Exploring Infiniband Hardware Virtualization in OpenNebula towards Efficient High-Performance Computing

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) application 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 Infrastructure-as-a-Service (IaaS) cloud. To avoid software virtualization towards minimizing the virtualization overhead, we employed a technique called Single Root Input/Output Virtualization (SRIOV). 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).**

Infiniband; virtualization; Cloud Computing; OpenNebula; SRIOV; 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. It also provides additional resources to the Laboratory´s Grid by providing Virtual Machines that run on the Cloud Infrastructure and increase its computing capacity. Nevertheless, the usage of VMs implies managing the virtualization and sharing of the physical hardware (HW) and resources, introducing performance overheads.

Cloud Computing has increased its popularity in recent years. This is due to big improvements in the virtualization techniques [10] 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 [9, 18]. Several techniques are helpful in overcoming or minimizing these overheads (for example, dedicated CPUs) but the big bottleneck is still the I/O Virtualization [13], especially high-speed network devices [4]. This is especially concerning on HPC applications, where a fast and reliable network is fundamental since they are often distributed and usually use the Message Passing Interface Protocol (MPI), very sensitive to changes in latency, such as the virtualization overhead. [10]

The FermiCloud team is interested on expanding its functionality to provide a useful platform to the scientific investigation by being able to run scientific applications and models. However, scientific computing relies on compute and data-intensive jobs that have to be coordinated among several nodes. Infiniband is an especially interesting technology for that since it is one of the interconnect links offering one of the highest throughputs and a particularly low latency, guaranteeing Quality of Service (QoS) and scalability. It uses its own protocol stack that allows the communication with the Host Channel Adapter (HCA) in each processor that can even exchange messages directly with the application, bypassing the Operating System. It is often used in high performance computing [14]. Several implementations of the protocol stack are offered as well as of the Subnet Manager and Administrator, which provides and maintains routing tables on the IB hosts since most switches are dummy and cannot handle the routing.

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 fast technology like Infiniband. To overcome the virtualization overhead we used a standard called SRIOV developed by the PCI Special Interest Group for Virtualized Servers and that achieves device virtualization without using device emulation by enabling a device to be shared by multiple virtual machines. With SRIOV, a PCIe device can export multiple virtual functions (VFs) subordinated to a physical function or PF (a full-featured PCIe function). These virtual functions cannot be configured and share the configuration of its physical function. They reside in the device itself sharing its resources but appear as an independent PCIe function. This model allows the hypervisor to simply map virtual functions to a specific virtual machine, which can achieve the native device performance even without using pass through and the increased CPU workload that it creates [4, 6]. The main limitations are a possible individual data rate reduction in concurrent transfers (VFs share HW resources of the device) and the inability of configuring individual VFs.

This work explores the potential use of Infiniband hardware virtualization in an OpenNebula cloud [17] towards the efficient support of MPI-based workloads. It describes 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 SRIOV 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 were performed to test the efficiency of this virtualization (SRIOV), defined as the ratio between a certain metric in a virtualized environment and in a non-virtualized one with the same resources. The tests included micro-benchmarks to measure latency and bandwidth and a resource intensive application (HPL Linpack). ***The results were excellent, with VMs reaching the same throughput and the same latency (for larger message sizes) of the bare-metal testbed.***

The rest of the paper is organized as follows. Section 2 describes how the deployment was done. Section 3 focuses on the performance evaluation, describing the battery of tests and its results. Section 4 describes published work related with this topic like SRIOV 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

A. SRIOV 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 to enable the functionality and specify some parameters like the number of PFs and VFs to be shown to the machine. The maximum number of VFs supported by the card used is 7 and it only has 1 PF (is a single port card). 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. It is worth mentioning that the host machines do not even need the entire Infiniband support software, only the kernel modules that manage the communication with the card and a Subnet Manager to handle the routing (in one of the hosts).

B. SRIOV and OpenNebula

OpenNebula and KVM (the core virtualization infrastructure that it uses) need to be changed because SRIOV needs the support of the hypervisor since it has to be aware that the VFs are not real devices. Besides, it must manage the automation and integration of the assignment of different VFs to each Virtual Machine: The configuration of its network and 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 SRIOV Infiniband devices [11]. 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, mainly to handle the assignment of VFs to a virtual machine, and needed to be adapted to the customized base directory in the OpenNebula deployment in FermiCloud.

This driver was developed for OpenNebula 4.0 and currently FermiCloud uses version 3.2. To use it, some changes in OpenNebula were made to upgrade some of the functionality to the version 4.0. A good example is the contextualization scripts that are used to start the VMs and that had to be completely changed to upgrade them to the new version. Finally, a virtual network was created and driver configuration files populated so that it could recognize VFs, assign them and handle them during VM migration. New VM images were also deployed with the new software installation, network and SSH server configurations and the OpenNebula functionality improvements.

The main design concern during this project was to simplify the procedure and minimize necessary changes so that the new functionality can be integrated with the rest of the system and in automation software like Puppet. For that, the project do not changes the already deployed design of FermiCloud. Also it tries to maximize the usage of packages provided by the Linux Distribution that were already tested and are simpler to deploy, since no compiling is needed and there are no incompatibilities with the kernel or its modules. This was the biggest lesson learnt throughout the project after experiencing a lot of difficulties. Finally, a lot of parallel work was done related with improving some aspects of FermiCloud and adapting them to this new functionality.

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 (after the 8b/10b codification, 16 Gb/s). It has 1 µs latency when used with MPI. This model has 8 virtual lanes that can create 1 physical function and 7 virtual functions via SRIOV.

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.

To manage the virtualization we used the Linux's Kernel-based Virtual Machine (KVM), a virtualization solution, and libvirt, the virtualization API that OpenNebula[17] uses and that lets us create, personalize, destroy and manage VMs. The MPI distribution used was OpenMPI.

B. *InfiniBand Network Level Evaluations*

The first test is a simple Ping-Pong benchmark that measures the bandwidth and latency of the communication between VMs and between native hosts. The benchmark used was the OFED Perftest package. 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.

Testing different operations and in different modes provided similar results. 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 [16].

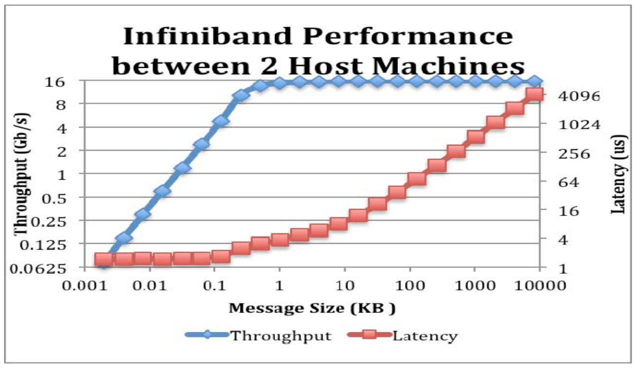

Figure 1. Infiniband Performance between 2 Host Machines

Figure 2 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 a direct IB channel between two hosts. The bandwidth efficiency is always around 100% and the latency efficiency is also close to that value (sometimes even slightly above, 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.

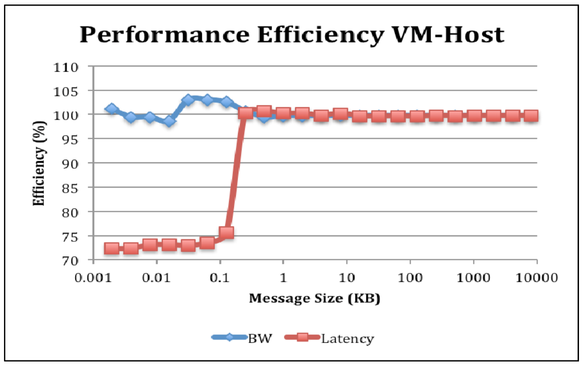

Figure 2. Performance efficiency between 2 VMs and 2 hosts

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 3:

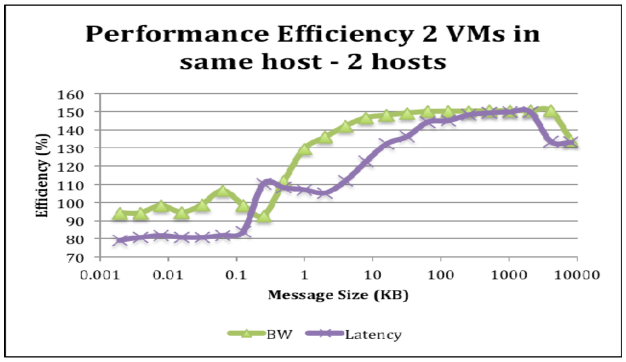

Figure 3. Performance efficiency between 2 VMs in the same host and 2 hosts.

Again, because we are using VFs, the just mentioned optimization for small messages is not available making the latency for small messages higher in the VMs, but both throughput and latency can have up to 150% efficiency 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, the cards provide a characteristic called VF switching that allows the VFs that belong to the same device to talk between themselves without using the interconnect, making the communication between them faster.

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 4 and 5 for a 2 and 8 host machine cluster respectively.

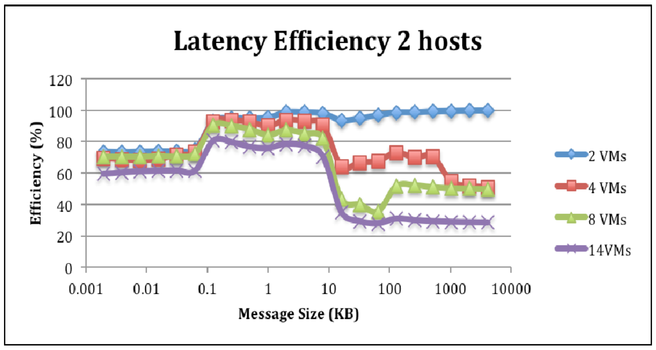

Figure 4. Latency efficiency between 2 hosts and 1,2,4 and 7 VMs/host

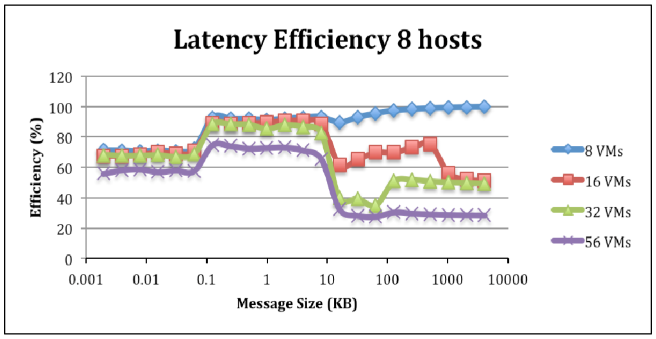

Figure 5. Latency efficiency between 8 hosts and 1,2,4 and 7 VMs/host

These graphs have some interesting characteristics. First of all, the behavior is similar in 2 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) explainable by the increase of traffic in the card of that specific host machine where the VFs share most of the HW. That drop in efficiency suddenly increases in big messages when the message size is over the Maximum Transfer Unit (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 packet in the queue, however if we have 7 VMs per host, it means 7 more packets to send. The difference between packets 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. We believe that this increase in the latency and message size also originates an increase in noise that causes some of the jumps seen in big message sizes in this graphs. 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.

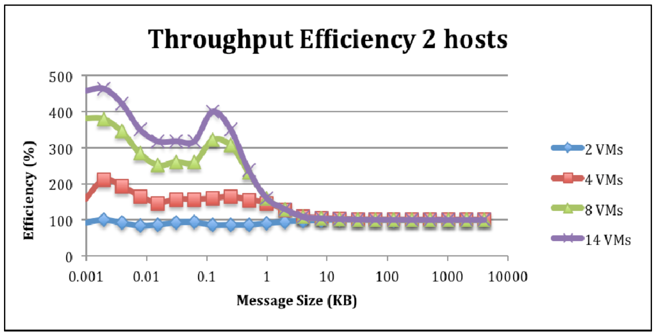

Figure 6. Throughput efficiency between 2 hosts and 1,2,4 and 7 VMs/host

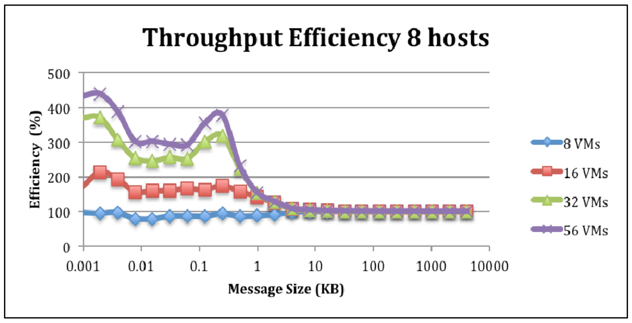

Figure 7. Throughput efficiency between 8 hosts and 1,2,4 and 7 VMs/host

In Figures 6 and 7 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.

D. Resource Intensive Benchmark (LINPACK)

After testing the performance of the network, it is interesting to see how a resource and MPI intensive application behaves. We used the HPL implementation of the Linpack benchmark over OpenMPI. This benchmark is widely used in HPC and can provide a better image of how a real scientific application behaves. 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, which allowed us to understand the network performance better. The benchmark size and parameters scaled in the same proportion as the cluster size.

In Figures 8 and 9, 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 (with the same conditions) only on the corresponding hosts (2 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.

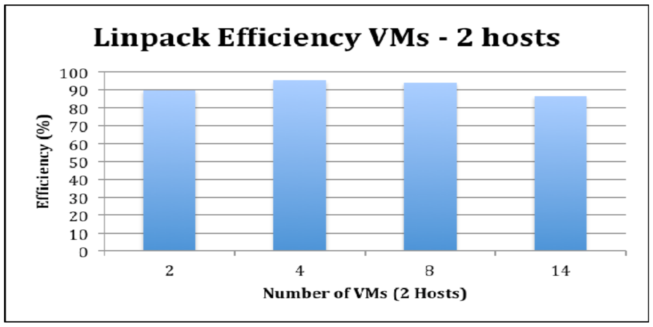

Figure 8. Linpack efficiency between 2 hosts and 1,2,4 and 7 VMs/host

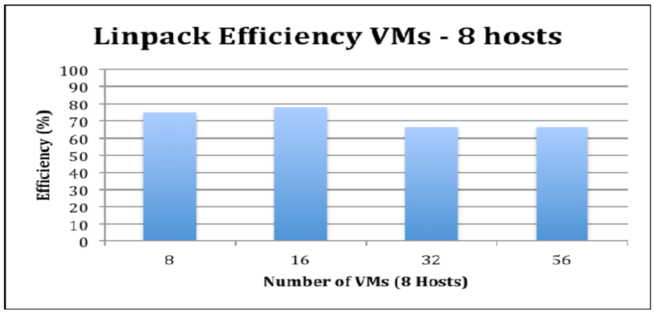

Figure 9. Linpack efficiency between 8 hosts and 1,2,4 and 7 VMs/host

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 2 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.

IV. Related Work

Infiniband is an especially attractive interconnect to virtualize because of its very low latency, high bandwidth, reliable transmission, remote DMA capabilities, among other features [14]. It is also widely adopted among the Top500 Supercomputers [4]. However, it presents some problems with virtualization mainly because of its architecture [14]. 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 of a VM is changed, like a live migration. Besides, the application can talk directly with the device complicating the task of updating information about the connection present in the application if there is a change. Finally, Infiniband does not 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 Infiniband 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 [14]. Other attempts improved 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 (worse when escalating them).

The solution to all the previous problems seems to be SRIOV, 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 getting the same bandwidth in a VM of the native Infiniband connection as well as a small latency overhead. However, no evaluation seems to have been published in a real cloud environment, integrated with the cloud manager, with more than 2 nodes and more complex benchmarks than the simple Ping-Pong test. We perform simple latency and bandwidth tests with similar results than the ones presented by the benchmark developer [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%, an excellent result much better than other software based solutions and that confirms the advantages of this using this approach for network virtualization.

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 a big scientific application and other MPI implementations.

The studies here were done on only 8 host machines since, although FermiCloud has 23 servers available, for now only 8 are in the same building and connected in an Infiniband network. Furthermore, the Infiniband cards only allow 7 VFs per host. 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. However, based on our results so far with 7 VMs per node, we are not optimistic that the efficiency results would be great at 63 VMs per node scale.

SRIOV is still in its infancy, it is poorly documented, and it has poor support from the manufacturers and 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 latency benchmarks. Small messages are the exception, due to limitations of the VFs, related with the process used to pack small messages, and not necessarily due to 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 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. Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on (pp. 777-781). IEEE.

[2] Yang, C. T., & Ou, W. S. (2013) Construction of a Virtual Cluster by Integrating PCI Pass-Through for GPU and InfiniBand Virtualization in Cloud. 14'th International Conference on Parallel and Distributed Computing, Applications and Technologies.

[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. (2013) SRIOV Support for Virtualization on InfiniBand Clusters: Early Experience.2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid),

[5] Mitarbeiter, B., & Stoess, I. J. (2011) Towards Virtual InfiniBand Clusters with Network and Performance Isolation. System Architecture Group, Karlsruhe Institute of Technology (KIT), Germany.

[6] Dong, Y., Yang, X., Li, J., Liao, G., Tian, K., & Guan, H. (2012). High performance network virtualization with SRIOV. 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] 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.

[9] 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.

[10] 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.

[11] Macleod, D. (2013). OpenNebula KVM SRIOV driver. Center for High Performance Computing, Department of Science and Technology, South Africa

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

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

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

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

[16] 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.

[17] 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

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