I/O Device Virtualization in the multi-core era, a QoS perspective

J. Lakshmi, S.K. Nandy
Indian Institute of Science, Bangalore, India
{jlakshmi, nandy }@serc.iisc.ernet.in

Abstract

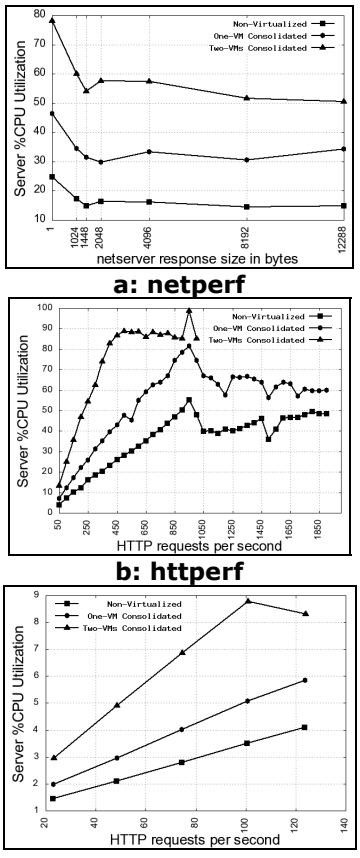
In this paper, we propose an extension to the I/O device architecture, as recommended in the PCI-SIG IOV specification, for virtualizing network I/O devices. The aim is to enable fine-grained controls to a virtual machine on the I/O path of a shared device. The architecture allows native access of I/O devices to virtual machines and provides device level QoS hooks for controlling VM specific device usage. For evaluating the architecture we use layered queuing network (LQN) models. We implement the architecture and evaluate it using simulation techniques, on the LQN model, to demonstrate the benefits. With the architecture, the benefit for network I/O is 60% more than what can be expected on the existing architecture. Also, the proposed architecture improves scalability in terms of the number of virtual machines intending to share the I/O device.

1. Introduction

In multi-core servers there is abundance of processors when compared to the number of I/O devices which leads to many virtual machines (VMs) sharing a single I/O device. Prevalent virtualization architectures and technologies have fine-grained support for processor time-sharing which results in minimal loss of application performance when it is moved from an isolated server to a virtual machine. However, this is not true with respect to the way I/O devices are virtualized. In a non-virtualized system itself, every I/O device access is achieved through a series of data-abstraction and device access layers that add to the access latency. Virtualization incurs an additional layer of indirection to this already overloaded access path[1]. The key to improving resource utilization in a virtualized machine is, to allow maximum possible resource access operations to perform natively with minimal virtual machine monitor (VMM) intervention[2] and for this the I/O devices need be virtualization aware.

To understand the effect of sharing a NIC (network interface card), hereafter called the network I/O device, at near maximum utilization bandwidth, on an application in virtualized environment, we examine the behavior of three benchmarks, *netperf[3]*, *httperf*[4] and *surge[5]* for three different environments, namely non-virtualized, virtualized and consolidated-virtualized server. For the *netperf* benchmark, *netperf* is the name of the client and *netserver* is the server component. For this study, we have chosen the TCP_CRR test of *netperf*. The TCP_CRR test measures the connect-request-response sequence throughput achievable on a server and is similar to the access requests used in http based applications. In the case of *httperf* and *surge* benchmarks, the client, httperf or surge, communicates with a standard *http* server using the *http* protocol. The *httperf* client allows for specifying the workload in terms of the number of *http* requests to the server in one second, for a given period of time, to generate statistics like average number of replies received from the server, average response time of a reply and the network bandwidth utilized for the workload. While *netperf* and *surge* give the achievable or achieved throughput, *httperf* gives an average throughput calculated for a subset of samples within the given experiment and hence gives an optimistic estimate which may fall short of expectation in situations where sustained throughput is a requirement. For this study, all the benchmarks were run on a single core, *linux* server with a total of 2GB RAM and hosting a Broadcom Ethernet card configured to serve a bandwidth of 100Mbps. Each of the experimental value plotted in the following charts is an average of five independent values measured for the benchmark run, to rule out the possibility of result variation. Each experiment was executed for a period of 600 seconds to represent steady state of server behavior. For the non-virtualized server, the machine is booted into a standard FedoraCore6 (FC6) *linux* system. For the virtualized server, the same system is booted into Xen3.0.3[6] with FC6 for the Domain-0 and Domain-1. Domain-0 is the IDD[7] (independent driver domain) hosting the network I/O device and Domain-1 is the virtual machine hosting the benchmark server. In case of the consolidated-virtualized server, two virtual machines, namely Domain-1 and Domain-2, are hosted on the machine along-with Domain-0. In this case, server component of the benchmark is hosted on each of the virtual machine and independent clients make requests to each of the servers. For each of the benchmarks, the comparison is between a non-virtualized server with that of a virtualized server and consolidated-virtualized server. We take two metrics for comparison; the first one is %CPU utilization of the server and the second one reply rate or throughput of

the server. We show three charts, one each for the benchmarks. Figure 1 shows the plot of %CPU utilization of the server against response message size for the *netperf* and request rate for *httperf* and *surge*. In each of the charts we observe that moving from a non-virtualized to a virtualized server, the CPU resource utilization to support the same workload increases. This is the effect of virtualization on application resource requirement. Further, when we look at the case where two-VMs are consolidated on to a server, the server resource utilization increases significantly, as expected. In this case each of the VMs sharing the same NIC, also share the VMM and IDD. When compared to servicing a single VM, the VMM and IDD now have to also service the requests of the second VM, which increases the %CPU utilization on the server. Also, the sharing of VMM and IDD by all device sharing VMs adds to device access latencies which contributes to a reduction in the maximum sustained throughput of the application.



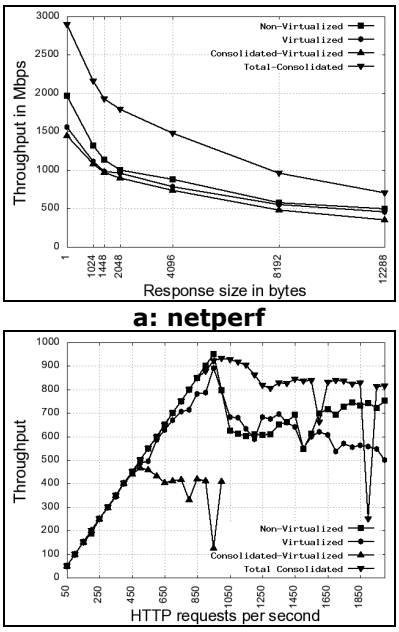
a: netperf

b: httperf

c: surge

Figure 1: Benchmark server CPU resource utilization charts for *netperf, httperf* and *surge* benchmarks. The comparison is for servers hosted on a non-virtualized, virtualized and consolidated-virtualized system.

This is illustrated in Figure 2, which gives details of the throughput that is achieved in the case of *netperf* and *httperf* benchmarks. In the case of the *netperf* benchmark the loss of achievable throughput, for an application moving from non-virtualized to virtualized environment ranges from 4.0% to 20% and 10% to 29% in the case of consolidated-virtualized server. The surprising result is the case of total throughput achievable across both the VMs on a consolidated-virtualized server for *netperf*. We observe that the total throughput achievable is almost double that of what is achievable for a single VM. The reason for this is that the service time required to generate a response to the *netperf* request is long enough ***not*** to allow complete utilization of the network I/O bandwidth. And hence we see a better utilization for the consolidated server wherein two VMs are using the same NIC. This gives a clear motivation for sharing of I/O device in virtualized servers. However, in the case of httperf the improvement is not so dramatic. We observe here that while the reply rate increases linearly with the increase in request rate, till the peak rate, for the non-virtualized server, there is a gradual drop of throughput starting from 2.0% to 12.5% for the virtualized server as we progress from the request rate of 500reqs/s to the peak rate. The sustained total throughput without loss for the consolidated-virtualized server improves to 800reqs/s but still falls short of that achieved for the non-virtualized server by 10%.



a: netperf

b: httperf

Figure 2: *netperf* and *httperf* throughput chart for benchmark server hosted on non-virtualized, virtualized and consolidated-virtualized system.

The other noteworthy point of observation is the behavior of each stream of benchmark in the case of consolidated-virtualized server. We see that in general

there is a further reduction in throughput, when compared to the virtualized environment, in *netperf* and *httperf,* with a marked decrement in the later case. This indicates the obvious; lack of QoS constraints would lead to severe interference in performance offered by the device sharing VMs. We note here that in *linux,* packet level QoS controls are available as a software stack through the *netfilter* module, which is a serious drawback. While the software controls work well for the transmission path, the reception path is virtually uncontrolled at the device level. It may so happen that the device is receiving a large number of packets for a VM that would eventually be dropped and this leads to loss of bandwidth for the other device sharing VMs.

As observed from the behavior of the benchmarks, we identify the following bottlenecks for sharing network I/O device across multiple VMs on a Xen virtualized server.

- Virtualization increases overheads in device utilization which leads to increased CPU utilization of the VMM and IDD hosting the device.
- Virtualization overheads cause loss of device bandwidth utilization from within a VM. Consolidation improves the device bandwidth utilization but further adds to CPU utilization of the VMM and IDD. Also, if the VMM and IDD do not support concurrent device access APIs they become the bottlenecks for the shared device access.
- QoS features for regulating incoming and outgoing traffic are currently implemented in the software stack. Uncontrolled incoming traffic to a VM sharing a network device can severely impact the performance of other VMs because the decision to drop an incoming packet is taken after the device has received the packet.

In this paper we propose an extension to I/O virtualization architecture, as recommended by the PCI-SIG IOV specification[8]. The PCI-SIG IOV specification defines the rudiments for making I/O devices virtualization aware. On the multi-core servers with server consolidation as the goal, particularly in the enterprise segment, being able to support multiple virtual I/O devices on a single physical device is a necessity. Already high speed network devices like 10Gbps NICs are appearing in the market. Pushing such devices to even 80% utilization needs fine-grained resource management at the device level. The basic goal of the proposed architecture is to be able to support finer levels of QoS guarantees, without compromising on device utilization. This proposal has its basis in exokernel's [9] implementation of I/O handling. The architecture is designed to enable native access of I/O devices to virtual machines and provide device level QoS hooks for controlling VM specific device usage. The architecture aims to reduce network I/O device access latency and enable improvement in effective usable bandwidth in virtualized systems by addressing the following issues:

- Separating device management issues from device access issues.
- Allowing native access of a device to a VM by supporting concurrent devices access and eliminating IDD from the path of device access.
- Enable fine-grained resource controls at the device.

The rest of the paper is organized as follows. We describe the need for extending I/O device virtualization architecture by presenting the related work in section 2. In section 3 we highlight the bottlenecks of the device access path in the existing architecture, followed by a detailed description of the proposed architecture to overcome these bottlenecks. We then describe the network packet workflow for the proposed architecture in section 4, which forms the basis for generating the LQN model used in the simulation studies for architecture evaluation. Brief description of the LQN model generation and detailed presentation of simulation results is covered in section 5. Finally, in section 6 we conclude with remarks on the benefits of the architecture.

2. Review of I/O virtualization techniques

Virtualization technologies encompass a variety of mechanisms to decouple the system architecture and the user-perceived behaviour of hardware and software resources. Among the prevalent technologies, there are two basic modes of virtualization, namely, full system virtualization like in VmWare [14] and para-virtualization as in Xen [6]. In full system virtualization complete hardware is replicated virtually. Instruction emulation is used to support multiple architectures. The advantage of this virtualization is that it enables unmodified operating systems (OS) to execute on the VM. Since it adopts instruction emulation, it tends to have high performance overheads. Para-virtualization is more efficient and comes with lower performance overheads. In this kind of virtualization the OS is also modified suitably to run concurrently with other virtual machines on the same hardware. In either case, system virtualization is enabled by a layer called the VMM that now provides the resource management functionality across multiple virtual machines. I/O virtualization started with dedicated I/O devices assigned to a VM and has now evolved to device sharing across multiple VMs through virtualized software interfaces[10]. A dedicated software entity, called the I/O domain is used to perform physical device management. The I/O domain can be part of the VMM or be an independent domain, like the IDD of Xen. In the case of IDD the I/O devices are private to the domain and memory accesses by the devices are restricted to the IDD. Any application in a VM seeking access to the device has to route the request through the IDD and the request has to pass through the address translation barriers of the IDD and

VM [17] - [22]. Recent publications on concurrent direct network access (CDNA)[15] and scalable self-virtualizing network interface are closer to the proposed work. However, the scalable self-virtualizing interface[16] describes assigning a specific core for network I/O processing on the virtual interface and exploits multiple cores on embedded network processors for this. The paper does not detail how the address translation issues are handled, particularly in the case of virtualized environments. The CDNA work is architecturally similar to the proposal in the paper. CDNA relies on per VM Rx/Tx ring buffers to manage VM specific network data. The VMM handles the virtual interrupts. However, it does not talk about the performance interference due to uncontrolled data reception by the device nor does it talk about the need for addressing the QoS constraints at the device level. The proposed architecture in this paper addresses these and also the issue of pushing the basic constructs to assign QoS attributes like required bandwidth and priority into the device to get fine-grained control on interference effects. The proposed architecture has it basis in *exokernel's* philosophy of separating device management from protection. In *exokernel*, the idea was to extend native device access to applications with *exokernel* providing the protection. In our approach, the extension of native device access is to the VM, the protection being managed by the VMM. A VM is assumed to be running the traditional OS. Further, the PCI-SIG community has realized the need for I/O device virtualization and has come out with the IOV specification to deal with it. The IOV specification however, talks about device features to allow native access to virtual device interfaces, through the use of I/O page tables, virtual device identifiers and virtual device specific interrupts. The specification presumes that QoS is a software feature and does not address this. Many implementations adhering to the IOV specification are now being introduced in the market by Intel[23], Neterion[24], NetXen[25], etc. CrossBow[26] suite from SUN Microsystems talks about this kind of resource provisioning, but it is a software stack over a standard IOV complaint hardware. The results published using any of these products are exciting in terms of the performance achieved, but almost all of these have ignored the control of reception at the device level. We believe that lack of such a control on highly utilized devices will either cause performance degradation or lead to under-utilization of the device bandwidth.

3. Extension of I/O virtualization architecture

In the existing Xen virtualization architecture, when we analyze the network packet workflow we observe few bottlenecks which we aim to eliminate in the proposed scheme. The bottlenecks are:

- Since the device is shared, the device memory behaves like a common memory for all the contending VMs accessing the device. One misbehaving VM can ensure deprivation leading to data loss for another VM.
- The IDD is also a bottleneck for all the VMs sharing the device. IDD incurs processing overheads for each VM. Current IDD implementations do not have any hooks for controlling the overheads on a per VM basis. Lack of such controls leads to performance interference in the device sharing VMs.
- Every network packet has to cross the address translation barrier of VMM to IDD to VM and vice-versa. This happens because of lack of separation of device management issues from device access issues. The service overheads of this stage-wise data movement causes drop in effective utilized device bandwidth. In multi-core servers with scarce I/O devices, this would mean having high-bandwidth under-utilized devices or low throughput applications on the consolidated server. To overcome the above listed drawbacks, we propose an extended architecture for virtualizing I/O devices that enables separation of device management issue from device access issue. This is done by building device protection mechanisms into the physical device and managed by the VMM. As an example, for the case of NIC, the VMM should be able to recognize the destination VM of an incoming packet by the interrupt raised by the device and forward it to the appropriate VM. The VM should then be able to process the packet as it would do so in the case of non-virtualized environment. Thus, device access and scheduling of device communication is managed by the VM that is using the device. This eliminates the intermediary VMM/IDD on the device access path and reduces I/O service time which improves the usable device bandwidth.

3.1 Proposed I/O Virtualization Architecture Description:

Figure 3 gives a block schematic of the proposed I/O virtualization architecture. The picture depicts a NIC card that can be housed within a multi-core server. The card has a controller that manages the DMA transfer to and from the device memory. The standard device memory is now replaced by a partitionable memory supported with n sets of device registers. A set of m memory partitions, where $m \bullet n$, with device registers forms the virtual-NICs. Ideally the device memory should be reconfigurable, i.e. dynamically partitionable, and the VM's QoS requirements would drive the sizing of the memory partition. The advantage of having a dynamically partitionable device memory is that any unused memory can be easily extended into or reduced from a vNIC in order to match adaptive QoS specifications. The NIC identifies a vNIC request by generating message signaled interrupts (MSI). The number of interrupts supported by the controller restricts the number of virtual-NICs that can be

exported. Although, the finite number of physical resources on the NIC restricts the number of vNICs that can be exported, judicious use of native and para-virtualized access to the vNICs, based on the QoS guarantees a VM needs to honour, can overcome the limitation. A VM that has to support stringent QoS guarantees can choose to use native access to the vNIC whereas those VMs that are looking for best-effort NIC access can be allowed para-virtualized access to the vNIC. The VMM can aid in setting up the appropriate hosting connections based on the requested QoS requirements.

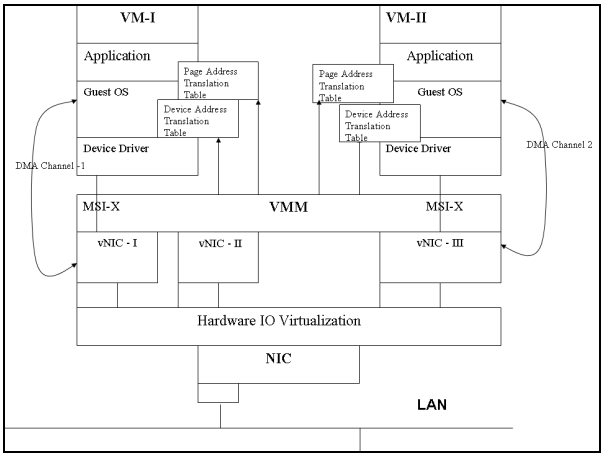


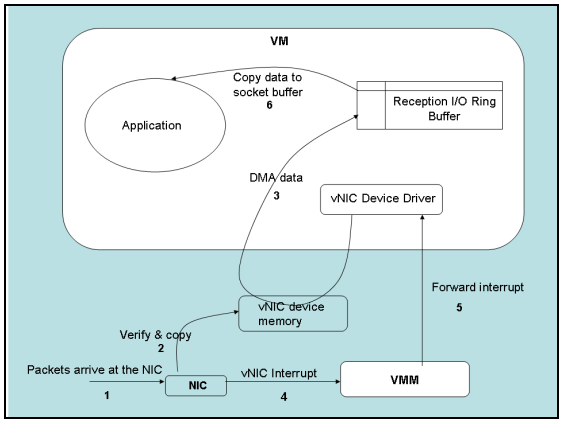
Figure 3: NIC architecture supporting MSI interrupts with partitionable device memory, multiple device register sets and DMA channels enabling independent virtual-NICs.

The architecture can be realized by the following modifications:

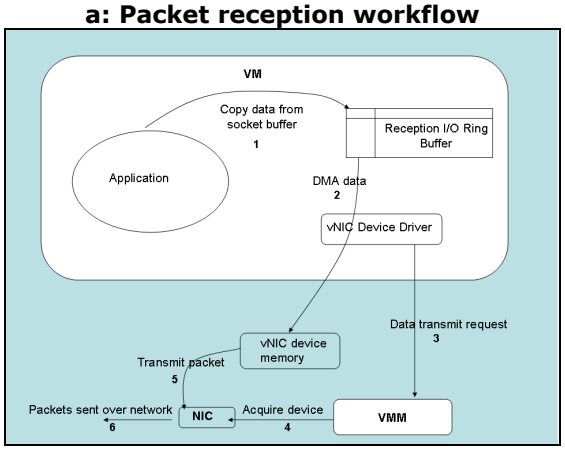
- **Virtual-NIC:** In order to define vNIC, the physical device should support time-sharing in hardware. For a NIC this can be achieved by using MSI and dynamically partitionable device memory. These form the basic construct to define a virtual device on a physical device as depicted in Figure 3. Each virtual device has a specific logical device address, like the MAC address in case of NICs, based on which the MSI is routed. Dedicated DMA channels, a specific set of device registers and a partition of the device memory are part of the virtual device interface which is exported to a VM when it is started. We call this virtual interface as the virtual-NIC which forms a restricted address space on the device for the VM to use and remains in possession of the VM till it is active or relinquishes the device.
- **Accessing virtual-NIC:** For accessing the virtual-NIC the IDD layer for network I/O in Xen is replaced by a VM's native device driver. This device driver can only manipulate the restricted device address space which was exported through the virtual-NIC interface by the VMM. With the virtual-NIC, the VMM only identifies and forwards the device interrupt to the destination VM. The OS of the VM now handles the I/O access and thus can be accounted for the resource usage it incurs. This eliminates the performance interference due to IDD handling multiple VMs' request to a shared device. Also, because the I/O access is now directly done by the VM, the service time on the I/O access reduces thereby resulting in better bandwidth utilization. With the virtual-NIC interface, data transfer is handled by the VM. While initializing the device driver for the virtual NIC the VM sets up the Rx/Tx descriptor rings within its address space and makes request to the VMM for initializing the I/O page translation table. The device driver uses this table and does DMA directly into the VM's address space.
- **QoS and virtual-NIC:** The device memory partition acts as a dedicated device buffer for each of the VMs and with appropriate logic on the NIC card one can easily implement QoS based SLAs on the device that translate to bandwidth restrictions and VM based processing priority. The key is being able to identify the incoming packet to the corresponding VM, which the NIC is now expected to do. While communicating, the NIC controller decides on whether to accept or reject the incoming packet based on the bandwidth specification or the device memory free level. This gives a fine-grained control on the incoming traffic and helps reduce the interference effects. The outbound traffic can be controlled by the VM itself, as is done in the existing architectures.

3.2 Network Packet workflow using the virtualized I/O architecture:

With the proposed I/O device virtualization architecture, each VM gets direct access to the shared I/O device without having to route the request through the IDD. Only the device interrupts get routed through the VMM. In Figure 4a, and Figure 4b the workflow for network data reception and transmission using the described device virtualization architecture is depicted. When a packet arrives at the NIC, it deciphers the destination address of the packet, checks if it is a valid destination, then copies the packet into the destination VM's portion of the device memory and issues DMA to the destination VM based on the virtual NIC's priority. On completion of the DMA the device raises an interrupt to the VMM. The VMM intercepts the interrupt, determines the destination VM, forwards the interrupt to the VM and schedules it. The VM's device driver then receives the data from the VM specific device descriptor rings as it would do in the case of non-virtualized server.



a: Packet reception workflow



b: Packet transmission workflow

Figure 4: Workflow of network I/O communication with improvised I/O device virtualization architecture.

In the case of transmission, the process is same as in the case of non-virtualized server, except that the VM's device driver DMA's data directly into the device memory allocated to its virtual-NIC. It may be worthwhile to note here that the code changes to support this architecture in the existing implementation will not be excessive. Each VM can use the native device driver for the exported virtual device interface. This device driver is the standard device driver for the IOV complaint devices with the only difference that it has now restricted device access. The device access restrictions in terms of memory, DMA channels, interrupt line and device register sets are setup by the VMM when the VM requests for a virtual device. With the virtual device interface the VMM now only has to implement the virtual device interrupts.

4. Architecture Evaluation

Since the architecture involves design of a new NIC and a change in both VMM and VM device handling code, we first choose to evaluate the architecture using simulation based on layered queuing network (LQN)[11] models to understand the benefits. The reason for choosing LQN based modeling is twofold. One, there is a lack of appropriate system simulation tools that allow incorporating design of new hardware along-with VMM and VM OS changes. Second, LQN models are intuitive queuing models that enable capturing of the end-to-end workflow, right from the application to the device including the intermediate layers of the VM, IDD and VMM. With appropriate profiling tools, the LQN models are fairly easy to build and effective in terms of capturing the causes of bottlenecks in the access path. Also, LQN models capture device contentions when multiple processes share a common device. For further details on general description of LQN modeling and MOL refer [13].

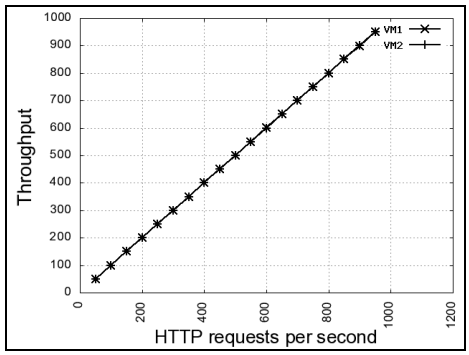
4.1 LQN model for the proposed architecture

We generate the LQN model manually using the LQNDEF [12] software developed at the *RADS* lab of *Carleton University*. For this paper we present results for the model generated for the *httperf* benchmark since the bottleneck issues are prominent for this benchmark. Complete details on generating of the LQN models and validating the models against experimental data for this benchmark are discussed in [27].

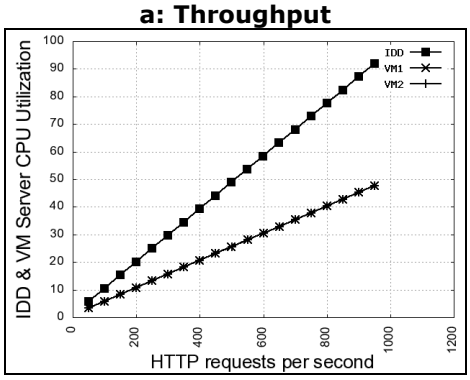
5. Simulation and Results

In order to evaluate the proposed architecture we use the *parasrvn* simulator of the LQNS software distribution [12] from *Carleton University*. The experimental results were obtained on a single core server to illustrate the issues of I/O device sharing on virtualized systems. As is observed in the later part of this section, these problems do not get resolved in the multi-core servers. Also, since the multi-core servers are now prevalent, we evaluate this architecture for such systems. For our study the LQN model consists of one VMM and two VMs and each of these is pinned to a different core. We validate the LQN model for the proposed architecture against the existing Xen architecture for a multi-core server. To observe the behavior on a multi-core server for the existing architecture, in the LQN model the VMM/IDD and each of the VMs are placed on an independent core. Figure 5 depicts the results of achievable throughput and server CPU utilization for a multi-core server with two VMs consolidated. The throughput graph for both the VMs is similar and appears overlapped in the chart. As can be noted from Figure 5, in a multi-core environment with Xen IDD, VM1 and VM2 each pinned to a core, and each VM servicing one httperf stream, the maximum throughput, without loss, achievable per stream is 950requests/s as against 450requests/s in the case of single-core. But, for the maximum throughput, we observe that the Xen-IDD, which is hosting the NIC of the server, the CPU utilization saturates. This indicates that further increase

in request rate is not possible since the processor core serving the Xen-IDD does not have any computing power left.



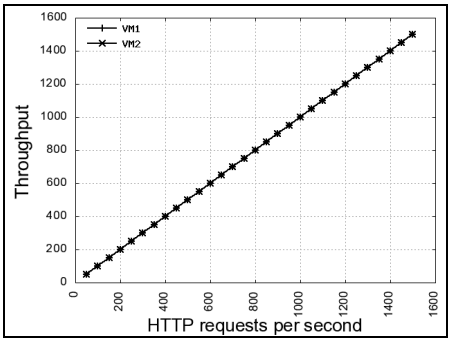
a: Throughput



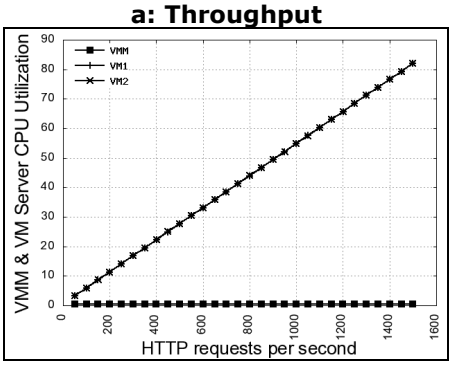
b: CPU Utilization

Figure 5: Charts for maximum throughput achievable per httperf stream and CPU utilization for existing Xen architecture on a multi-core server hosting two VMs each servicing one of the httperf stream. The IDD, VM1 and VM2 are pinned to independent cores.

Figure 6 shows these statistics for a similar situation but with the proposed architecture. We observe that the maximum throughput achievable now per stream increases to 1500 req/s, which is an increase of about 60% more. The total throughput achievable at the NIC, derived from consolidating the throughput of both the streams, also increases by 60% when compared to what was achieved on the existing architecture. If we look at the CPU utilization of each VMs, we observe that the Dom0 which is the VMM for the NIC, now consumes very less CPU. The reason for this is that, the NIC is now offloading the identity of the destination of the packet and this identification happens at hardware speeds. To account for this, a very low value of service time is assigned in the LQN model. Also, in the existing model, bridging software that routes the packets to a VM and which has substantial overhead, is done away with in the proposed architecture. The net result is improved throughput, reduced virtualization overhead, and reduction of VMM/IDD resource consumption on behalf of VMs.



a: Throughput



b: CPU Utilization

Figure 6: Maximum achievable throughput and CPU utilization charts for a multi-core Xen server incorporating the proposed I/O virtualization architecture and hosting two VMs, pinned to different cores, each servicing one httperf stream.

We also notice that the VMM is now spending almost constant time which results in eliminating the performance interference and also improves the scalability of sharing the device across VMs. With this architecture each VM is now accountable for all the resource consumption, thereby leading to better QoS controls.

6. Conclusion

In this paper we described how the lack of virtualization awareness in I/O devices leads to latency overheads on the I/O path. Added to this, the intermixing of device management and data protection issues further increases the latency, thereby reducing the effective usable bandwidth of the device. Also, lack of appropriate device sharing control mechanisms, at the device level lead to performance interference on the device sharing VMs. To address these issues we proposed I/O device virtualization architecture, as an extension to the PCI-SIG IOV specification, and demonstrated its benefit through simulation techniques. The architecture evaluation was done by capturing it as an LQN model and analyzing using simulation of the model. The preliminary simulation results show a utilization benefit of about 60%, without enforcing any

QoS guarantees or applying any of the software optimization techniques to the I/O path. The proposed architecture also improves the scalability of VMs sharing the NIC. Although the evaluation was done for para-virtualized systems like Xen, we believe the ideas proposed would benefit fully virtualized systems like VmWare too. The reason being, in any of these virtualization techniques I/O device sharing is currently regulated through a common software entity, which is eliminated in the proposed architecture.

References:

[1] M. Welsh and D. Culler, "Virtualization considered harmful: OS design directions for well-conditioned services", *Hot Topics in OS*, 8th Workshop, 2001.

[2] Robert P Goldberg, "Survey of Virtual Machine Research", *IEEE-Computer*, 1974, vol7, no.6, p34-45.

[3] Rick A. Jones, 'netperf: A Network Performance Benchmark,' Revision 1. Information Networks Division, Hewlett-Packard Co., March 1993.

[4] D. Mosberger and T. Jin, "httperf: A Tool for Measuring Web Server Performance," *ACM, Workshop on Internet Server Performance,* pp. 59-67, June 1998.

[5] Paul Barford and Mark Crovella. "Generating Representative Web Workloads for Network and Server Performance Evaluation.", *In Proceedings of the ACM SIGMETRICS*, p151-160, Nov.1998. ACM.

[6] Paul Barham , Boris Dragovic , Keir Fraser , Steven Hand , Tim Harris , Alex Ho , Rolf Neugebauer , Ian Pratt , Andrew Warfield, "Xen and the art of virtualization", *19th ACM SIGOPS*, Oct. 2003.

[7] K. Fraser, et.Al. "Safe hardware access with the Xen virtual machine monitor." *1st Workshop on OASIS*, Oct 2004.

[8] PCI-SIG IOV Specification available online at http://www.pcisig.com/specifications/iov

[9] M. Frans Kaashoek, et. Al., "Application Performance and Flexibility on Exokernel Systems ", 16th ACM SOSP, Oct, 1997.

[10] Scott Rixner, "Breaking the Performance Barrier: Shared I/O in virtualization platforms has come a long way, but performance concerns remain", *ACM Queue – Virtualization*, Jan/Feb 2008.

[11] C. M. Woodside, J. E. Neilson, D. C. Petriu and S. Majumdar, "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software", *IEEE Trans. on Computers*, vol. 44, no. 1, January 1995, pp. 20-34.

[12] Layered Queueing Network Solver software package, http://www.sce.carleton.ca/rads/lqns/

[13] J.A. Rolia and K. C. Sevcik, "The Method of Layers", IEEE Transactions on Software Engineering, Aug 1995.

[14] "VMware ESX Server 2 - Architecture and Performance Implications", 2005, available at http://www.vmware.com/pdf/esx2_performance_implications.pdf

[15] Willmann, P., Shafer, J., Carr, D., Menon, A., Rixner, S., Cox, A. L., Zwaenepoel, W. Concurrent direct network access for virtual machine monitors. In Proceedings of the International Symposium on High-Performance Computer Architecture,2007 (February).

[16] H. Raj and K. Schwan. Implementing a scalable self-virtualizing network interface on a multicore platform. In *Workshop on the Interaction between Operating Systems and Computer Architecture*, Oct. 2005.

[17] J. Liu, W. Huang, B. Abali, and D. K. Panda. High performance VMM-bypass I/O in virtual machines. In P*roceedings of the USENIX*, June 2006.

[18] Menon, A. L. Cox, and W. Zwaenepoel. Optimizing network virtualization in Xen. In *Proceedings of the USENIX Annual Technical Conference*, June 2006.

[19] 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 ACM/USENIX Conference on Virtual Execution Environments*, June 2005.

[20] J. Sugerman, G. Venkatachalam, and B. Lim. Virtualizing I/O devices on VMware Workstation's hosted virtual machine monitor. In *Proceedings of the USENIX Annual Technical Conference*, June 2001.

[21] T. von Eicken and W. Vogels. Evolution of the virtual interface architecture. *Computer*, 31(11), 1998.

[22] Santos, J. R., Janakiraman, G., Turner, Y., Pratt, I. 2007. Netchannel 2: Optimizing network performance. Xen Summit Talk (November).

[23] Intel Virtualization Technology for Directed-I/O www.intel.com/technology/itj/2006/v10i3/2-io/7-conclusion.htm

[24] Neterion http://www.neterion.com/

[25] NetXen http://www.netxen.com/

[26] CrossBow: Network Virtualization and Resource Control http://www.opensolaris.org/os/community/networking/crossbow_sunlabs_ext.pdf

[27] J.Lakshmi, S.K.Nandy, "Modeling Architecture-OS interactions using Layered Queuing Network Models", to appear in International Conference Proceedings of HPC Asia, Mar. 2009, Taiwan.