficial. The results for Shared Bridge and Separate Bridge configuration are similar to each other, being 15–30% slower than native Linux. The Shared-Separate bridge configuration gives excellent performance considering it utilizes virtual interfaces. This is due to its effective use of the second interface. As discussed Shared-Separate configuration cannot be used for OMPI, therefore we used an equivalent Multiple-Shared bridge configuration. This configuration performs slightly better than the Shared bridge and Separate bridge configurations, as it is utilizing two interfaces, but it fails to match performance of Shared-Separate bridge configuration. This is due to the fact that eth1 and eth2 of the guest VMs are not completely isolated when sending the inter-domain messages. This results in bandwidth sharing among the two co-located VMs and therefore as seen previously, we see high cache misses.

7. Conclusions

From the inter and intra-domain communication results, it can be concluded that in the case where a job spans a number of physical machines, exporting network interfaces to guest machines achieves near native performance. Therefore if job migration is not an essential requirement, the HPC community can benefit from the virtualization technology without any significant cost. This is even in the absence of any fast intra-domain communication mechanism.

If exported interfaces cannot be utilized, then Xen virtualization can be slow compared to non-virtualized environment unless there are spare CPUs available for Dom0. We see that inter-domain communication in Xen via Xen bridges is quite CPU intensive. This weakness is only exposed if one is utilizing multiple GigE interfaces on a SMP Xen host. This is in contrast to findings in other previous work e.g. (5), where no significant degradation in Xen's network performance over native Linux was seen. We conclude that Xen bridge infrastructure in general is CPU intensive and depending upon communication requirements and patterns, can give 20–40% degraded application performance as compared to native Linux.

However, we have found that, compared to the default configuration (Shared Bridge), using Separate Bridges can improve micro-benchmark performance, and that using a combination of the two can improve application performance to almost as good as native Linux.

Another observation is that the current bridge architecture of Xen is not aware of the fact that certain VM's might be utilizing another XenBridge for communication. In principle, all the guest domains should use shared memory transport if they are hosted on same VMM; therefore the bridge architecture should be made co-VM-aware. Although this arrangement will not result in native memory transfer rates, it will be at least much faster than utilizing the physical switch in the case of Separate Bridge or Exported Interface configurations.

Mechanisms like IVC (8) can be utilized for faster intra-domain communication. However our experiments show that their impact on applications spanning multiple VMMs will not be very high. These solutions are not generic in nature as they are MPI implementation centric and Xen version specific. As Xen is continuously evolving, these solutions quickly become obsolete or there is a requirement to port them to newer versions. One such example is Xenloop, which can only work with Xen 3.1 but fails on Xen 3.3. The best way to ensure shared memory or equivalent intra-domain

communication is by overhauling the netfront-netback implementation of Xen hypervisor in the official Xen trunk. This will result in a generic solution, not requiring ports to various MPI implementations and domain drivers for every new version.

We have seen that in the presence of additional CPU cores for VMM, Xen's network performance is competitive to native Linux for MPI application spanning multiple physical nodes. With the evolution of heterogeneous core platforms, a simple but CPU intensive operations like memory copying and page flipping can be offloaded to simple integer or similar low-end core. This will greatly increase VMM's performance in general. Under this scenario, the need for fast intra-domain communication mechanisms would be questionable.

We have also noted that MPI implementation is an important factor in this context; in particular, routing mechanism can affect the way multiple network interfaces may be used. It also affects performance, with OMPI's architecture better being able to use multiple interfaces, as described in Section 5.

Future work includes extending these evaluations to other platforms, in particularly to highly multi-core clusters with more powerful communication interfaces. It also includes further development and evaluation of fast inter-domain communication mechanisms. In general, the outlook is positive that HPC can benefit from the advantages of virtualization, at the cost of small or even negligible performance loss.

8. Acknowledgements

We acknowledge Alexander Technology for donating us the hardware and the friendly OpenMPI community for prompt and detailed replies to our queries.

References

[1] Mpich2. http://www.mcs.anl.gov/research/projects/mpich2/, 2008.

[2] Openmpi developers mailing list. http://www.open-mpi.org/community/lists/users/2006/03/0837.php, 03 2006.

[3] Openmpi. http://www.open-mpi.org/, 06 2008.

[4] Openmpi faq: Tuning the run-time characterisitics of mpi tcp communications, 01 2009.

[5] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive fault tolerance for hpc with xen virtualization. In ACM, editor, *In Proceedings of the 21st Annual International Conference on Supercomputing (ICS07)*, 2007.

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

[7] Niels. Fallenbeck, Hans-Joachim. Picht, Matthew. Smith, and Bernd. Freisleben. Xen and the art of cluster scheduling.

[8] Huang.Wei, Koop. Matthew, Gao. Qi, and Panda. K. Dhabaleswar. Virtual machine aware communication libraries for high performance computing. In *Super Computing 07, Reno, Nevada, USA*, Nov 10-16 2007.

[9] Kangho Kim, Cheiyol Kim, Sung-In Jung, Hyun-Sup 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 (VEE'08)*, pages 11–20. ACM, March 2008.

[10] NASA Advanced Supercomputing. NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.

[11] Nowlabs, Ohio State University. OSU Benchmarks. http://mvapich.cse.ohio-state.edu/benchmarks/.

[12] Peter Strazdins, Richard Alexander, and David Barr. Performance enhancement of smp clusters with multiple network interfaces using virtualization. *Springer-Verlag*, LNCS 4331:452–463, 2006.

[13] Ian Wang, Kwame-Lante Wright, and Gopalan Kartik. Xenloop : A transparent high performance inter-vm network loopback. In *International Symposium on High Performance Distributed Computing (HPDC)*, June 2008.

[14] Lamia Youseff, Rich Wolski, Brent Groda, and Chandra Krintz. Paravirtualization for hpc systems. In *Lecture Notes in Computer Science*, pages 474–486. Springer-Verlag, 2006.

[15] Xiaolan Zhang, Suzanne McIntosh, Pankaj Rohatgi, and John Linwood Griffin. Xensocket: A high-throughput interdomain transport for virtual machines. In *Middleware 2007*, volume 4834/2007, pages 184–203. Springer Berlin / Heidelberg, 2007.