Exploring I/O Virtualization Data paths for MPI Applications in a Cluster of VMs: A Networking Perspective

Anastassios Nanos, Georgios Goumas, and Nectarios Koziris

Computing Systems Laboratory,
National Technical University of Athens,
{ananos,goumas,nkoziris}@cslab.ece.ntua.gr

Abstract. Nowadays, seeking optimized data paths that can increase I/O throughput in Virtualized environments is an intriguing task, especially in a high-performance computing context. This study endeavors to address this issue by evaluating methods for optimized network device access using scientific applications and micro-benchmarks.

We examine the network performance bottlenecks that appear in a Cluster of Xen VMs using both generic and intelligent network adapters. We study the network behavior of MPI applications. Our goal is to: (a) explore the implications of alternative data paths between applications and network hardware and (b) specify optimized solutions for scientific applications that put pressure on network devices. To monitor the network load and the applications' total throughput we build a custom testbed using different network configurations. We use the Xen bridge mechanism and I/O Virtualization techniques and examine the trade-offs. Specifically, using both generic and intelligent 10GbE network adapters we experiment with assigning network Virtual-Physical Functions to VMs and evaluate the performance of a real scientific application using several networking configurations (multiplexing in hypervisor-level vs. firmware-level via IOV techniques). Preliminary results show that a combination of these techniques is essential to overcome network virtualization overheads and achieve near-native performance.

1 Introduction

Today, with the advent of virtualization techniques, Cloud Computing infrastructures are becoming a great trend, providing flexibility, dedicated execution and isolation to a vast number of services. These infrastructures, built on clusters of multicores, offer huge processing power, ideal for mass deployment of compute-intensive applications. However, bridging the gap between I/O techniques in virtualized environments and application demands seems to be a major challenge. Numerous studies both in native [1,2] and virtualized environments [3,4,5] explore the implications of alternative data paths that increase the system's I/O throughput and help applications overcome significant bottlenecks in data retrieval from storage or network devices.

Typical HPC applications often utilize adaptive layers to overcome limitations that operating systems impose in order to ensure security, isolation and fairness in resource allocation and usage. These layers are usually communication libraries (e.g. MPI) or mechanisms to bypass the general purpose kernel-algorithms for (i) process scheduling (CPU affinity, process priority) and (ii) device access (user-level networking, direct I/O techniques such as zero-copy, page-cache bypass, etc.). Intelligent interconnects, suitable for HPC applications, provide adapters that offload protocol processing and achieve fast message exchange. These adapters feature specialized hardware such as DMA engines, volatile memory, I/O or network processors and an interface to the physical medium. To avoid the overhead associated with user-to-kernel–space communication, HPC interconnects often utilize a user-level networking approach. To use such a method in virtualized environments, several issues have to taken into account.

Data retrieval from storage or network devices in virtualized environments is usually realized by software layers within the hypervisor, which allow VMs to interface with the hardware. A common implementation of such interfaces is a *split driver model*. These layers host a *backend* driver that communicates with the native driver and the device, while guest VM kernels host a *frontend* driver, exposing a generic device API to guest user– or kernel–space. The backend, to exchange control information and data with the frontend, exports a communication mechanism, implemented using interrupt routing, page flipping and shared memory techniques. This mechanism multiplies the already numerous data paths and complicates the way data flow from applications to the network.

Similarly to operating systems, the hypervisor in virtualized environments multiplexes guest kernels which run on VMs and are not directly aware of the underlying hardware. For example, for an application running in a VM to utilize a user-level communication mechanism, both the VM kernel and the hypervisor (or the privileged guest) have to be notified. Moreover, the application has to access specific resources on the network adapter's hardware. However, letting applications access I/O devices without regulation raises security issues.

Currently, only a subset of the aforementioned adaptive layers is implemented in virtualization platforms. For example, SR/MR-IOV [4] lets VMs exchange data with the network via a direct data path, bypassing the hypervisor and the privileged guest. This is realized using intelligent adapters that directly export a part of their capabilities to VMs. Device access by multiple VMs is then multiplexed in firmware running on the hardware itself. Thus, VMs can communicate with the network without the intervention of the Virtual Machine Monitor (VMM) on the critical path. However, these features are only implemented for general purpose networking adapters (such as Ethernet) and as a result, cannot be used with High-performance interconnects such as Myrinet or InfiniBand.

Our work is focused on integrating HPC interconnect semantics into the VMM split driver model [5]. We aim to decouple data transfers from the virtualization layers and explore direct application-to-NIC data paths. Nonetheless, the implications of this mechanism on the overall throughput constitute a possible caveat of our approach: the way the control path interferes with data communication may result in significant overhead. Thus, in order to justify developing a framework to support standard features of HPC interconnects (user–level networking, zero–copy etc.) in VM environments, we need to examine the behavior of HPC applications in such environments [6]. In this work, we deploy network benchmarks and a real scientific application in a cluster of ParaVirtualized Xen [7] VMs and present some preliminary results.

The rest of this paper is organized as follows: Section 2 presents network performance measurements using common micro-benchmarks. In Section 3 we describe the evaluation of an advective equation application when deployed in a cluster of VMs over smart 10GbE interfaces. In Section 4 we discuss evaluation issues and related work. Section 5 concludes.

2 Network Performance in Xen VMs

In this section, we evaluate various network configurations using two popular network micro-benchmarks. Our testbed consists of two host machines, connected back-to-back. The host machines (host_0, host_1) are two dual quad-core Xeons@2.0GHz with two Neterion X3110 10GbE adapters, hosting 8 dual-core VMs ($\text{node}_1 \ldots \text{node}_8$) with 1.5GB of memory each. To determine the optimum data path of our testbed, we consider three configurations: `NATIVE`, the baseline of our testbed, running vanilla linux-kernel; `BRIDGED`, the default Xen setup, where all network traffic crosses the privileged guest (Dom0) either by copying or by *granting* the pages that hold the frames to the specified guest; `I/O Virtualization (IOV)`, our optimized setup. Specialized network adapters export PCI functions to the OS providing a direct VM-to-NIC data path.

We measure the bandwidth achieved by each VM separately on different hosts ($\text{node}_1 \rightarrow \text{node}_4$, $\text{node}_2 \rightarrow \text{node}_6$ and so on) and compare its sum to the aggregate bandwidth measured in the Native case ($\text{Host}_0 \rightarrow \text{Host}_1$).

Table 1. Bandwidth (MiB/sec) achieved using netperf TCP_STREAM test

	node1	node2	node3	node4	total
NATIVE					811.73
BRIDGED	90.45	123.03	112.23	100.26	425.97
IOV	160.33	159.43	152.45	162.63	634.84

`netperf`: we used netperf [8] to test the maximum achievable bandwidth that our testbed can sustain. Table 1 shows the bandwidth in MiB per second. The bandwidth achieved in the BRIDGED case is about 65% of the IOV case. On the other hand, IOV sustains 80% of the bandwidth achieved with the NATIVE case, but remains bound at only 50% of the theoretical maximum of the 10GbE link (1250MiB/sec).

Table 2. Bandwidth (MiB/sec) achieved using iperf TCP test (1 process)

	node1	node2	node3	node4	total
NATIVE					1238
BRIDGED	205.00	190.00	181.25	172.50	748.75
IOV	221.25	222.25	221.25	220.00	884.75

`iperf`: to obtain further insight on the apparent degraded performance, we used the iperf benchmark [9] (over TCP) and performed the tests on all the previous configurations. Table 2 shows that the IOV configuration appears to be close to the NATIVE case, which sets the theoretical maximum and outperforms the BRIDGED case by 140MB/sec.

3 Deploying an MPI application in a cluster of VMs

In order to project the results obtained by network benchmarks to a real scientific paradigm, we deploy an HPC application on top of our mini VM-cluster. Our application computes an advective process in a XxYxZ space for a time window T [10]. We choose a fixed grid size ($512x512x512$,

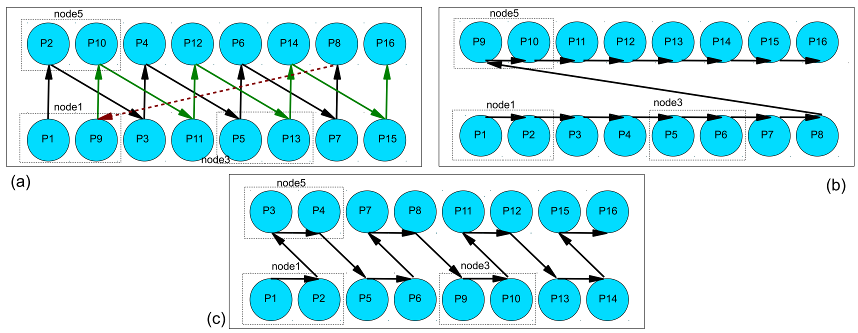


Fig. 1. Communication pattern according to process placement when using all 8 VMs

$T = 512$), distributing X, Y or Z dimension across all processes (16 total processes).

Our physical nodes (Host_0 and Host_1) provide 4 dual-core VMs each, resulting in an 8-node, 16-core cluster (node_1 to node_8). Each process communicates with its nearest neighbor, providing a linear communication pattern. We place processes across cores using three different placement patterns (Figure 1): *a. inter–node*, *b. intra–node*, *c. hybrid*. At first, we choose to place the processes ($P_1 \ldots P_{16}$) in a way that data cross the network in every MPI operation (inter-node). For example, P_2 communicates with P_1 and P_3: we place P_2 on node_5 in Host_1 and $P_{1,3}$ on $\text{node}_{1,2}$ in Host_0 respectively. In order to study how the process placement influences the application's behavior, we then choose the intra-node communication pattern: we place $P_1 \ldots P_8$ on $\text{node}_1 \ldots \text{node}_4$ in Host_0 and $P_9 \ldots P_{16}$ on $\text{node}_5 \ldots \text{node}_8$ in Host_1. Thus, network communication occurs only between node_4 and node_5 (intra–node).

Figure 2 presents the execution time of the advective equation application when using the inter–node and the intra–node cases. In the first bar we plot the application's performance on a native linux kernel setup. In the second and third bar we plot the Xen case, with the BRIDGED and IOV configurations respectively.

This figure raises some interesting issues: (i) in the IOV case, the application execution time is almost half the time of the BRIDGED case for the inter-node communication pattern and its performance achieves 63% of the NATIVE case; (ii) there is significant performance degradation in the case of IOV in intra-node communication; this is expected due to the fact that I/O Virtualization techniques optimize network access

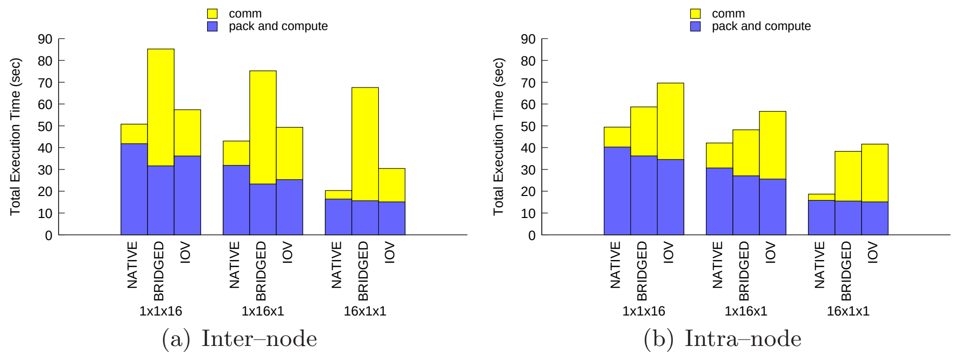


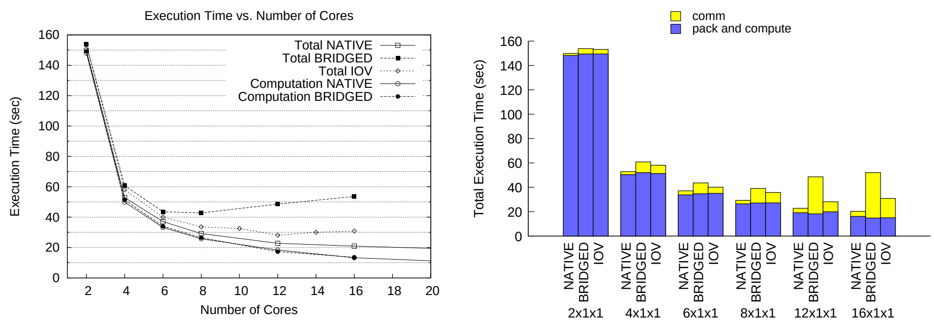
Fig. 2. Advective equation execution time for the linear case $(1x1x16, 1x16x1, 16x1x1)$

and, as a result, data flow directly from the VM to the adapter. In this case, the optimized configuration seems to be the BRIDGED case. An alternative, would be to provide a shared memory mechanism across VMs, as presented in [11,12]; (iii) the speed-up obtained using IOV techniques (Figure 2(a)) compared to the BRIDGED case is not proportional to the bandwidth measured with micro-benchmarks.

To gain further insight on the scalability of the advective application when adding cores, we deployed the application using $2 \ldots 16$ cores. To provide a baseline we deployed the application in a 4-node cluster of machines identical to $\mathrm{Host}_{0,1}$ (32 total processes) using the inter-node placement pattern. Figure 3(a) presents the computation time and total execution time vs. the number of cores for the NATIVE and the BRIDGED case.

In general, it is important to note that the computation time is almost the same for all cases. This is due to the fact that we use Xen in ParaVirtualized mode (PV) and, thus, the application is executed directly on the physical cores (in this mode, the virtualization overhead is negligible, $\approx$ 1 or 2%). Moreover, we observe that in the NATIVE case the communication part of the execution time becomes noticeable over 16 cores. This performance degradation appears in the Virtualized environment as well, and can be attributed to application characteristics. This issue must be addressed in the NATIVE case. Since we are interested in the virtualization overheads on the communication part of the execution, we can study its behavior using 16 cores without loss of generality.

In the BRIDGED case, the application's performance starts to degrade when we add more than 8 cores. Since computation time remains the same in both cases, this degradation is due to the communication



(a) (b)

Fig. 3. Total Execution Time Breakdown for NATIVE, BRIDGED and IOV

overhead associated with the Xen bridge mechanism. We also plot the total execution time of the IOV case (the computation time appears to be the same as in the NATIVE case). We observe a significant performance improvement with IOV due to optimizations in the network layers. Direct data paths allow messages to traverse the network, bypassing the hypervisor or the privileged guest. IOV's performance is nearly 80% of the NATIVE case.

Figure 3(b) presents the execution time breakdown for the $\{X\text{x}Y\text{x}Z\} = \{2 \ldots 16\text{x}1\text{x}1\}$ process distribution using the inter–node communication pattern. In the BRIDGED case (2^{nd} bar), the negative scaling factor is obvious as we add cores to the application. This negative factor is due to the communication part of the execution (light part); the computation part (dark part) remains constant. On the other hand, the IOV case follows the scaling pattern of the NATIVE case, with a constant overhead due to virtualized communication layers.

Based on Figure 1, we can also examine the application's behavior when customizing the number of communication (inter– or intra–node) messages needed for execution. The total number of MPI operations per iteration between 16 processes is 15. Thus, according to the placement pattern (Figure 1): in case a, all MPI operations traverse the network, so the inter–node communication mechanism is the only means of data exchange ($15/15 = 100\%$); in case b, only one MPI operation crosses the network, so the intra–node communication is dominant ($1/15 \approx 6\%$); in case c, there are 7 inter-node messages, leading in a hybrid model, which is the usual communication pattern in a native cluster of SMPs ($7/15 \approx 46\%$).

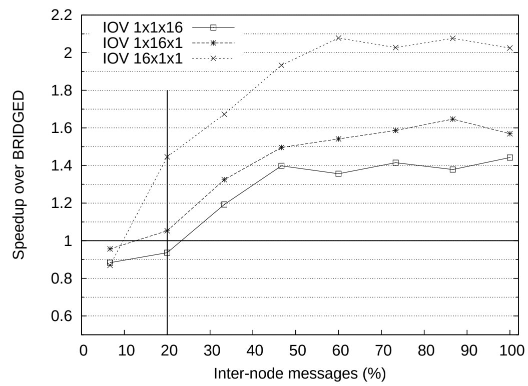


Fig. 4. IOV Speedup over BRIDGED

We plot the speedup of the IOV case over the BRIDGED case vs. the percentage of inter–node messages when distributing dimension X, Y or Z across all 16 processes in Figure 4. We observe that when 50% of MPI operations traverse the network, IOV outperforms the BRIDGED case by at least 40%. The only case where one should choose the BRIDGED case, is when network operations are lower than 20% of all MPI operations (for example Figure 1 case (b)).

4 Discussion and Related work

In virtualized environments, the basic building blocks of the system (i.e. CPUs, memory and I/O devices) are multiplexed by the hypervisor in a secure, isolated and flexible manner. However, as HPC applications often utilize adaptive layers to bypass general-purpose OS constraints, the hypervisor should provide alternative mechanisms of data exchange to VMs and applications. Towards this approach, we deploy a real HPC application in a cluster of VMs in order to examine its behavior. Different network configurations raise some interesting issues:

Xen Networking : Using I/O Virtualization techniques, our application outperforms the generic case. Nonetheless: (i) IOV requires specialized hardware, specific software support and its capabilities are often bound by hardware constraints; for instance, the number of Virtual network interfaces available in our setup, depends on the number of the adapter's Rx/Tx queues; (ii) SR/MR-IOV is currently implemented for ethernet adapters, enforcing all communication libraries to stack their protocols above TCP/IP and Ethernet. HPC applications can benefit greatly from an SR/MR-IOV implementation over custom interconnects (Myrinet, InfiniBand), or over generic Ethernet, leaving all complicated protocol stacks such as TCP/IP to only handle generic network traffic.

HPC applications in clusters of VMs : As shown in Section 3, the computation part of the application's execution time in Xen is the same compared to the NATIVE case either in the BRIDGED or in the IOV mode; the overhead associated with the virtual environment is solely due to the communication part of the execution. Thus, by utilizing a direct optimized data path, the application achieves nearly 88% of the NATIVE case when all MPI operations traverse the network and 70% of the NATIVE case when only one process communicates over the network (Figure 1 for the communication pattern and Figure 2 for the total execution time, cases (a) and (b) respectively).

Several research papers [6,7], have analyzed Xen's performance. In [6] the authors investigate the overheads imposed by the Xen hypervisor using various linux kernel versions (including one custom-built) when running HPC benchmarks in VMs. They conclude that the perceived significant overheads in HPC application execution due to virtualizations layers are unwarranted. Based on this fact, we focus on the evaluation of Xen from a networking and message exchange point of view.

In [11], the authors present a shared memory communication library used for intra-node communication using the KVM hypervisor and achieve near native performance in terms of MPI bandwidth. Huang et al. [12] design an inter-VM, intra-node communication library, implement it on top of a popular MPI library and evaluate its performance. They show that a VM-aware MPI library, in conjunction with VMM-bypass data paths [3] imposes very little overhead to the execution of HPC applications in VM environments. In this work, we evaluate the need to develop a virtualization interconnection framework suitable for HPC applications that can integrate high-performance interconnect semantics into popular hypervisors such as Xen.

5 Conclusions and Future Work

We have presented preliminary performance evaluation results of a real scientific application running in a cluster of Xen VMs. Our work demonstrates the need for profiling application behavior prior to deploying HPC applications in virtualized environments. We explore alternative data paths for network communication between HPC applications that run on clusters of VMs. Specifically, we have shown that for a given parallel HPC application, its communication pattern has to be examined before placing processes in VMs; for instance, our application in a generic Xeon-based cluster using intelligent 10GbE adapters, behaves better when most of the inter-VM communication occurs over the network. We should also note that the computation part of the application execution is not altered when migrating to a VM environment. These results show that HPC applications *can* be executed in VM environments with very little overhead, provided that their communication pattern is examined and that all parallel processes are distributed in a way that data flow through the optimum ad-hoc data path (direct or indirect). We plan on evaluating message passing using shared memory techniques when processes co-exist in VM containers. Our agenda also consists of evaluating higher level frameworks for application parallelism based on MapReduce and its extensions in VM execution environments.

References

1. Geoffray, P.: OPIOM: Off-Processor I/O with Myrinet. Future Gener. Comput. Syst. **18**(4) (2002) 491–499
2. Koukis, E., Nanos, A., Koziris, N.: GMBlock: Optimizing data movement in a block-level storage sharing system over myrinet. Cluster Computing
3. Liu, J., Huang, W., Abali, B., K. Panda, D.: High performance VMM-bypass I/O in virtual machines. In: ATEC '06: Proc. of USENIX '06 Annual Technical Conference, Berkeley, CA, USA, USENIX Association (2006)
4. PCI SIG: SR-IOV (2007) http://www.pcisig.com/specifications/iov/single_root/.
5. Nanos, A., Koziris, N.: MyriXen: Message Passing in Xen Virtual Machines over Myrinet and Ethernet. In: 4th Workshop on Virtualization in High-Performance Cloud Computing, The Netherlands (2009)
6. Youseff, L., Wolski, R., Gorda, B., Krintz, C.: Evaluating the Performance Impact of Xen on MPI and Process Execution For HPC Systems. In: 1st Intern. Workshop on Virtualization Technology in Distributed Computing. VTDC 2006.
7. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I.A., Warfield, A.: Xen and the Art of Virtualization. In: SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, New York, NY, USA, ACM (2003) 164–177
8. Jones, R.: Netperf http://www.netperf.org.
9. Tirumala, A., Gates, M., Qin, F., Duganand, J., Ferguson, J.: Iperf - the TCP/UDP bandwidth measurement tool http://iperf.sourceforge.net/.
10. Goumas, G., Drosinos, N., Koziris, N.: Communication-Aware Supernode Shape. IEEE Transactions on Parallel and Distributed Systems **20** (2009) 498–511
11. Diakhate, F., Perache, M., Namyst, R., Jourdren, H.: Efficient shared memory message passing for inter-VM communications. In: 3rd Workshop on Virtualization in High-Performance Cloud Computing (VHPC '08), Euro-par 2008. (2008)
12. Huang, W., Koop, M.J., Gao, Q., Panda, D.K.: Virtual machine aware communication libraries for high performance computing. In: SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, New York, NY, USA, ACM (2007)