Efficient High-Performance Computing with Infiniband Hardware Virtualization

Tiago Pais Pitta de Lacerda Ruivo*†, Gerard Bernabeu Altayo*, Gabriele Garzoglio*, Steven Timm*, Hyun Woo Kim*, Seo-Young Noh[#], Ioan Raicu†+

*Scientific Computing Division, Fermi National Accelerator Laboratory
†Department of Computer Science, Illinois Institute of Technology
[#]Global, Science Experimental Data Hub Center, Korea Institute of Science and Technology Information
+Math and Computer Science Division, Argonne National Laboratory

tpaispit@hawk.iit.edu, {gerard1,garzogli,timm,hyunwoo}@fnal.gov, rsyoung@kisti.re.kr, iraicu@cs.iit.edu

***Abstract*— It has been widely accepted that software virtualization has a big negative impact on high-performance computing (HPC) applications performance. This work explores the potential use of Infiniband hardware virtualization in an OpenNebula cloud towards the efficient support of MPI-based workloads. We have implemented, deployed, and tested an Infiniband network on the FermiCloud private IaaS cloud. To avoid software virtualization towards minimizing the virtualization overhead, we employed a technique called Single Root Input/Output Virtualization (SR-IOV). Our solution spanned modifications to the Linux's Hypervisor as well as the OpenNebula manager. We evaluated the performance of the hardware virtualization on up to 56 virtual machines connected by up to 8 DDR Infiniband network links, with micro-benchmarks (latency and bandwidth) as well as with a MPI-intensive application (the HPL Linpack benchmark).**

Keywords: Infiniband; virtualization; Cloud Computing; OpenNebula; SR-IOV; Linpack; HPC.

I. INTRODUCTION

FermiCloud is a private cloud providing Infrastructure-as-a-Service services to Fermilab employees and users and it manages dynamically allocated services for both interactive and batch processing. As part of the computing infrastructure of the Laboratory, this distributed computing system complements the bigger FermiGrid project, a distributed campus infrastructure that manages conversely statically allocated compute and storage resources for batch processing. In particular FermiGrid is used to run compute-intensive jobs related to experiments conducted here at Fermilab as well as in other locations that collaborate with it like the LHC. FermiCloud provides additional resources to the Grid by providing Virtual Machines that run on the Cloud Infrastructure and increase the computing capacity. These virtual machines (VM) imitate a physical computer in all its features and functionalities and it is possible to run several of them on a physical machine, sharing its resources and so optimizing its resource utilization. Nevertheless, the usage of VMs implies managing the virtualization and sharing of the physical hardware (HW) and resources. This virtualization introduces overheads in performance.

As part of a national laboratory, the FermiCloud team is especially interested on expanding its functionality to provide a useful platform to the scientific investigation performed on Fermilab. A big part of it comes from being able to run scientific applications and models. However, scientific computing relies on compute-intensive and data-intensive jobs that have to be coordinated among several nodes. Although cloud computing provides a better utilization of the resources, the overhead introduced by the virtualization can make the application too inefficient and as of today, most scientific computing is still done on High Performance Computing (HPC) clusters and supercomputers which do not support virtualization. These applications are also distributed among several nodes, usually using the Message Passing Interface Protocol (MPI) that is very sensitive to changes in latency, such as the virtualization overhead. [12, 26]

A fast and reliable network in and among virtual machines is a key element for a cloud system to be capable of running scientific applications, facilitating the transfer of data and communication between VMs [12]. Infiniband is an especially interesting technology since it is one of the interconnect links offering one of the highest throughputs and lowest latency, guaranteeing QoS and scalability. It is often used in supercomputers and high performance computing [19].

The main challenge to overcome in the deployment of the network is the already discussed overhead introduced when virtualizing the hardware of a machine to be used (and shared) by the VMs. This overhead slows drastically the data rate reducing the efficiency of using a fastest technology like Infiniband. To overcome the virtualization overhead we used a technology called SRIOV that achieves device virtualization without using device emulation by enabling a device to be shared by multiple virtual machines. With SR-IOV, a PCIe device can export multiple virtual functions besides physical functions. These virtual functions that reside in the device itself actually share the resources that are provided by the device. This model allows the hypervisor to simply map virtual functions to virtual machines, which can achieve the native device performance even without using pass through [6].

Using SR-IOV with Mellanox InfiniBand cards means installing new firmware in the devices [17] and changes in Cloud system, OpenNebula [25] in our case. [14]. *This work focuses on how this Infiniband network was implemented and deployed on the VMs adapting it to the existent FermiCloud infrastructure by deploying it in the host machines, configuring the SR-IOV and migrating this work to OpenNebula, the cloud manager, focusing on simplicity and compatibility with the rest of functionalities. After the deployment, a battery of tests was performed to test the efficiency of this virtualization (SR-IOV).*

In this work, we define efficiency as the percentage of a metric in a virtualized environment compared with the same metric in the non-virtualized environment with the same resources. To measure the performance and efficiency of the virtualization, several tests were performed: micro-benchmarks to measure latency and bandwidth as well as the HPL Linpack benchmark to measure the efficiency of a real scientific oriented application. All these benchmarks were scaled to up to 8 nodes with a maximum of 7 VMs per node, the maximum permitted by our hardware. ***The results were excellent, with VMs reaching the same throughput and the same latency (depending on the size of the messages transmitted) of the native machines.*** Some latency overhead in small messages lowered the efficiency of the Linpack benchmark to as low as 70%.

The rest of the paper is organized as follows. Section 2 gives a small overview of the technologies used and describes how the deployment was done. Section 3 focus on the performance evaluation, describing the battery of tests and its results. Section 4 describes published work related with this topic like SR-IOV deployments on other networks or other Infiniband virtualizations. Finally in section 5, we present the conclusions based on the performance evaluation as well as possible future work.

II. PROPOSED WORK

This work explores the potential use of Infiniband hardware virtualization in an OpenNebula cloud [25] towards the efficient support of MPI-based workloads. We have implemented, deployed, and tested an Infiniband network on the FermiCloud private IaaS cloud. To avoid software virtualization towards minimizing the virtualization overhead, we employed a technique called SR-IOV. Our solution spanned modifications to the Linux's Hypervisor as well as the OpenNebula manager. We first cover a high-level overview of Infiniband followed by a description of the SR-IOV virtualization. We then continue drilling into the challenges brought by the network virtualization, requiring additional new functionality.

A. Infiniband

Infiniband provides point-to-point bidirectional serial links between processors or with high-speed peripherals (often time the bottleneck of a system) for a fast Input/Output (I/O) and has support for multicast operations. It uses a switched fabric topology (nodes connect using switches for a point-to-point communication, usually without the need of a hierarchical structure). Each processor has a Host Channel Adapter (HCA) and each peripheral has a Target Channel Adapter (TCA) that exchange data as well as metadata for security or QoS and that are the interface with the machine. These adapters can handle the entire Infiniband protocol stack without using CPU (more efficient) and bypassing the kernel, communicating directly with the application avoiding the corresponding overhead [19].

The throughput depends on the hardware. At a physical level there are several configurations with data rates ranging from 2.5 to 25.78125 Gb/s in each direction. Several links can links can be aggregated in sets of 4x of 12x, multiplying the corresponding data rates by these values. However, redundancy encoding slows this speed. Another attractive characteristic of Infiniband is offers very low latency that can go down to 100ns, making the end-to-end latency usually in the order of microseconds [19].

Infiniband specification only includes a list functions that must be implemented in order to build the protocol stack. The de-facto standard implementation is called OFED, developed by OFA, that provides the entire protocol stack, the kernel modules used to manage the communication with the device, as well as several monitoring and configuration tools [17]. In our case, an implementation provided by the Linux distribution (Scientific Linux), containing most of the tools, had to been used since the one provided by the card manufacturer was not compatible with newer kernel versions. Another important element of an Infiniband deployment is the Subnet Manager and Administrator, in this case OpenSM, which provides and maintains routing tables on the IB hosts. The Switch we used is a dummy switch and does not have the necessary logic to do the routing. One Subnet Manager must run in one of the physical nodes in IB network.

An interesting feature of the Infiniband stack is IPoIB (IP over Infiniband), the protocol that defines how to send IP packets using Infiniband by creating a normal IP network interface. This wastes some of the functionalities and efficiency of the higher layers of the Infiniband protocol stack and drops performance but the user can now use the much wider set of applications built for TCP-IP and overpass some of the virtual functions restrictions.

Because of all the previous features, Infiniband is commonly used not only in super computing and HPC but also in clouds and datacenters that require a fast network. Infiniband is common in the TOP500 supercomputers, in military applications and in the financial sector [19].

B. SR-IOV Overview

SR-IOV is a standard developed by the PCI Special Interest Group for Virtualized Servers. It uses the concepts of physical and virtual functions. A Physical Function (PF) is a full-featured PCIe function discovered, managed and manipulated like any other physical PCIe device. PFs can control the PCIe device and have full configuration capabilities (besides doing the I/O). A Virtual Function (VF) is a lightweight function that lacks the configuration resources and is limited to processing the I/O streams, to move data. VFs cannot be configured since they don't have access to the physical device. A VF is subordinated to a PF and all the VFs that depend on that PF have the same configuration. If a VF could change its

configuration options, would also change the options of its PF as well as all the other VFs that depend of it [6,8].

The virtualization consists on dividing the card on a set of PFs and VFs that appear in the bare metal machine as a set of independent PCIe Infiniband devices. In this environment, we can associate each VF directly to a virtual machine that has the exclusive use of that function, and therefore sharing a physical IB resource without using any device emulation in hypervisor or in user space. [22]

The virtualization is mostly done in hardware (needing support by the device as well as the BIOS. Each PF and VF receive a unique PCI Express Requester ID that allows the I/O Memory Management Unit (IOMMU) to differentiate the traffic going to the different VFs resulting in non privileged data flows from the PF to one VF without affecting the others [17]. Since a VF cannot be treated as a full PCIe device, the OS or hypervisor must also be aware that it is not to block any configuration options or tools that can change the other VFs [6, 17]. The specification indicates that each device can have up to 256 VFs. These numbers are theoretical maximums since the virtualization needs hardware resources, and thus the practical upper bound is 63 VFs (plus 1 PF) [22].

This solution is much more efficient than device emulations in hypervisor which when receiving or sending data must interrupt a CPU core to inspect the packet and determine which VM should receive it, the data packet to the CPU core that serves the corresponding VM and interrupt it to process the I/O. This process introduces a significant latency and CPU workload. By using SR-IOV, we bypass this virtualization assigning a physical port directly and exclusively to a VM maximizing the utilization of the HW virtualized device [6, 22].

C. SR-IOV Implementation

The virtualization configuration in the bare metals is a simple process but very poorly documented, probably because it is fairly recent. It requires a change in the different components involved in the virtualization. To enable it on the hardware, a Firmware change is required that enables the functionality and specifies the number of PFs and VFs to be shown to the machine, each one appearing as an independent device with its own PCI address. The maximum number of VFs supported by our card is 7 and it only has 1 PF (is a single port card).

The process is sensitive since it is very low-level software that doesn't have any error check. A wrong configuration may cause the card not to load when the system boots and can lead to a crash of the entire system. Recovering the card after a bad Firmware update requires special manufacturer HW.

Since the virtualization also has to be supported by the OS, we have to enable it and specify the proper options in the BIOS as well as in the Infiniband card modules. Besides, the kernel has to have the IOMMU (I/O Memory Management Unit) activated to allow the communication of the VFs with memory and the PFs without using the CPU.

After this, the bare metal appears to have 8 Infiniband cards and we can pass each one of them directly to a VM. To manage the virtualization we used the Linux's Kernel-based Virtual Machine (KVM), a virtualization solution, and libvirt, the virtualization API that OpenNebula[25] uses and that lets us create, personalize, destroy and manage VMs.

To use Infiniband in the virtual machines, it is necessary to acknowledge that virtual functions have several limitations. As it was discussed before, virtual functions cannot be configured since they emulate one physical function. Because of that, some Infiniband tools do not work on VFs, mainly the configuration ones and the ones that belong to higher layers of the protocol stack. To overcome this, Infiniband offers the possibility of using the already mentioned IP protocol over Infiniband (IPoIB). Most applications use IPoIB to establish the connection and then change to the Infiniband protocol stack. This maintains the original Infiniband performance but uses the familiar IP directions, much more simple than the Infiniband's LID. Besides, the Subnet Manager doesn't assign individual directions to the Virtual Functions and so they are not visible from other hosts [1]. The applications used in our tests use this technique like OpenMPI, the MPI application used or Perftest, a ping-pong benchmark. To use IPoIB is necessary to create an Infiniband network interface with its IP and load the corresponding modules that recognize that interface and make the translation of the IP to the corresponding HCA.

In the bare metals, it is not even necessary to install the entire set of Infiniband support but only to load the kernel modules that handle the communication with the card (including the ones managing IPoIB). In our Linux distribution that can be done by activating a service called Rdma, also activated in the VMs. Besides, at least one of the host machines (can be more than one to avoid having a single point of failure) must be running the already mentioned subnet manager to handle the routing. Because of the previous experience with OFED and its incompatibilities with the driver, we focused here and throughout the entire project on finding RPMs that were already tested on this Linux distribution and avoided compiling applications since that makes the deployment more complex and fail more often because of the referred incompatibilities. By using libvirt, a virtualization API, it is possible to create VMs with the virtualized devices assigned from each host machine. To automate it and centralize it, the work must be migrated to OpenNebula[25], FermiCloud's cloud manager.

D. SR-IOV and OpenNebula

OpenNebula and its KVM need to be changed because SR-IOV needs the support of the hypervisor since it has to be aware that the VFs are not real devices. Besides, it must support the integration and manage the assignment of different VFs to each Virtual Machine: The configuration of its network, how to handle the different actions (migration, live migration, saving, killing, etc). For that, the South African Center for High Performance Computing implemented a VMM (Virtual Machine Monitor) driver for OpenNebula that addressed these issues. The driver is similar to the KVM used in FermiCloud but includes the management of SR-IOV Infiniband devices [14]. However, FermiCloud has some particularities that required modifying it before replacing it for the old VMM.

The driver required changing some configuration and permission options (like running the libvirt API as root) that we wanted to avoid, mainly to handle the assignment of VFs to a virtual machine. Secondly, OpenNebula deployment in FermiCloud is using a customized base directory while the new VMM assumes the OpenNebula standard directory.

Finally, the driver was developed for OpenNebula 4.0 and currently FermiCloud uses the 3.2 version. Although it is backwards compatible there are some differences such as the contextualization scripts that are used to start the machines. To use the driver we had to change completely the way the machines are contextualized in FermiCloud and update the old scripts to new version, while maintaining all the configuration options and customization of FermiCloud.

The driver required changing the physical machines configuration to use the new VMM. It also required the creation of a new virtual network on OpenNebula with the IPs that are going to be assigned to the Infiniband interfaces. This network uses a predetermined bridge and so, when a VM is launched in OpenNebula that includes this network, the driver is aware that one of the VFs available in the physical machine has to be assigned to that virtual machine. The IP of that network is passed from OpenNebula to the VM by creating an Ethernet dummy interface (turned off and not used) on the VM that has codifies the IP as part of a fake MAC Address. With this information, after launching the machine, the contextualization scripts recognize the dummy interface and translate this fake MAC into the IP that is used to create and start the Infiniband's network interface.

Finally, to be able to create VMs in OpenNebula with this new functionality, virtual machine images also need changes so that virtual machines can be launched with proper configurations to become part of IB cluster. These include software installation, network configurations, and ssh server configurations.

During the deployment of the new VMM driver, one challenge was to simplify the procedure and minimize necessary changes so that the new system can be integrated with the rest of the system. A lot of parallel work was done too related with improving some aspects of FermiCloud and adapting them. This is very visible in the process that OpenNebula uses to deploy machines where changes were made that involved the entire deployment process. Once we understand the basic technical details of the new VMM driver and necessary changes and modifications to FermiCloud deployment, we could optimize and automate the process with the Puppet environment in FermiCloud.

E. Limitations

SR-IOV has some limitations worth discussing. The VFs must have the same configuration as the PF and cannot be customized and so they cannot change the configuration of the physical device. The specification, however, gives some freedom to the implementation to manage the communication in the device. There are some devices, for example, that have VF switching that allows the VFs that belong to the same device to talk between themselves without the need of a physical switch connected to that device, making the communication between them faster. Another limitation is the potential interference between VFs under concurrent data transfers, which might affect the individual data rate due to sharing significant parts of the device's hardware. Both characteristics are observed in the following evaluation.

III. PERFORMANCE EVALUATION

We evaluated the performance of the hardware virtualization on up to 56 virtual machines connected by up to 8 DDR Infiniband network links, with micro-benchmarks (latency and bandwidth) as well as with a MPI-intensive application (the HPL Linpack benchmark).

A. Testbed

The **Infiniband card** used in this test is the MHRH19B-XTR ConnectX-2 running with firmware version ConnectX2-rel-2_9_1000. This model has a single port with QSFP (Quad Small Form-factor Pluggable) and connects to the machine using PCI Express 2.0 8x. It uses 4x (4 Infiniband links) with a DDR data rate for a total theoretical speed of up to 20 Gb/s and after the 8b/10b codification 16 Gb/s. It has 1 μs latency when used with MPI. It has Virtual Port Interconnect that gives the possibility of using it for Infiniband or Ethernet. This model has 8 virtual lanes that can create 1 physical function and 7 virtual functions via SR-IOV.

The servers are Koi Computers with 2 quad core Intel E5640 Westmere processors, 48Gb of RAM and 600Gb of SAS Disk, 12TB of SATA, an 8 port RAID Controller, 2 1Gb Ethernet networks and Brocade FiberChannel HBA besides the already mentioned HCA.

Our IB HCA cards are interconnected via a 24 port Mellanox InfiniScale III DDR **switch** that can take up to 24 20Gb/s DDR 4x connections with a total capacity of 960Gb/s.

All of the above resources are part of **FermiLab**'s computing resources spread among 2 buildings in its facilities and that also include the much bigger FermiGrid. Together they process all of the data of the experiments performed on the Department of Energy's Lab (as well as its collaborations), focusing on High-Energy Physics.

B. InfiniBand Network Level Evaluations

The first test is a simple ping-pong benchmark that allowed us to measure the bandwidth and latency of the communication between VMs and between native hosts. The benchmark used was the OFED Perftest package [24] that has the advantage of allowing a lot of customization like changing message sizes, type of connection (Infiniband provides 2 different transmission modes, datagram and connected mode and in the latter is gives the option of reliable or unreliable transmission), size of buffers or functionalities (send, remote read and write from memory).

Figure 1 shows a send operation between 2 host machines. As we can see there is not a significant difference between measured throughput and theoretical maximum of 16 Gb/s. It is possible to observe in Figure 1 the linear evolution of both

the latency and bandwidth respect of the size of the sent message, as one would expect.

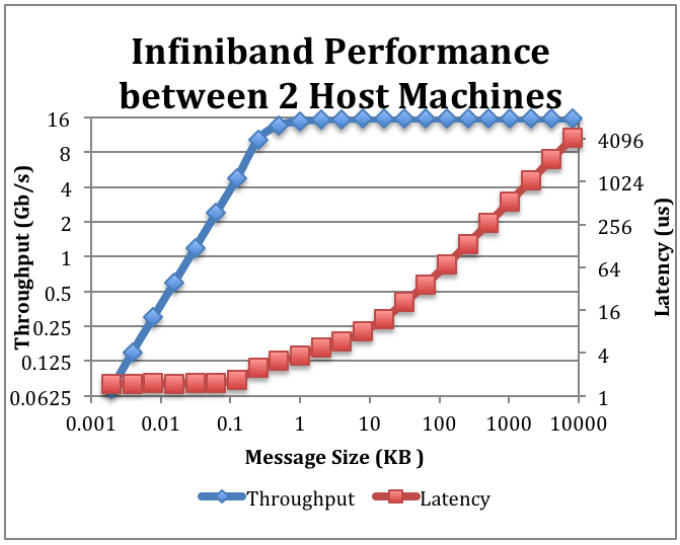

Figure 1. Infiniband Performance between 2 Host Machines

In Figures 2 and 3, we represent the results of the same ping-pong tests in the host machines varying options in the tool such as the size of the receiving and transmitting buffers, the transmission mode or the type of transmission:

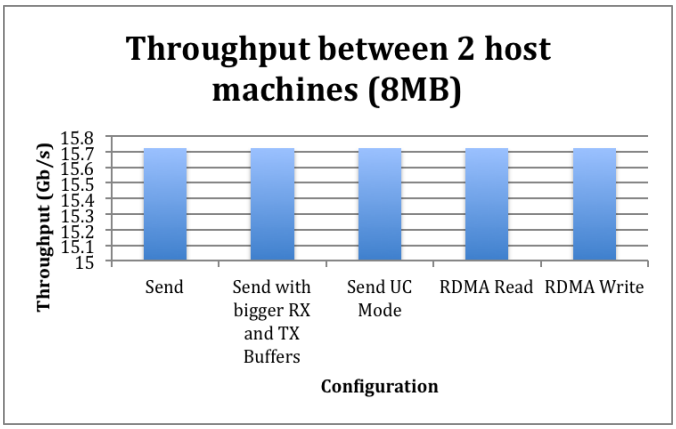

Figure 2. Infiniband throughput between 2 hosts under different configurations and modes

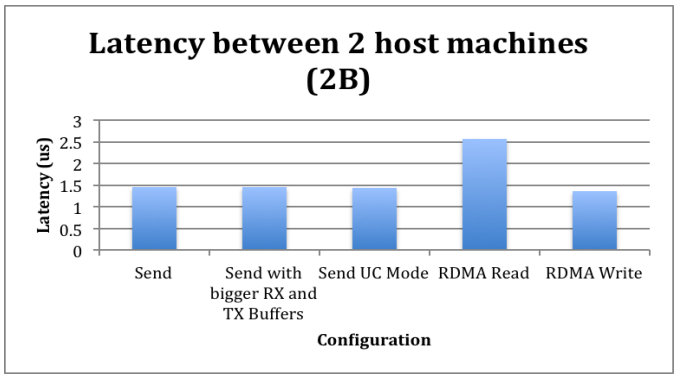

Figure 3. Infiniband latency between 2 hosts under different configurations and modes

It is also interesting to observe that there is very little difference between reliable and unreliable mode (likely due to the reliable network connection that resulted in 0 packet loss) or with different buffer sizes; furthermore, the final measured throughput is very close to the theoretical maximum, 16Gb/s certifying the efficiency of the protocol stack and the deployment. The only observable difference is in the remote read from memory that has a slightly larger latency since the read operation has to fetch some user data from the receiver side main memory before start reading, introducing an overhead that in small messages represents an important percentage [23].

Figure 4 represents virtualization efficiency calculated from the ratio of bandwidth and latency measurements of IB communication between two VMs in different hosts and separate measurements of direct IB channel between two hosts.

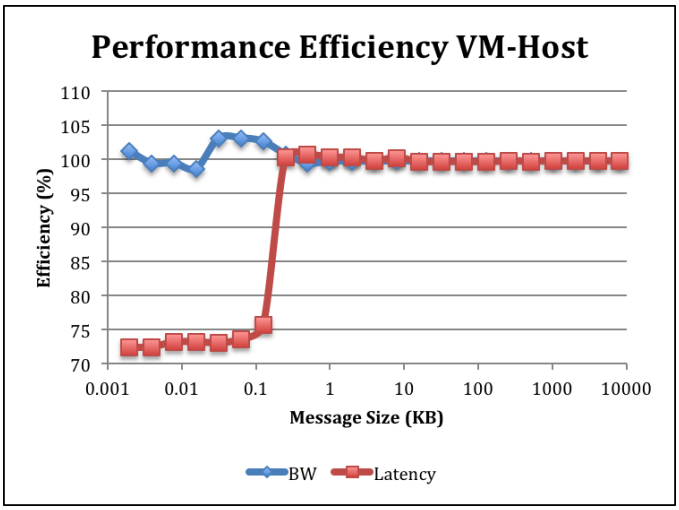

Figure 4. Performance efficiency between 2 VMs and 2 hosts

The bandwidth efficiency is always around 100% and the latency efficiency is also close to that value (sometimes even slightly above that value, since the tests have some noise) for messages bigger than that 128B. In fact, there is a jump in latency between 128B and 256B in the host machines that does not exist in the VMs as is shown in Table 1.

TABLE I. LATENCY IN HOST MACHINES AND VMS BEFORE AND AFTER BEFORE THE LATENCY JUMP IN THE FORMER

Size (B)	*Hosts (us)*	*VMs (us)*	*Efficiency (%)*
128	1.69	2.235	75.61%
256	2.475	2.47	100.2%

This jump in the host machines is explained by how the card's Connect X architecture packages messages under 256B inside the doorbell used to warn the receiver end that the user wants to send a message (in the Infiniband architecture the receiver must be warned before receiving data to create the corresponding receiving queue) [19,23]. In virtual functions, this optimization does not exist (the so called max inline data is 0) what results in the higher latency for small messages [4].

Essentially using 256B messages will guarantee that latency performance is close to the native performance speed.

It is also interesting to see how these results change completely if the 2 VMs are in the same host like it is shown on Figure 5:

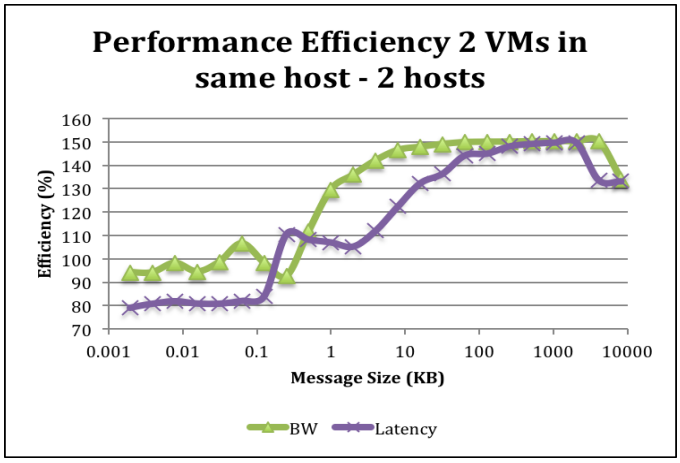

Figure 5. Performance efficiency between 2 VMs in the same host and 2 hosts.

Again, because we are using VFs, the just mentioned optimization for small messages not available making the latency for small messages larger in the VMs, but both throughput and latency can have up to 150% efficiency and reduction respectively compared to the hosts. Although it may seem counterintuitive to have a efficiency higher than 100%, the performance in the virtualized environment is actually better than in the host machines because, as was explained before, provide of a characteristic called VF switching that optimizes the communication between VFs belonging to the same device so that the communication does not need to use the interconnect.

C. MPI Performance on MicroBenchmarks

After these ping-pong tests between 2 hosts or 2 VMs, the goal is to scale the tests up to see if the performance that we observe in the ping-pong tests is maintained at larger scales. For that, we used the Ohio State University Benchmarks [4], in particular the tests osu_multi_lat and osu_mbw_mr to measure latency and bandwidth respectively among several nodes. We show the efficiency (defined as before) depending on the message size for different cluster configurations (1, 2, 4 and 7 VMs per host) in Figures 6, 7 and 8 for a 2, 4 and 8 host machine cluster respectively.

These graphs have some interesting characteristics. First of all, the behavior is similar in 2, 4 and 8 hosts which indicates that the latency efficiency is scalable with increased number of hosts.. However, there is a slight drop in performance when increasing the number of VFs per host (the efficiency having 7 VFs in use simultaneously in one machine is around 80% of the efficiency if it there is only 1 VF). That drop suddenly increases in big messages when the message size is over the MTU and so the message has to be divided in several packets. If we are working on the host machines or have only 1 VM per host that means having one more package in the queue, however if we have 7 VMs per host, it means 7 more packages to send. The difference between packages in the queue is worse when we increase the number of packets per message. Besides, when the message reaches this size, even in host machines, the latency increase is much faster (Figure 1). That increase rate is even bigger if we have more than 1 VM per machine since there are much more packets and the HCA routing to each VF is more complex. Except for this drop in performance in intensive communication, the curve is similar to the tests with the initial performance tests between 2 native hosts with the efficiency improving significantly after 128B.

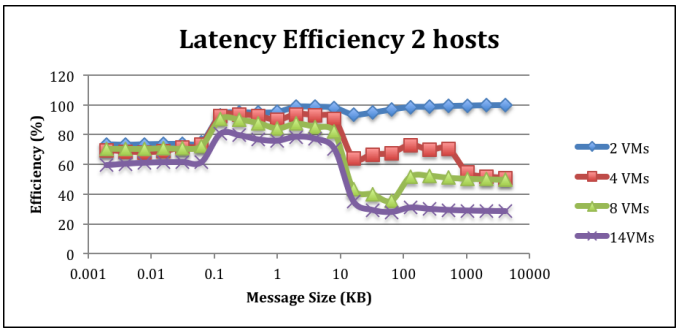

Figure 6. Latency efficiency between 2 hosts and 1,2,4 and 7 VMs/host

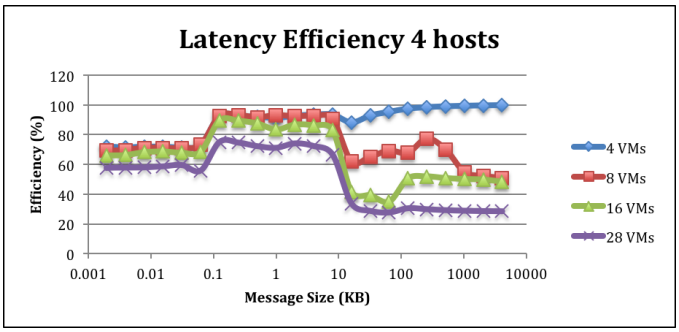

Figure 7. Latency efficiency between 4 hosts and 1,2,4 and 7 VMs/host

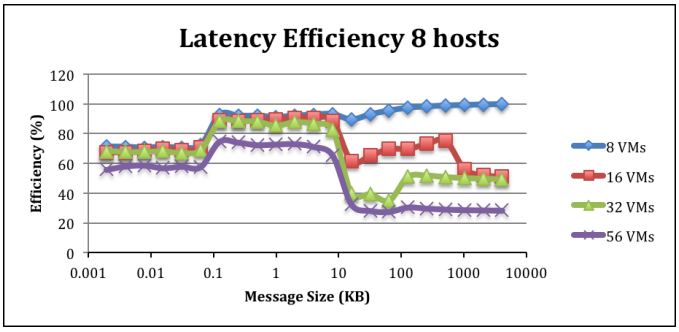

Figure 8. Latency efficiency between 8 hosts and 1,2,4 and 7 VMs/host

In Figures 9, 10 and 11 we can see the same graphs as before but for throughput. Regarding the bandwidth efficiency, the results do not show dependency on the number of hosts, but we see dependency on the number of VMs per host. In this case, having more machines means a throughput speedup, where an increase of 2x in the number of machines can mean an increase of 2x in speed. As before, here the

efficiency is above 100% for small messages. This is mainly because, for small messages, the latency is much bigger than the actual time of transmission and so it is possible to handle the transmission of several messages at the same time. Besides, if there is more than one VM per machine, part of the communication is between VMs in the same machine that, as it was discussed before, is much faster because of VF Switching. However, for bigger message sizes, as we saturate the network, the average bandwidth is the same in the host and the VMs regardless of the number of VMs per host.

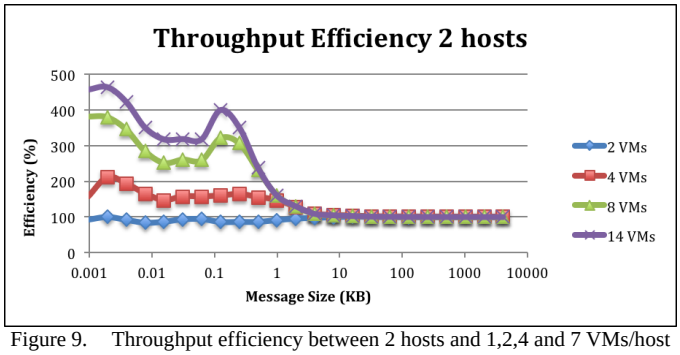

Figure 9. Throughput efficiency between 2 hosts and 1,2,4 and 7 VMs/host

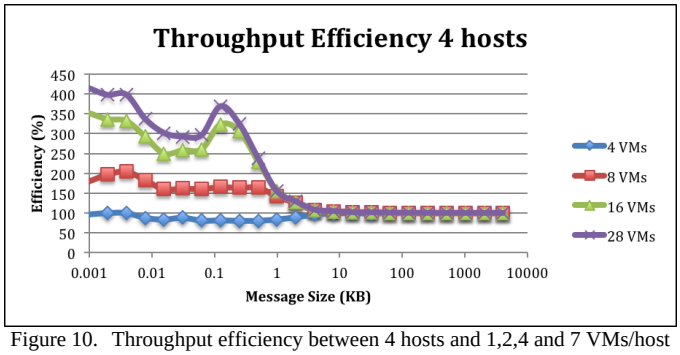

Figure 10. Throughput efficiency between 4 hosts and 1,2,4 and 7 VMs/host

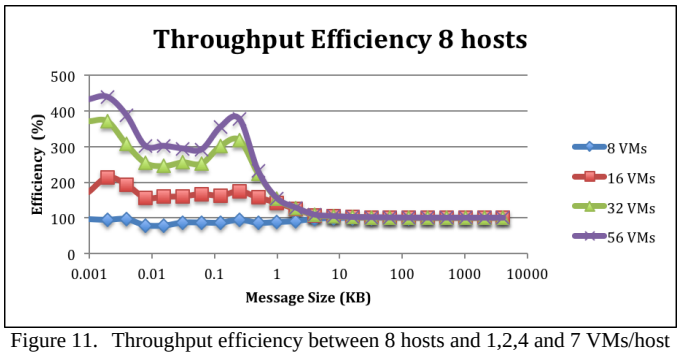

Figure 11. Throughput efficiency between 8 hosts and 1,2,4 and 7 VMs/host

D. Real Application (LINPACK)

After testing the performance of the network, it is interesting to see how a real application behaves. We used the HPL implementation of the Linpack benchmark over OpenMPI. For the tuning, we wanted to test the worst-case scenario and so we used a small block size (32). This small block size also made the application more communication-intensive. This small block size also made the application more communication-intensive, which allowed us to understand the network performance better.

Each one of our hosts had 16 hardware threads that were split between the VMs of that host. If we had 4 VMs, for example, each one would have 4 dedicated HW threads. In the case of 7 VMs, 2 of the VMs had 3 hardware threads and the other 5 had 2. When assigning memory, we followed the same rule. The total number of processes of each execution depended of the number of hosts and not of the number of VMs per host since the VMs split the resources available in its host. The problem size should depend on the available memory so the approach is the same. To assign its value, we searched the optimal value on 2 hosts and multiplied it by √2 every time we doubled the number of hosts. Since Linpack is a squared matrix multiplication and the problem size is the dimension of the matrix, a matrix with √2 more rows has twice more elements (and so there it uses approximately the double of memory).

In Figures 12, 13 and 14, we can see the efficiency, again defined as the percentage of the result of running the benchmark with VMs (1, 2, 4 and 7 per host) and the result of running the same test only on the corresponding hosts (2, 4 and 8). However, the metric analyzed is the output of HPL, which is the metric of how fast a given cluster can conduct numerical operations per second. Previous sections present simple metrics, bandwidth and latency.

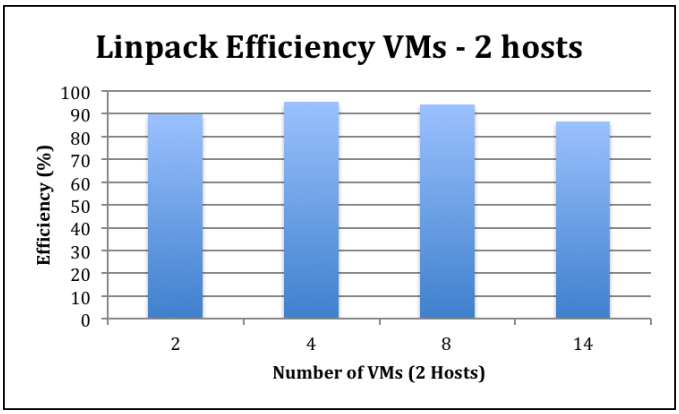

Figure 12. Linpack efficiency between 2 hosts and 1,2,4 and 7 VMs/host

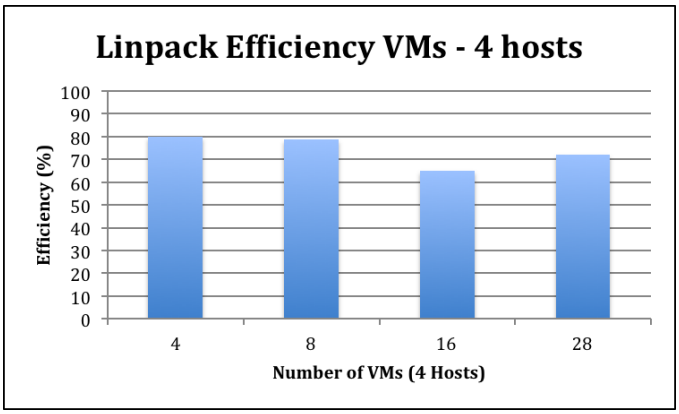

Figure 13. Linpack efficiency between 4 hosts and 1,2,4 and 7 VMs/host

There is a difference when increasing the number of nodes. With 8 nodes, the efficiencies are around 70%, far from the efficiency measurements around 90 % that we observe with two hosts. Part of the drop in efficiency comes from the CPU virtualization efficiency. In fact, when doing the test in 1 host, without communication, the efficiency was about 90% too. However, we believe that the main drop in efficiency comes from the latency overhead that VFs introduce in small messages discussed earlier and that does not scale as we increase the network traffic.

The CPU usage of all the machines was at 100% throughout the entire duration of the tests with less than 28 VMs when we believed the network saturated (More VMs means more communication). In fact, there is a drop in efficiency in the tests with 28, 32 and 56 VMs. Regarding the number of VMs in each host, it seems that 2 and 4 VMs give the best results. The worse configuration is 7 VMs per machine because it is more communication and also because in this case not all the VMs have the same memory and CPU.

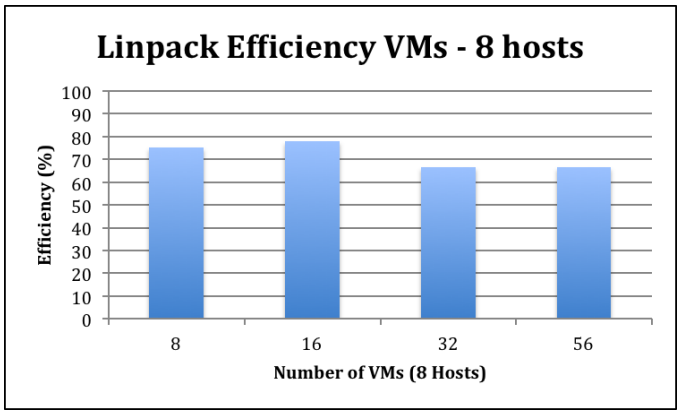

Figure 14. Linpack efficiency between 8 hosts and 1,2,4 and 7 VMs/host

IV. Related Work

Cloud Computing has increased its popularity in recent years. This is due to big improvements in the virtualization techniques [12] as well as a very comfortable model that maximizes resource usage (by sharing the resources among several Virtual Machines) and a provisioning scheme based on an on-demand delivery through the network that is comfortable, very customizable, elastic, scalable and immediate with very little interaction with the service provider [1, 11, 12]. Although there is clearly a shift, its adoption is slow in the HPC domain that is still sensitive to the overheads inherently introduced by the virtualization process [11].

Several techniques are helpful in overcoming or minimizing these overheads (for example, dedicated CPUs) but the big bottleneck is still the I/O Virtualization [18], especially high-speed network devices [4]. This is especially concerning on HPC applications, often distributed and where fast communication between nodes is fundamental.

Infiniband is an especially attractive interconnect to virtualize because of its very low latency, high bandwidth, reliable transmission, remote DMA capabilities, among other features [19]. It is also widely adopted among the Top500 Supercomputers [4]. However, it presents some problems with virtualization mainly because of its architecture [19]. There is part of the connection information stored in the hardware and the OS or the software only use handles to access the device, making much more difficult the actions where the device is changed, like a VM live migration. Besides, the application can talk directly with the device complicating the task of updating application information if there is a change. Finally, Infiniband doesn't use MAC addresses or IPs. The port addresses are referred by LIDs controlled by an external subnet manager that may be in other machine and cannot be changed [1].

There have been some attempts to perform software virtualization but with very poor results. Virtio, a Linux standard to virtualize I/O devices, can be used to paravirtualize Infiniband, with results under one order of magnitude worst [21]. There have been attempts to improve this standard like Virt-IB, described on [1], but they are still in an early phase and only get a performance efficiency of 50% in ping-pong throughput and latency tests.

The solution to all the previous problems seems to be SR-IOV, a low-level virtualization available in a big percentage of the Infiniband cards nowadays. Because it is handled at a lower level, mostly by the device, the virtualization problems discussed before are mostly handled in the device [6]. Besides it provides very good performance results, not only on Ethernet cards [6,8] but also on Infiniband networks, both in Xen [2,10] and in KVM [4] environments (early results) getting the same bandwidth in a VM of the native Infiniband connection as well as a small latency overhead.

We perform simple tests called ping-pong tests [4] with similar results than [4]. Nevertheless, we scaled up to more than 2 nodes getting the results from the previous section, a perfect throughput equal or even better than the native one and a latency efficiency of more than 70% in small messages that goes up to 100% in mid size messages. We also tried real scientific benchmarks. In particular we used HPL, a Linpack implementation available online following recommendations of clustering testing [7,8,9] and getting efficiencies compared with a non-virtualized environment of always over 70%.

V. Conlcusions and Future Work

In this paper, we analyzed the worse case scenario choosing a benchmark configuration that sent small messages that, as was seen earlier, have smaller latency efficiency. An interesting study would be to analyze in depth how the network virtualization reacts to applications that send larger messages as well as other scientific applications and with other MPI implementations.

The studies here were done on only 8 host machines, although FermiCloud has 23 servers available, for now only 8 were in the same building and connected in an Infiniband network. Furthermore, the Infiniband cards only allow 7 VFs per host. However, the new Mellanox cards allow up to 63 VFs per card so it would be interesting to scale up and repeat the tests with more hosts and VMs per host, not possible with the current hardware in Fermilab. However, based on our

results so far with 7 VMs per node, we are not optimistic that the efficiency results will be great at 63 VMs per node scale.

SR-IOV is still in its infancy, it is poorly documented, and it has poor support from the manufacturers. For example, even the drivers that support it are fairly recent. Nevertheless, this virtualization technology has good results, and can deliver almost 100% of efficiency in bandwidth and in latency benchmarks. Small messages are the exception because of limitations of the VFs, related with the process used to pack small messages, and not necessarily with the virtualization itself.

With 8 hosts and 56 VMs, the Linpack efficiency we measured was around 70%, a value close to the latency overhead for small messages and so a value that may be a lower bound. This value, the worst we had, using the worst case possible (small messages), is still much better than all the software virtualizations analyzed for this work and more than one order of magnitude better than virtio, the Linux standard for virtualizing I/O devices.

Virtualization has many advantages, such as isolation, protection, adaptation, customization, and flexibility. Virtual machines deliver management control such as elasticity, scalability, better utilization of resources and dynamic provisioning – all essential advantages brought on by Cloud Computing platforms. These results are definitely promising and open the door for further research that would improve them and lead to advances in the technologies used while adding to the debate of running scientific applications in the Cloud.

ACKNOWLEDGMENTS

This work is supported by the US Department of Energy under contract number DE-AC02-07CH11359 and by KISTI under a joint Cooperative Research and Development Agreement. CRADA-FRA 2013-0001/ KISTI-C13013.

REFERENCES

[1] Ma, Y. M., Lee, C. R., & Chung, Y. C. (2012, December). InfiniBand virtualization on KVM. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on* (pp. 777-781). IEEE.

[2] Yang, C. T., & Ou, W. S. Construction of a Virtual Cluster by Integrating PCI Pass-Through for GPU and InfiniBand Virtualization in Cloud.

[3] Regola, N., & Ducom, J. C. (2010, November). Recommendations for virtualization technologies in high performance computing. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on* (pp. 409-416). IEEE.

[4] Jose, J., Li, M., Lu, X., Kandalla, K. C., Arnold, M. D., & Panda, D. K. D. SR-IOV Support for Virtualization on InfiniBand Clusters: Early Experience.

[5] Mitarbeiter, B., & Stoess, I. J. Towards Virtual InfiniBand Clusters with Network and Performance Isolation.

[6] Dong, Y., Yang, X., Li, J., Liao, G., Tian, K., & Guan, H. (2012). High performance network virtualization with SR-IOV. *Journal of Parallel and Distributed Computing*, *72*(11), 1471-1480.

[7] Dongarra, J. J., Luszczek, P., & Petitet, A. (2003). The LINPACK benchmark: past, present and future. *Concurrency and Computation: practice and experience*, *15*(9), 803-820.

[8] Liu, J. (2010, April). Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on* (pp. 1-12). IEEE.

[9] Ramakrishnan, L., Canon, R. S., Muriki, K., Sakrejda, I., & Wright, N. J. (2012). Evaluating Interconnect and Virtualization Performance forHigh Performance Computing. *ACM SIGMETRICS Performance Evaluation Review*,*40*(2), 55-60.

[10] Goldenberg, D. (2006). Infiniband device virtualization in xen. *Xen summit, January*, *19*.

[11] Huang, W., Liu, J., Abali, B., & Panda, D. K. (2006, June). A case for high performance computing with virtual machines. In *Proceedings of the 20th annual international conference on Supercomputing* (pp. 125-134). ACM.

[12] Hwang, Kai, Geoffrey C. Fox, and J. J. Dongarra (2012). *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Amsterdam: Morgan Kaufmann. Print.

[13] Petitet, A. (2004). HPL-a portable implementation of the high-performance Linpack benchmark for distributed-memory computers. *http://www. netlib-. org/-benchmark/hpl/*.

[14] Macleod, D. (2013). OpenNebula KVM SR-IOV driver.

[15] Krause, M., & Recio, R. (2006). I/O Virtualization And Sharing. *Microsoft Corporation*, 1-26.

[16] Burke, T., & Hat, R. (2010). Red Hat Enterprise Linux6 Roadmap.

[17] Mellanox Technologies (2013). Mellanox OFED for Linux User Manual Rev 2.0

[18] Shafer, J. (2010) "I/O Virtualization Bottlenecks in Cloud Computing Today". In Proceedings of the 2nd Conference on I/O Virtualization (WIOV'10).

[19] Grun, P. (2010). Introduction to infiniband for end users. *White paper, InfiniBand Trade Association*.

[20] M. T. Jones. Virtio: An I/O virtualization framework for Linux. http://www.ibm.com/developerworks/linux/library/l-virtio/

[21] Motika, G., & Weiss, S. (2012). Virtio network paravirtualization driver: Implementation and performance of a de-facto standard. *Computer Standards & Interfaces*, *34*(1), 36-47.

[22] Intel LAN Access Division (2010). PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology, *White paper*

[23] Sur, S., Koop, M. J., Chai, L., & Panda, D. K. (2007, August). Performance analysis and evaluation of Mellanox ConnectX InfiniBand architecture with multi-core platforms. In *High-Performance Interconnects, 2007. HOTI 2007. 15th Annual IEEE Symposium on* (pp. 125-134). IEEE.

[24] Open Fabrics Enterprise Distribution, http://www.openfabrics.org/

[25] IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures, R. Moreno-Vozmediano, R. S. Montero, I. M. Llorente. IEEE Computer, vol. 45, pp. 65-72, Dec. 2012.

[26] Sadooghi, Iman, and Raicu, Ioan. "Towards Scalable and Efficient Scientific Cloud Computing", Doctoral Showcase, IEEE/ACM Supercomputing/SC 2012