From the Proceedings of IEEE Cloud 2014

Bridging the Virtualization Performance Gap for HPC Using SR-IOV for InfiniBand

Malek Musleh*, Vijay Pai*, John Paul Walters†, Andrew Younge‡, and Stephen P. Crago†
*School of Electrical and Computer Engineering
Purdue University West Lafayette, IN 47907
Email: {musleh,vpai}@purdue.edu
†Information Sciences Institute
University of Southern California, Arlington, VA 22203
Email: {jwalters,crago}@isi.edu
‡Pervasive Technology Institute
Indiana University, Bloomington, IN 47408
Email: {ajyounge}@indiana.edu

***Abstract*—This paper shows that using SRIOV for InfiniBand can enable virtualized HPC, but only if the NIC tunable parameters are set appropriately. In particular, contrary to common belief, our results show that the default policy of aggressive use of interrupt moderation can have a negative impact on the performance of InfiniBand platforms virtualized using SR-IOV. Careful tuning of interrupt moderation benefits both Native and VM platforms and helps to bridge the gap between native and virtualized performance. For some workloads, the performance gap is reduced by 15-30%.**

***Index Terms*—SR-IOV, HPC, InfiniBand, Virtualization**

I. Introduction

With advancements in recent virtualization technologies, Cloud computing has realized a resurgence in recent years. This model offers two key benefits to consumers: 1) faster setup & deployment time and 2) reduced cost as customers are charged based on exact usage rather than total allocation times. Despite these benefits, earlier virtualization techniques had significant overheads costs that proved too costly for the benefits offered. Nonetheless, modern virtualization techniques have significantly reduced virtualization overhead to a point where the tradeoffs have become acceptable for many computing and storage environments. Nonetheless, virtualization has still not made major inroads in HPC environments. HPC environments run computationally intense, scalable algorithms for large inputs aiming to maximize utilization and throughput at all available nodes. I/O overheads are particularly unacceptable since they limit parallel speedup.

I/O virtualization is either performed in software with the assistance of the virtual machine monitor (VMM), or directly through the use of specialized hardware [1], [2], [3]. In the former approach, guest virtual machines (VMs) on a host are not able to access physical devices, so the VMM is responsible for routing traffic to/from the corresponding VMs. This method incurs repeated memory copies and context switches, leading to reduced performance. In contrast, specialized hardware allows direct access from within a guest VM [4]. The guest VM can thus performing I/O operations without duplicate overheads caused by VMM intervention. Figure 1 provides a high-level illustration of software virtualization and two hardware virtualization strategies: PCI-passthrough and SR-IOV [5]. The center block shows that only one VM has access to a specific NIC at a time, whereas the rightmost part shows how a single NIC can be shared across different VMs. In both PCI-passthrough and SR-IOV the VMM is bypassed, which eliminates the extra overhead mentioned earlier. This is in contrast to the leftmost component of Figure 1 that illustrates the: Virtual Machine Device Queue (VMDq) w/NetQueues technique, that requires the VMM to route incoming packets from the NIC to the correct VM.

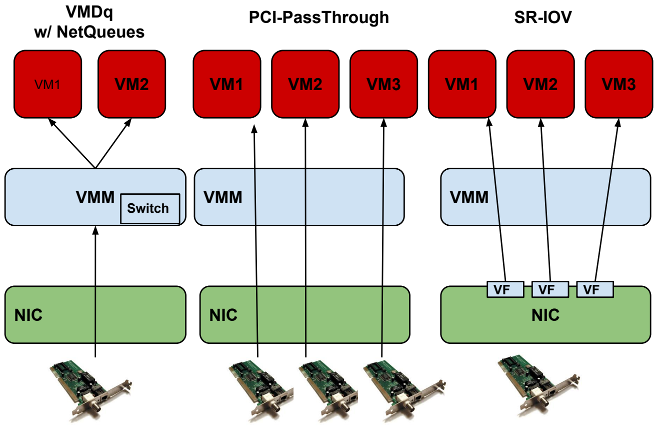

Fig. 1: Software Virt. vs. PCI-Passthrough vs. SR-IOV

As depicted in the Figure, SR-IOV compared to PCI-passthrough offers the advantage of concurrent sharing of physical devices among multiple VMs. Although the SR-IOV standard has existed for several years now, hardware vendor support for it on InfiniBand HPC interconnects has only started to emerge. A recent work by Jose et al. is the first to evaluate SR-IOV performance for InifiniBand clusters. Their initial experiments conclude that due to significant performance overhead for certain collective algorithms, it would seem unfeasible to adopt virtualization for HPC [3].

This paper offers a more thorough investigation and analysis of SR-IOV performance over InfiniBand that shows that virtualization overhead can be mitigated. We investigate the sources of overhead, and propose parameter-tuning optimizations in order to improve responsiveness of the VM to show that SR-IOV performance is competitive with a non-virtualized environment. More specifically, default parameter values that improve interrupt moderation are defined for native environments, but may be suboptimal for virtualized environments.

II. Background

For most of the last decade, InfiniBand has been regarded as the fabric of choice for HPC clusters because it is a high-bandwidth, low-latency interconnect. It is deployed in many commodity clusters, and as of TOP500 list of June 2011, used as the communication network in 41.2% of all systems. However, even as InfiniBand usage continues to grow, several factors continue to hinder full utilization of the technology's capabilities. The QDR InifiniBand with a bandwidth-rate of 40 Gbps, is often chosen more for its low latency rather than for its raw-bandwidth capabilities.

A. RDMA

InfiniBand provides the Remote Direct Access Memory (RDMA) feature, which provides for lower latency and allows for zero-copy transfers (i.e., place data at the desired target location without buffering). A client, also referred to as the initiator, issues a read request that includes a destination memory address in its local memory. The target, or server, responds by writing the desired data directly into the client's memory at the requested location. Eliminating the need to buffer messages, having the network adapters directly copy data minimizes operating system (OS) involvement across a low-latency fabric and provides a fast mechanism for transferring data.

B. IPoIB

IPoIB (IP-over-InfiniBand) is a protocol that defines how to send IP packets over IB. For example, Linux has a driver that implements this protocol which effectively creates a network interface for each IB port on the system. As a result, it makes a Host Channel Adapter (HCA) behave as an ordinary NIC. Although the driver provides this interface, IPoIB does not utilize the full capabailities of the HCA. For example, network traffic progresses through the normal IP stack, which effectively requires a system call for every message sent. Second, the host CPU is still required to process the data packets. Despite not being able to fully utilize the HCA, utiilzing IPoIB does mean that applications that use normal IP sockets will be able to send messages across the faster speed of the IB-link.

Jose et al. evaluate IPoIB performance between virtualized and non-virtualized environments using the NetPerf benchmarks. The observed result is more than 2x difference in performance, which they suggest may be rooted in the overheads of virtualizing the TCP/IP stack [3].

C. Single Root I/O Virtualization

SR-IOV allows a PCIe device to export a set of virtual functions as well as the number of PCI physical functions to enable sharing resources on the I/O device. The simplified architecture for server virtualization is shown in Figure 2. In this model, no passthrough is necessary, because virtualization occurs at the end device, allowing the hypervisor to simply map virtual functions to VMs to achieve native device performance with the security of isolation.

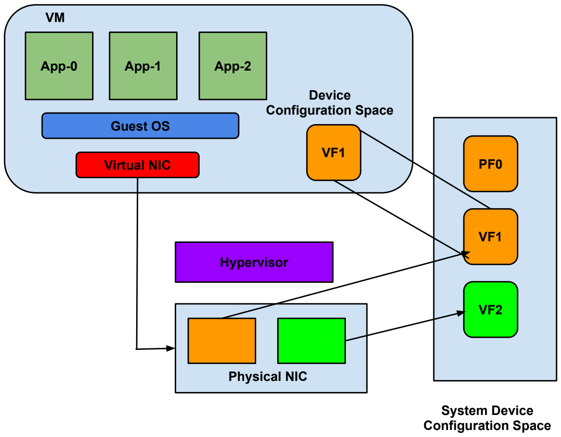

Fig. 2: SR-IOV

D. PCI passthrough

PCI-passthrough allows guests to have exclusive access to PCI devices for a range of tasks. It allows PCI devices to appear and behave as if they were physically attached to the guest operating system. Two key disadvantages of this method is that 1) it doesn't allow for device sharing among multiple guests and 2) supporting live-migration of the guest is very difficult–features which are highly valuable for HPC computing.

III. Motivation

Although virtualization has much to offer the HPC community, broad adoption of virtualization technology remains scarce mainly due to the following aspects.

A. Virtualization Performance Overhead

Traditional server virtualization brings sigificant performance overhead. The hypervisor, also known as the Virtual Machine Monitor (VMM), runs at the highest privilege, while the VM and guest OS run at the user-level VMM. While running the guest OS is embedded into the VMM when encountering a privileged operation. As a result, expensive context switching is required, especially during device access of which is unacceptable for HPC applications.

B. Resource coordination between VMs

Virtualization adoption for HPC systems would require efficient deployment and management of hundreds, if not thousands of VMs across the different nodes. How to quickly allocate and setup hardware resources dynamically for different VMs on a single node while maintaining low system overhead has proven increasingly difficult due to I/O overhead.

The functional advantages that SR-IOV offers over previous virtualization methods are 1) device sharing capabilities between VMs on the same host and 2) simpler requirements for detaching/reattaching device accesses to better facilitate live-migrations.

Specifically, by removing an extra layer of virtualization and memory copies by facilitating the virtualization on the device itself rather than in the VMM, SR-IOV has the potential to offer significant performance improvement over PCI-passthrough. Furthermore, it provides bandwidth multiplexing of the hardware resource across the VMs during simultaneous access, whereas this was not possible in PCI-passthrough. Although previous work has shown it is possible to implement live-migration with use of passthrough virtualization, the process remains quite complex [6], [7].

IV. Network Interrupt Tuning

In our evaluation, we find that proper tuning of network interrupts can have a considerable impact on InfiniBand performance. In this section, we discuss the various parameters that can be tuned to control the effect of network interrupts on virtualization performance. Experiment results for our findings are presented in Section VI.

A. Polling vs. Blocking Mode

Jose et al. discuss the methods used in InfiniBand for retrieving completion events: Polling and Event-based mode. As their names suggest, polling mode requires the user application to continously poll the completion queue (CQ), whereas Event-based/Blocking mode requires the application register for completion events. Polling mode provides for faster response, but depends on processor availability during the entire runtime for efficient utilization so that the OS is not performing ill-timed context switches.

Event-based mode is more suitable in cases when the host machine is fully subscribed as the user application will only access the completion queue when there is work to be processed. Upon receiving an event, the network interface generates an interrupt to signal the user thread that it is to be scheduled according to OS scheduling policies. It has been shown that Blocking Mode incurs additional overhead as a result of interrupt and OS scheduling latencies than compared to operating in Polling Mode.

Although they show that SR-IOV performance is comparable to that of native when operating in Polling Mode, there remains a noticable performance gap between virtualized and non-virtualized mode when operating in Blocking Mode [3]. Given that virtualized environments are deployed in fully subscribed scenarios, it is imperative that this performance gap be reduced in order to make SR-IOV suitable for HPC applications. To do so, we investigate the differences in performance between virtualized and non-virtualized modes with respect to interrupt handling.

B. Interrupt Management

Interrupt moderation is used to decrease the frequency of network adapter interrupts to the CPU. Mellanox network adapters use an adaptive interrupt moderation algorithm by default. The algorithm checks the transmission (Tx) and receive (Rx) packet rates and modifies the Rx interrupt moderation settings accordingly [8].

Interrupt Coalescing (IC) is used to reduce the interrupt processing overhead and becomes necessary when the minimum packet interarrival latency becomes comparable to the per-packet interrupt processing overhead. Unfortunately, IC can impact active network measurement tools that use closely spaced packet pairs, trains, or streams. In particular, capacity and available bandwidth estimation techniques can provide incorrect results if they ignore IC.

It is recommended practice that to improve application scalability and latency, interrupts requests (IRQs) be distributed among the processors. The IRQ Balancer in the kernel is responsible for *intelligently* distributing interrupts among the cpus. However, depending on the type of interrupt, such as the networking interrupt, it is advisable that the interrupt goes to only one core. In cases where the frequency of a specific interrupt class is high, then the IRQ balancer would assign it to a specific core to maximize cache efficiency [9].

Shared Receive Queues (SRQ) provides a model to efficiently share receive buffers across connections while maintaining the benefits offered by a connection oriented transport–good performance and reliability. Thus, the SRQ is a good candidate for achieving scalable buffer management.

InfiniBand provides an asynchronous event associated with a SRQ called SRQ_LIMIT_REACHED. This asynchronous event is triggered when the number of unoccupied entries in the receive buffer fall below the watermark threshold (preset by the application).

Increasing this threshold limit causes interrupts to be triggered more frequently as the receiver strives to maintain a larger number of free work entries. On the other hand, decreasing the threshold reduces the frequency of the interrupts as the receiver only needs to process the queue's work entries when there is only 1 free entry remaining.

Sur et al. propose a calculable solution for an appropriate low watermark threshold such that the event is not triggered too often nor too infrequently to ensure there is enough time to post the buffers so that the SRQ remains non-empty [10].

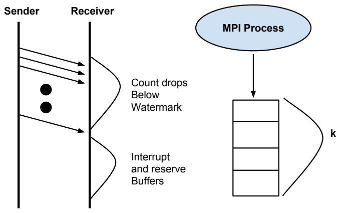

Fig. 3: SRQ Limit Watermark Threshold

V. Experimental Methodology

Our experimental testbed consists of two compute nodes featuring the Intel Sandy Bridge-EP platform. Each node has dual Intel Xeon E5-2670 operating at 2.6 GHz with a maximum turbo frequency of 3.3 GHz. The sixteen cores are split into two NUMA nodes with 8 physical cores and a 20 MB L3 cache. Each compute node has 48 GB of main memory split between the two NUMA nodes. The platform is equipped with one PCIe 3.0 slot. Table I illustrates additional details regarding our testbed setup. Our virtualized environment consists of two VMs, one on each physical node.

The VMs are configured to match as closely as possible to the native hardware. The same Linux kernel and OS are used in both on the native platform and within the VM. The number of vCPUs matches the number of host CPUs. Each vCPU is mapped to its corresponding physical CPU to avoid NUMA migration issues. Similarly, we choose a VM size with 40GB of memory. We choose this type of configuration based on the assumption that users running HPC-workloads would utilize the maximum amount of resources available.

TABLE I: Machine Configuration

Parameter	Value
Number of Physical Nodes	2
Processors per Node	16
NUMA	2 Sockets
Main Memory	48 GB RAM
Caches	L3-20 MB
Hard disk	600 GB
HyperThreading	Disabled
Operation System	CentOS 6.4 (RHEL6)
Linux Kernel	2.6.32-358.23.2.el6.x86_64
VMM	KVM [2]
MLX OFED	MLNX_OFED_LINUX-2.0-3.0.0-rhel6.4-x86_64 [11]
IB Card	Mellanox ConnectX-3 FDR (40 Gbps)

We evaluate our system at the network and micro-benchmark levels in addition to application-level benchmarks. We use the commonly available IB-Verbs benchmarks, and the OSU Micro-Benchmarks [12]. The IB-Verb benchmarks were used to evaluate performance at the low-level IB-Verbs, and the OSU benchmarks to depict the building blocks of HPC-applications:collective communication operations. All MPI experiments were run using MVAPICH2 1.9a2 [13].

Each experiment runs for a sufficient number of iterations in order to mitigate divergence due to OS interference. In addition, results are averaged across multiple trials in order to ensure their accuracy. Further, we employ process to core binding to avoid thread migration delays. In Table II, we describe each benchmark as well as its input size.

VI. Performance Results

In this section, we present the performance evaluation results of SR-IOV compared to native mode for the different type of communication benchmarks and algorithms. We do not include bandwidth performance results as Jose et al. show and we confirm that MPI bandwidth evaulation reveals that under both Polling & Event-based Modes, SR-IOV achieves near to peak bandwidth [3].

More specifically, we present our results with two objectives in mind:

- deeper statistical evaluation of performance evaluation to include min, max, and percentile analysis
- understanding how targeted parameter tuning, geared towards increasing interrupt frequency improves SR-IOV performance.

Simply comparing the default reported benchmark average, which depending on how they are computed by the respective benchmarks, can be insufficient under certain circumstances. For example, small input working-sets or smaller iteration counts may not be enough to mitigate indeterminate OS behavior (such as system daemons or context switches), such that even a few divergences may skew results.

The additional statistics offer deeper insight into SR-IOV tail-latency overhead. Taming the long latency tail has been a major focus in the Cloud Computing domain, because as systems scale, performance overhead is dominated by the slowest percentile [14].

Given the variations between trials of the same experiment, we use hypothesis testing to determine the relative performance between the native and SR-IOV cases. Specifically, we use a paired t-Test, with a two-tailed distribution to validate our NULL-hypothesis. The Null-Hypothesis being evaluated is: *the means between two populations being compare are equivalent:* $\mu_a == \mu_b$.

Table entries listed as < 0.001 indicate that the computed value can be approximated to 0. *P-values below 0.05 are generally considered statistically significant, while one of 0.05 or greater indicates no difference between the groups.*

A. Network-Level Performance

As indicated in section II, InfiniBand provides both send-recv & RDMA. We illustrate the performance of IB-Verbs in Figures 4–6.

As Mentioned in Table II We run ib_write_lat, ib_read_lat, and ib_atomic_lat for N=1000, N=10000, and N=100,000 iterations. Due to space constraints, we only plot the reported averages (median), and the 90th/95th percentiles for ib_write_lat. Because the virtual NIC disables message inlining, we also report our native results with message inlining being disabled. This is done so that comparisons to the VM experiments are fair. Figures 5 and 6 show that the 90th and 95th percentiles of the native runs are equivalent despite executing for different iterations. In contrast, the percentile values for the VM runs show notable difference for different iteration values. Specifically, we notice a drop in the computed percentile. We attribute this effect to the long-tail latency commonly encountered in virtualized environments. We use this insight to suggest 1) majority of program execution (90-95%) is comparable between native and SR-IOV and 2) most of the latency overhead can be attributed to the long-tail latency

TABLE II: Evaluated Benchmarks

	Benchmarks	**Input Size**	**Description**
Network-Level	ib_write_lat ib_read_lat	N=1000,10000,100000	latency test of RMDA writes latency test of RDMA reads
Micro-Level	osu_latency osu_barrier osu_alltoall osu_allgather osu_allreduce	N=100000	Send/Receive Latency Test MPI_Barrier Latency Test MPI_Alltoall Latency Test MPI_Allgather Latency Test MPI_Allreduce Latency Test
Macro-Level	CG	Class C	Uses a conjugate gradient method to compute an approximation of the smallest eigenvalue of a large, sparse, matrix
	LU		Simulated CFD application–Employs SSOR numerical scheme to solve a regular, sparse, triangular system
	SP		Simulated CFD application–Uses linear equations to Navier-Stokes equation
	EP		Kernel-only coordination of pseudorandom number generation at the beginning and result collection at end

effect. A similar result is illustrated in Figure 7 with the MPI-AlltoAll benchmark. In these figures, the x-axis represents message size in bytes and the y-axis illustrates the execution time in microseconds (μs), with the line plots illustrating the average execution time of the 90/95/99th percentiles. The graph illustrates that performance gap of the 90th percentile between native and virtualized runs are more comparable, whereas there is a significant difference between the two modes in the 99th percentile. Similar behavior is observed with the MPI-allReduce and MPI-allGather benchmarks, but are not shown here due to space constraints.

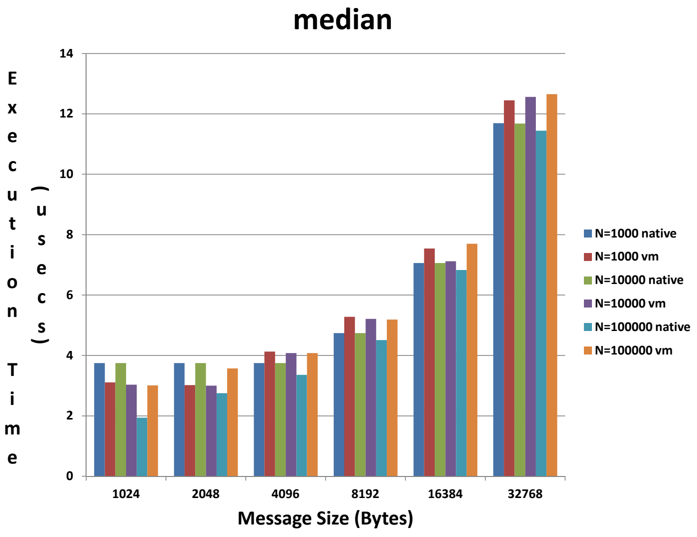

Fig. 4: IB-Verbs RDMA Write Latency

B. *Micro-level MPI Benchmarks*

We next look at evaluating the basic communication operations (barrier + send/receive), upon which the more complex collective algorithms (e.g. allReduce, allGather, AlltoAll) are based on. Table III depicts the 1) average reported latency across 15 trials for each configuration between the means of

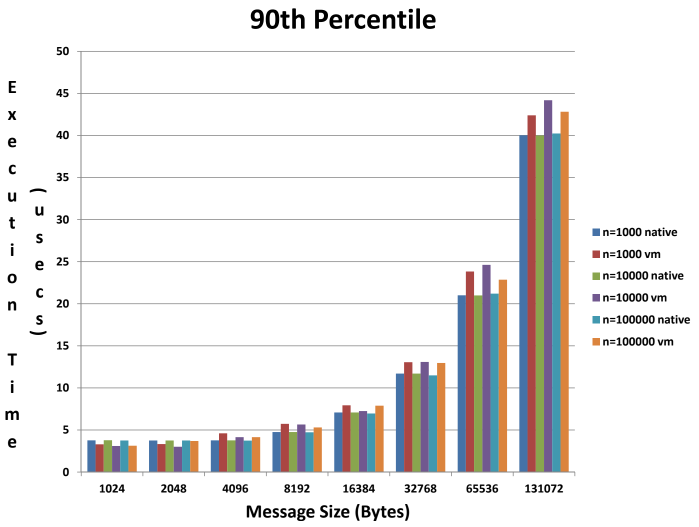

Fig. 5: IB-Verbs RDMA Write Latency

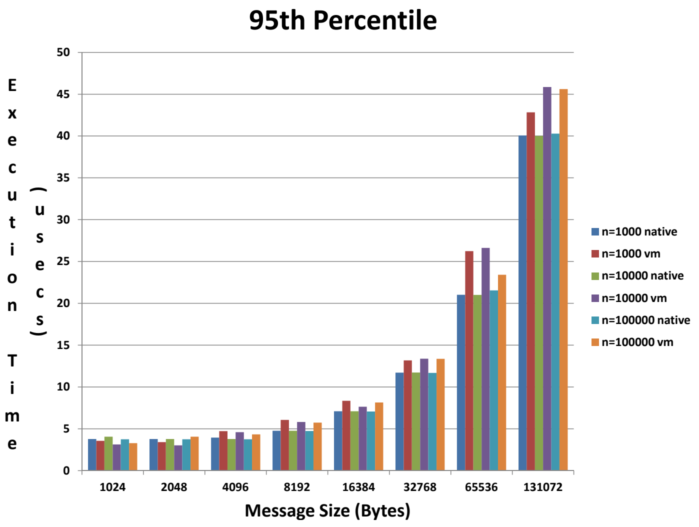

Fig. 6: IB-Verbs RDMA Write Latency

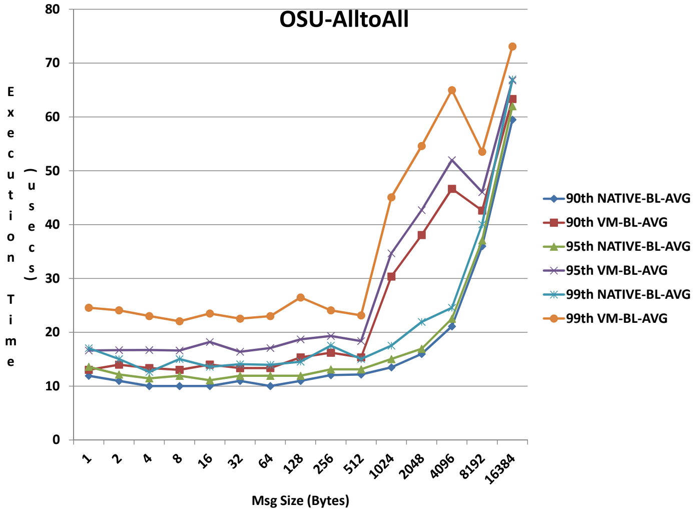

Fig. 7: Performance for N=1000000

TABLE III: Latency (μs) Blocking Mode Avg

Bytes	Native-RX88	Native-RX1	VM-RX88	VM-RX1
64	5.05	3.77	7.06	5.4
128	5.23	3.8	7.12	5.36
256	5.61	4.5	7.47	6.27
512	5.86	4.64	7.62	6.68
1024	6.23	5.39	8.03	7.58
2048	7.74	6.62	8.90	8.65
4096	8.86	7.61	10.01	9.40
8192	11.45	9.98	11.60	10.78
16384	16.2	14.45	14.8	14.2

the four populations (Native-RX88, Native-RX1, VM-RX88, & VM-RX1) for the osu-latency benchmark.

Columns noted with -RX88 correspond to the HW-NIC Rx-frame count trigger for interrupt moderation being set to the default value 88, where RX1 corresponds to the Rx-frame count set to 1. This table shows that the average latency for a Message Size (M) <= 16384 bytes, VM-RX1 is competitive with that of Native-RX1, and both are better than the default.

Although we see from Table III tuning the RX-frame count improves both the native and virtualized experiments of osu-latency benchmark, that is not always the case. Table IV shows the computed averages across 15 trials and P-value for the osu-AllGather benchmark between Native-RX88 and Native-RX1. Here, we see that the average between the two configurations are either statistically equivalent when the P-value > 0.05 or that Native-RX1 is statistically worse than Native-RX88 when the P-value < 0.05. Our detailed results (not-included) show that VM-RX88 is worse than VM-RX1 even though Native-RX88 is equal to or better than Native-RX1.

In Figure 8 we illustrate the performance impact of changing the RX-frame trigger count has with varying number of active processes for the osu-AllGather benchmark. The x-axis denotes the message size while the y-axis represents execution time in (μs). Experiments are run for N=2,4, and 8 total processes with processes per node (PPN) = 1,2,4 respectively. The graph again confirms the competitive performance between native and VM for a Message Size (M) <= 16384 bytes. Correspondly, Table V presents the calculated P-Value for the osu-AllGather comparing Native RX88 and VM-RX1.

TABLE IV: OSU-AllGather (μs) Blocking Mode Avg

Bytes	Native-RX88	Native-RX1	P-Value
8	2.92	2.91	0.84
16	2.87	2.89	0.66
64	2.76	2.97	< 0.001
128	3.60	3.67	0.10
256	3.82	3.96	< 0.001
512	4.128	4.21	< 0.001
1024	4.92	4.87	0.36
2048	5.95	6.02	< 0.001
4096	6.76	7.03	< 0.001
8192	10.12	10.14	0.12

TABLE V: OSU-AllGather Blocking Mode P-Value

Bytes	N=2,PPN=1	N=4,PPN=2	N=8,PPN-4
128	0.53	< 0.001	< 0.001
256	0.05	< 0.001	< 0.001
512	0.12	< 0.001	< 0.001
1024	0.65	< 0.001	< 0.001
2048	0.008	< 0.001	0.21
4906	0.12	0.0015	< 0.001
8192	0.21	< 0.001	0.49
16384	< 0.001	< 0.001	< 0.001

Figure 9 shows the performance for the osu-barrier benchmark with adjustment to the SRQ-Limit parameter. On the x-axis there are three main groups of results: Avg/Min/Max Latency. Within each subgroup we present results for a varying number of total active (N) processes, and the number of process per node. The y-axis represents the execution time in μs. We evaluate the adjustment for both Polling & Interrupt Modes. We find that increasing the SRQ-Limit threshold results in about a 15% (13 down to 9 (μs) latency improvement compared for Blocking Mode operation, and also a slight improvement in Polling Mode.

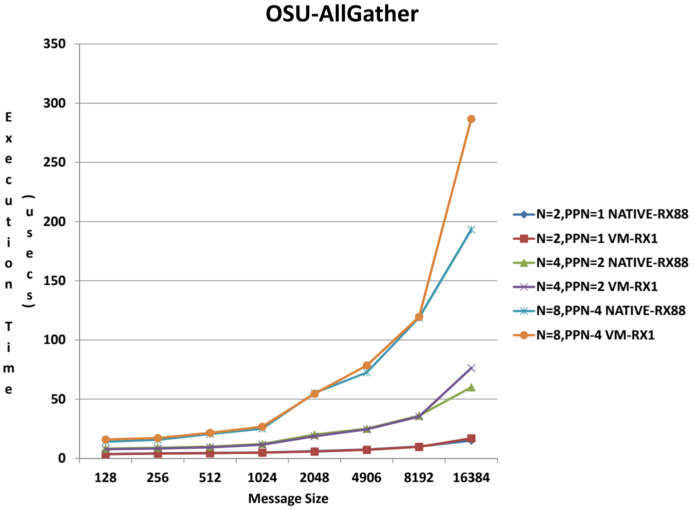

Fig. 8: Performance between Native-RX88 / VM-RX1

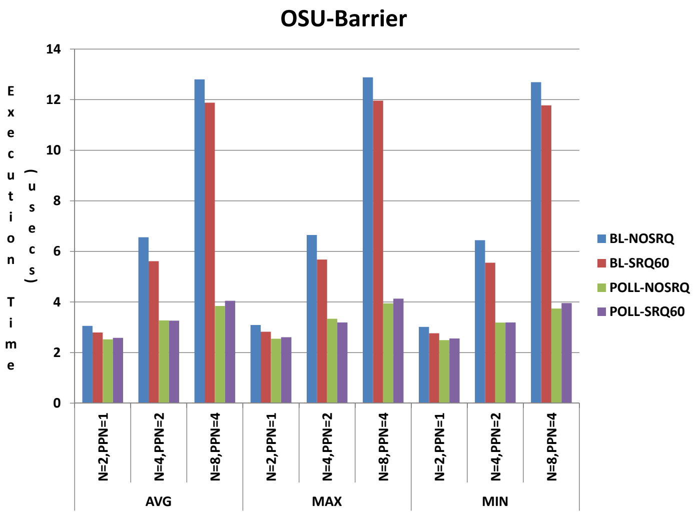

Fig. 9: Performance w/varying SRQ Limit

C. Performance Analysis

Additonal breakdown of the per-iteration performance results illustrate a couple of key points. In comparing the absolute maximum latency encountered for an iteration within the same experiment between native and VM, we find the values to be mostly consistent for small and medium. However, there are few instances where the SR-IOV experiment may experience a comparatively long delay before completion. Table VI depicts the absolute maximum latency experienced when running the osu-bcast benchmark for N=100000 iterations.

The 99th percentile numbers reported in Table VI indicate that Native somewhat outperforms VM in long-tail latency. This effect is further exacerbated in the maximum latency numbers seen. For the large message sizes (262144–4194304 bytes) the VM experiment experiences an absolute maximum per-iteration latency that is a factor of 4-5 times larger than its native counterpart. This is likely caused by inefficiencies of OS scheduling policies when communicating very large messages. Although such instances are clear outliers occuring rarely, they are able to skew the computed average for experiments of short duration.

TABLE VI: OSU-Bcast
Max-Value | 99th Percentile (μs)

	Max-Value		99th Percentile	
Bytes	**Native**	**VM**	**Native**	**VM**
1	382.9	303.03	6.91	11.92
2	423.90	307.08	7.15	10.01
4	275.85	299.93	7.15	10.11
8	276.09	288.96	7.15	10.0
16	416.99	303.03	7.15	10.0
32	275.85	279.90	7.15	10.0
64	371.933	277.99	7.15	10.0
128	412.94	282.05	7.15	10.96
256	282.05	289.92	7.15	10.96
512	381.95	281.10	9.06	10.01
1024	429.15	288.96	9.06	11.91
2048	404.12	295.88	10.01	12.87
4096	418.90	296.12	10.01	13.83
8192	406.03	305.89	14.06	15.97
16384	405.07	309.94	18.12	20.98
32768	424.15	324.97	25.034	27.90
65536	413.179	369.07	35.05	36.97
131072	436.07	345.96	56.03	59.13
262144	489.9	2902.03	96.01	105.86
524288	585.08	3915.07	185.97	191.93
1048576	699.99	2436.16	345.0	361.93
2097152	1063.82	4861.83	662.01	693.01
4194304	1678.23	5449.77	1321.01	1415.97

D. Macro-level MPI Benchmarks

In this subsection we present performance results for four application benchmarks from the NPB-2.4 suite: *CG*, *LU*, *SP*, and *EP*.

For reasons discussed in Section VI-A, we disable message inlining and use of shared memory optimizations by MVAPICH2 for the native trials to present a fairer comparison. Through experimentation we determine that message inlining does offer a performance boost for some benchmarks (7% for CG), but not others (LU) when operating in fully subscribed mode. Due to space constaints, we do not illustrate the performance impact of these optimizations for native.

Figures 10–13 show the performance for the four NPB benchmarks illustrated showing the impact of the interrupt parameters adjusted. The x-axis indicates the trial number, while the y-axis depicts the execution time in *seconds*. For CG, LU, and EP we illustrate the results for 32 processes, whereas for SP we illustrate for 16 processes. Due to the algorithmic structure of SP, we are unable to experiment with a larger number of processes without oversubscribing the nodes. We present the baseline native performance against VM performance with the interrupt parameters tuned according. We vary the SRQ_Limit, the RX-Frame trigger count, and the interrupt moderation timer window.

For each of the graphs, we see that tuning the interrupt parameters improves VM performance compared against the native default value for RX frame count. In contrast with the micro-benchmarks, we see that increasing the Rx-frame rate, or the software SRQ_Limit threshold, has little impact on three of the four benchmarks, the exception being EP. However, by adjusting the smallest & largest interrupt moderation timers to 40 & 1000 (μs) respectively, we see about a 7% performance improvement for VM experiments.

Timer moderation values represent the window in which an interrupt would occur: rx-usecs-low represents the soonest, and rx-usecs-high the latest. For latency-bound traffic interrupts are usually triggered based on rx-usecs-low, while bandwidth-bound traffic triggers on rx-usecs-high. By increasing the timer window, the NIC is better able to support different program phases: ones that are predominately latency-bound and others that are bandwidth-bound.

Increasing the timer interrupt window, either through increasing the RX frame trigger count or by increasing the moderation window seems to increase overhead by delaying the critical path for a benchmark such as EP (Figure 12) that has little / no communication.

In contrast, the virtualized performance for benchmarks CG, LU, and SP (Figures 10, 11, and 13 respectively) improves as the interrupt window is increased. For example, Figure 10 illustrates that having the default RX frame trigger performs better than RX=1. Performance is further improved if we also increase the interrupt moderation window. This is in contrast to micro-benchmarks where RX=1 improved performance.

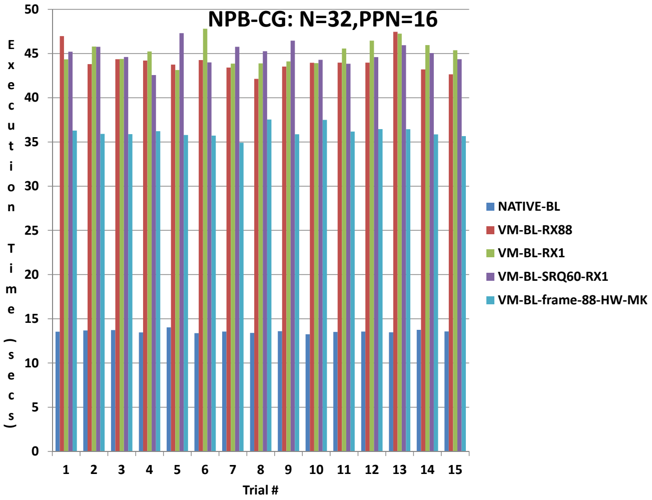

Fig. 10: Performance Native / VM

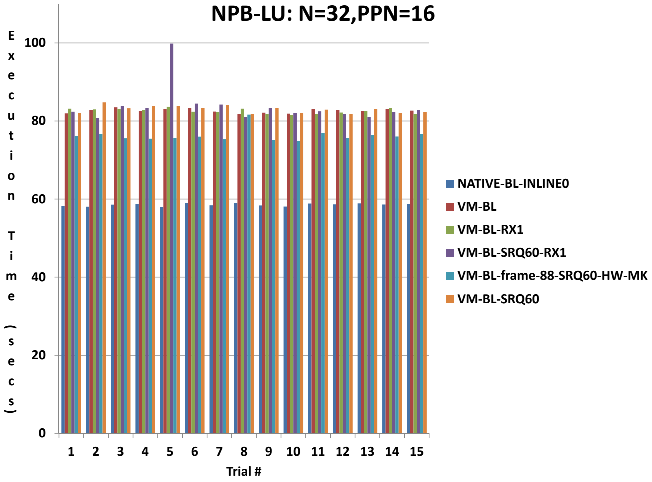

Fig. 11: Performance Native / VM

VII. Related Work

Prior work can be categorized into either software-based or hardware-based categories. In this section we focus our discussion of related works that target hardware-based solutions because they 1) generally demonstrate better performance compared to software-based ones and 2) software-based solutions are primarily focused on the Xen platform [15], [16].

A. PCI Passthrough

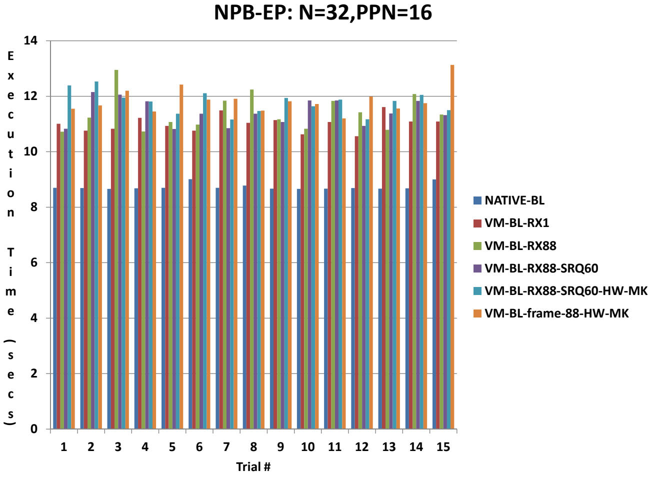

Fig. 12: Performance Native / VM

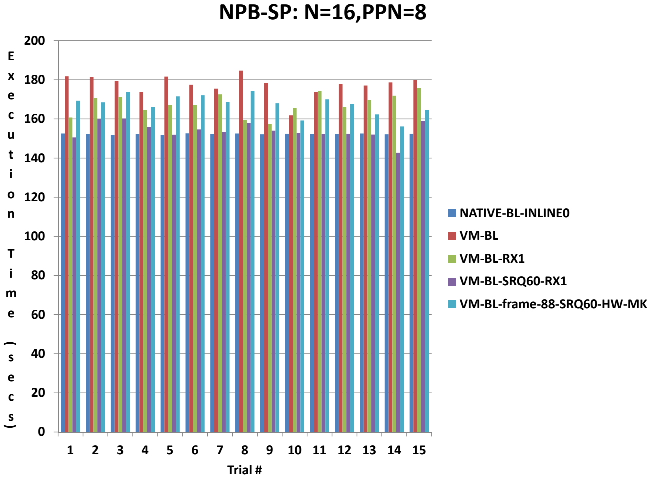

Fig. 13: Performance Native / VM

Regola et al. evaluate PCI-passthrough performance for both HPC applications and disk I/O operations under different hypervisors: KVM, Xen, and OpenVZ [17]. They show that utilizing PCI-passthrough that despite allowing improved scalability when compared to other virtualization techniques, such as para-virtualization, overhead significantly increases. Despite recent advances in eliminating CPU overhead, high I/O overhead remains.

B. SR-IOV

1) Ethernet: Several studies have demonstrated that SR-IOV is significantly better than software-based solutions for 10GigE networks [18]–[20]. Liu et al. provide a detailed performance evaluation on the environment of SR-IOV capable 10GigE Ethernet in KVM [18].

They study several important factors that impact network performance in both virtualized and native systems. Dong et al. have conducted experiments to compare SR-IOV performance with a paravirtualized network driver. The results show that SR-IOV can achieve high performance, high scalability, and with a low CPU overhead at the same time [19].

Huang et al. address two important issues: redundant interrupts and single-threaded NAPI, which affect performance and scalability of SR-IOV with 10GigE network. Their results also demonstrate that SR-IOV approach can achieve high performance I/O in a KVM-based virtualized environment

[20]. Furthermore, previous studies with Xen demonstrated the ability to achieve near-native performance in VM-based environment for HPC [21], [22], [23], [24].

2) InfiniBand: Jose et al. compare SR-IOV performance against native on InfiniBand to determine if SR-IOV is prime-time for HPC applications. With their preliminary experiences, they show that for the performance of MPI and PGAS point-to-point communication benchmarks over SR-IOV with InfiniBand is comparable to that of the native InfiniBand hardware, for medium and large message lengths. However, the performance of certain MPI collective operations over SR-IOV with InfiniBand is noticably worse when compared to the native designs [3].

Although they show that there exists virtualization overhead, a more thorough evaluation of SR-IOV performance remains to be done in order to fairly evaluate SR-IOV's potential. As discussed in the above section, current research on SR-IOV mainly pays attention to the environment of 10GigE network.

Given that InfiniBand usage continues to grow within the HPC community, it is critical for researchers to fully understand the benefits and performance bottlenecks to be able to fully utilize its potential. This paper concentrates on this scenario, which is different from other works, and provides a more detailed investigation into how performance can be tuned evaluated under a broader set of applications than [3].

More importantly, we show that network parameters tuned for native performance may be suboptimal in a virtualized environment. In tuning interrupt-specific parameters, we show that virtualization performance improves as a result of increased system responsiveness.

VIII. Conclusion

The HPC community has avoided adopting virtualization due to CPU and I/O overhead, while pursuing complex interconnects that offer extremely low-latency interconnect fabrics. In this paper, we present our detailed evaluation of SR-IOV with InfiniBand. Further, we find that virtualized performance for HPC workloads can be competitive with that of native if network-interrupt parameters are tuned to increase the responsiveness of the SR-IOV system. Additional analysis also shows that much of the remaining overhead is due to long-tail latency. For some benchmarks, the performance gap is reduced by 15-30%, thus increasing the attractiveness of SR-IOV for HPC platforms.

References

[1] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor," in *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2001, pp. 1–14.

[2] Kernel virtual machine. [Online]. Available: http://kvm.qumranet.com/

[3] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, "Sr-iov support for virtualization on infiniband clusters: Early experience," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 385–392, 2013.

[4] J. R. Santos, Y. Turner, G. Janakiraman, and I. Pratt, "Bridging the gap between software and hardware techniques for i/o virtualization," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ser. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 29–42.

[5] Single root i/o virtualization. [Online]. Available: SRIOV,http://www.pcisig.com/specications/iov/singleroot/

[6] A. Kadav and M. M. Swift, "Live migration of direct-access devices," in *Proceedings of the First Conference on I/O Virtualization*, ser. WIOV'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 2–2.

[7] R. Takano, H. Nakada, T. Hirofuchi, Y. Tanaka, and T. Kudoh, "Cooperative vm migration for a virtualized hpc cluster with vmm-bypass i/o devices," *2012 IEEE 8th International Conference on E-Science*, vol. 0, pp. 1–8, 2012.

[8] Performance tuning for mellanox network adapters. [Online]. Available: http://www.mellanox.com/related-docs/prod_software/Performance_Tuning_Guide_for_Mellanox_Network_Adapters.pdf

[9] Irq balancer. [Online]. Available: http://irqbalance.org/documentation.html

[10] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda, "Shared receive queue based scalable mpi design for infiniband clusters," in *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, ser. IPDPS'06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 101–101.

[11] Mlnx ofed (openfabrics enterprise distribution). [Online]. Available: http://www.mellanox.com/related-docs/prodsoftware/PB_OFED.pdf

[12] Osu micro-benchmarks. http://mvapich.cse.ohio-state.edu/benchmarks.

[13] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. K. P, D. Ashton, W. Gropp, D. Buntinas, and B. Toonen, "Design and implementation of mpich2 over infiniband with rdma support," in *In In Proceedings of Intl Parallel and Distributed Processing Symposium (IPDPS 04*, 2004.

[14] Tail latency. [Online]. Available: http://highscalability.com/blog/2012/3/12/googletamingthelonglatencytailwhen-moremachinesequal.html

[15] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the xen virtual machine environment," in *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*, ser. VEE '05. New York, NY, USA: ACM, 2005, pp. 13–23.

[16] P. Apparao, S. Makineni, and D. Newell, "Characterization of network processing overheads in xen," in *Proceedings of the 2Nd International Workshop on Virtualization Technology in Distributed Computing*, ser. VTDC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 2–.

[17] N. Regola and J.-C. Ducom, "Recommendations for virtualization technologies in high performance computing," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 409–416.

[18] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 gbe nics with sr-iov support." in *IPDPS*. IEEE, pp. 1–12.

[19] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, Jan. 2010, pp. 1–10.

[20] Z. Huang, R. Ma, J. Li, Z. Chang, and H. Guan, "Adaptive and scalable optimizations for high performance sr-iov." in *CLUSTER*. IEEE, pp. 459–467.

[21] W. Huang, M. J. Koop, Q. Gao, and D. K. Panda, "Virtual machine aware communication libraries for high performance computing," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, ser. SC '07. New York, NY, USA: ACM, 2007, pp. 9:1–9:12.

[22] W. Huang, J. Liu, B. Abali, and D. K. Panda, "A case for high performance computing with virtual machines," in *Proceedings of the 20th Annual International Conference on Supercomputing*, ser. ICS '06. New York, NY, USA: ACM, 2006, pp. 125–134.

[23] J. Liu, W. Huang, B. Abali, and D. K. Panda, "High performance vmm-bypass i/o in virtual machines," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ser. ATEC '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 3–3.

[24] W. Huang, J. Liu, M. J. Koop, B. Abali, and D. K. Panda, in *VEE*, C. Krintz, S. Hand, and D. Tarditi, Eds.