Modeling Architecture-OS Interactions using Layered Queuing Network Models.

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

Abstract

The prevalent virtualization technologies provide QoS support within the software layers of the virtual machine monitor(VMM) or the operating system of the virtual machine(VM). The QoS features are mostly provided as extensions to the existing software used for accessing the I/O device because of which the applications sharing the I/O device experience loss of performance due to crosstalk effects or usable bandwidth. In this paper we examine the NIC sharing effects across VMs on a Xen virtualized server and present an alternate paradigm that improves the shared bandwidth and reduces the crosstalk effect on the VMs. We implement the proposed hardware-software changes in a layered queuing network (LQN) model and use simulation techniques to evaluate the architecture. We find that simple changes in the device architecture and associated system software lead to application throughput improvement of up to 60%. The architecture also enables finer QoS controls at device level and increases the scalability of device sharing across multiple virtual machines. We find that the performance improvement derived using LQN model is comparable to that reported by similar but real implementations.

1. Introduction

Multi-core systems are a logical evolution of single core processors due to saturation of computing power on a single core. In the HPC segment, such systems are an attractive alternative to mid-sized distributed clusters owing to inherent architecture benefits of low latency inter-processor communication. In the enterprise segment, coupled with system virtualization, multi-core systems are being promoted as the one-stop solution to enterprise data-center issues like server outages, energy consumption, support for legacy applications, etc. Apart from this, multi-core systems are also being seriously considered for consolidating on-board real-time applications in automobile and aerospace application domains.

The current multi-core systems are well-suited to facilitating compute workloads that exploit the presence of many processors on chip and on-board. Also, most efforts are directed at improving system architectures for such workloads. In compute intensive parallel jobs, inter task/process communication or file I/O handling, are well characterized and designed in such a way that, very specific tasks carry out the I/O activity. While in enterprise segments, where the workloads are dominated by a mix of I/O and compute intensive jobs, multi-core systems have a disadvantage due to the predominance of CPUs over I/O devices. Most server consolidation scenarios involve hosting on to a physical machine, large numbers of serial or parallel applications that require significantly less number of CPUs when compared to scientific parallel jobs. This brings in the question of many independent jobs sharing a common I/O device like the network interface (NIC) or a local disk. Many of these workloads are either response sensitive or throughput sensitive because of which it is essential to specify, maintain and cater to application specific quality of service (QoS) guarantees [1].

In the prevalent virtualization technologies QoS support is mostly provided as extension to the existing software used for accessing the I/O device. As an example, consider a Xen [2] virtualized server hosting multiple VMs that share a common NIC of the server. In Xen the NIC is accessed through a privileged domain called the independent driver domain (IDD) [3] that hosts the device driver of the interface and is responsible for the physical data transfer to and from the machine. Any VM intending to use the NIC is given access through the virtual network interface which is implemented in software by the netfront and netback drivers. The netfront driver is hosted in the VM and netback is hosted in the IDD. In such virtualized systems, for every I/O device that is shared, the IDD hosting the device is also shared amongst the VMs. Each VM can have its own ip-address which is enabled by bridging or routing software within the IDD. If a VM requests for specific bandwidth on the incoming or outgoing traffic, this is implemented in software, like netfilter[5] of linux in the IDD. Supporting QoS features in software does not give fine-grained control on device access which leads to either crosstalk* or loss of bandwidth on the device

* By crosstalk we refer to loss of performance observed in one VM because of sharing the NIC with another VM.

access path. In this paper we examine the NIC sharing effects across VMs on a Xen virtualized server and present an alternate paradigm that improves the shared bandwidth and reduces the crosstalk effect on the VMs. We implement the proposed hardware-software changes in a layered queuing network (LQN) model and use simulation techniques to evaluate the architecture.

The rest of the paper is organized as follows. In section 2 we detail the effects of NIC device sharing in the existing Xen virtual machine architecture to bring out the motivation for this work. In section 3 we define the goals and detail improvements on the existing NIC architecture, the VMM and VM's OS. In this section we also detail the modeling of the architecture using layered queuing network model and describe the network data reception and transmission workflow for the architecture. In section 0 we describe the architecture evaluation and present the results. Finally, in section 5 we conclude highlighting the issues of shared I/O devices in virtualized systems and ways of tackling these issues.

2. Effects of I/O device sharing

It is expected that any device sharing would lead to some overheads and this issue is more prominent in virtualized multi-core systems. In a non-virtualized server, for general purpose operating systems like Unix, I/O is performed by routing every device's access through the operating system's I/O system call interface. Virtualization adds one more layer of indirection to this path; in Xen this additional overhead manifests due to the data transfer and page address translations at the IDD hosting the device and VM layer, and the virtualization of device interrupts through the event channel. How this overhead affects the realized throughput, measured as reply rate, at the application level is depicted in Figure 1. The figure contains a plot of throughput achieved by the httperf [4] benchmark for a specified request rate. The throughput is measured as the reply rate of the http server in response to httperf request rate. The chart shows three graphs each depicting the case when an http server is hosted on a non-virtualized, virtualized and a virtualized-consolidated single-core server. The non-virtualized server is realized on a standard linux OS and the virtualized server on Xen and Xeno-linux.

The virtualized and virtualized-consolidated servers differ by the number of virtual machines running on the virtualized system. In the case of virtualized server, only one Xen-VM hosting the http server is activated. In the case of virtualized-consolidated server, two Xen-VMs are activated, both of them sharing the same NIC. Each of the VMs hosts an http server that responds to a different httperf client. From Figure 1 we observe that while the throughput increases linearly with the increase in request rate for the non-virtualized server, there is a gradual drop of throughput for the virtualized server from the request rate of 500reqs/s. This is due to the virtualization overheads on I/O operations caused due to not only sharing of the device but also sharing of the VMM and the IDD. This sharing increases the device access latency and causes loss of bandwidth thereby loss of achievable application throughput.

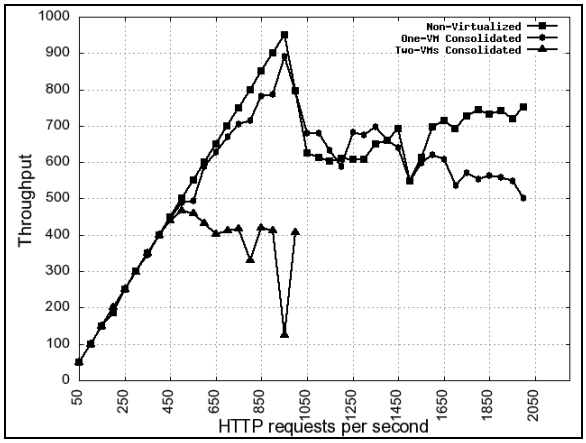


Figure 1: httperf throughput chart for http server hosted on a non-virtualized and virtualized Xen server.

This loss of bandwidth is additive and is depicted in the case of consolidated server. The non-virtualized server achieves a throughput of 950 replies/s for a request rate of 950 requests/s, after which the server exhibits packet loss which results in throughput loss for the application. For the virtualized servers the loss of throughput starts off as early as 500 requests/s, in the case of single VM consolidated server, and at 450 requests/s for the two-VMs consolidated server. In Figure 1 each graph refers to the achievable throughput for a single client of httperf. We also observe that for one-VM consolidated server case the virtualization overheads do not allow complete utilization of the NIC bandwidth when compared to the non-virtualized case. For the two-VMs consolidated server case, the per VM achievable throughput without, any loss, further drops but the total throughput, for both VMs put together, supported by the NIC is 10% less than the non-virtualized server. This raises the issue of scalability in terms of sharing the I/O device with multiple VMs. We notice that in Xen with an addition of VM, sharing the NIC, a reduction in the usable bandwidth of the device occurs. This indicates the need for reducing the device access latencies and also fine-grained QoS controls for

device shared by the VMs. In Xen, I/O device access is routed through the IDD, which incurs CPU resource consumption on behalf of the VM whose access it is servicing. There is no control on the amount of this resource consumption on a per VM basis [6]. This can cause crosstalk effect leading to performance loss in one VM due to an errant application in the other VM. Also, in terms of device resource and bandwidth limiting, the controls are within the IDD. While this control may work well on the outgoing route, on the incoming route the NIC receives the packet after which the IDD takes a decision on acceptance or rejection. The effort spent on a rejected packet manifests as loss of utilizable bandwidth on the device path.

3. Enabling QoS at the device level

In order to overcome the crosstalk effect and performance loss we propose an alternate architecture for the NIC. The goals of this design are:

- I/O devices to be made virtualization aware; physical device should enable logical partitioning of device resources guided by QoS guarantees.
- VMM to control and define a virtual device, using the logical partitioning of a device. A virtual device is exported to a VM.
- The virtual device is private to a VM and is managed by the VM. IDD is eliminated.

The proposed device architecture supports multiple virtual devices on a single physical device. Each virtual device is defined using a subset of device resources and is exported through a virtual device interface. The virtual device interface incorporates identity and protection in the virtual device itself. The VMM is modified to manage the virtual device to VM association and handle the device interrupt virtualization. We eliminate the IDD functionality and replace it by the device driver resident within the VM. This device driver accesses the virtual device interface rather than the physical device which allows for native device access to the VM and thus reduces the latency on the device access path. Since a single device can now support multiple virtual devices, concurrent device access support is built into the device and the VMM. To allow for direct access into a VM, the device and the VMM also support I/O device page translation tables that are initialized based on the virtual device interface.

3.1 NIC architecture description

Figure 2 gives a block schematic diagram of the proposed architecture. The picture depicts a NIC card that would be housed within a multi-core server. The card would have a controller that manages the DMA transfer to and from the device memory. The standard device memory is now replaced by the resizable memory partitions supported with *n* sets of device registers, multiple DMA channels and interrupt lines. A device memory partition, a specific set of device registers along-with dedicated DMA channels and interrupt line forms the virtual-NIC. Ideally the device memory should be reconfigurable and the VM's QoS requirements would drive the sizing of this memory. The controller is capable of generating message signaled interrupts (MSI). The number of interrupts supported by the controller restricts the number of virtual-NICs that can be exported. The proposed architecture can be achieved by the following modifications.

- **Virtual-NIC:** Device hardware should support time-sharing in hardware. For a NIC this can be achieved by using MSI and concurrent access to multiple device memory partitions. Each virtual device has a specific logical device address, like the MAC address in case of NICs, based on which the MSI is routed. The virtual-NIC, which is now a subset of the physical device resources, is exported to a VM when it is started. It forms a restricted address space on the device and is usable only by the VM it is exported to. The VM retains the possession of the virtual-NIC till it is active or till it relinquishes the virtual-NIC. The VM possessing the virtual-NIC can use it for native device access or can become the IDD for VMs wanting to share the virtual-NIC.

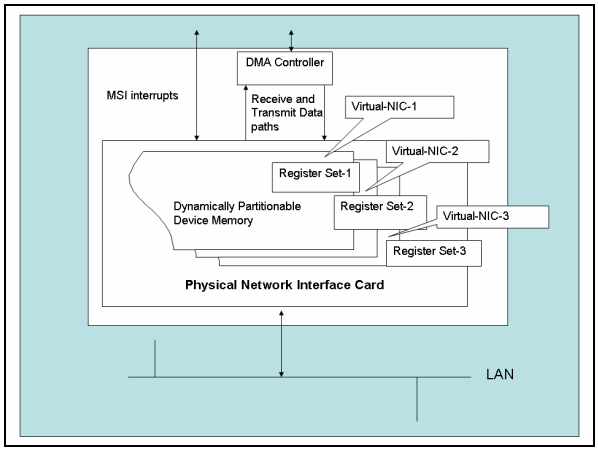


Figure 2: Proposed **a**rchitecture of the NIC supporting MSI interrupts with 3 device memory partitions and three device register sets enabling three virtual-NICs.

- **Accessing virtual-NIC:** For accessing the virtual-NIC the hypervisor 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 and handles concurrent device access, eliminating the IDD altogether. The OS of the VM now handles the I/O access and thus can be accounted for the resource usage it incurs [Figure 3]. This eliminates the VM-crosstalk due to the sharing of IDD.

- **QoS and virtual-NIC:** The device memory partition acts as a dedicated device buffer for each of the VMs. With appropriate logic on the NIC one can easily implement QoS based service level agreements (SLAs) on the device. For example the QoS requirement of specific bandwidth can be implemented by allocating appropriate device memory to the virtual-NIC interface exported to the VM. While communicating, the NIC controller then 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 crosstalk effects. The outbound traffic can be controlled by the VM itself, as is done in the non-virtualized server. The NIC controller can also implement the notion of priority by raising the appropriate MSI depending on the assigned priority of the VM.

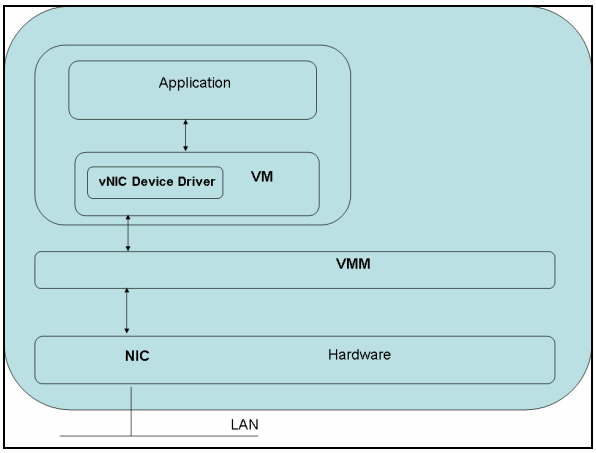


Figure 3: Architecture of system software for the proposed virtual-NIC.

3.2 Network Packet workflow

In Figure 4 and Figure 5 the workflow for network data reception and transmission using the proposed device virtualization architecture is shown. When a packet arrives at the NIC, it deciphers and checks the destination address of the packet, then copies the packet into the destination VM's portion of the device memory. The VM's device driver receives the data from the VM specific device memory using DMA as it would do in the case of non-virtualized server. Subsequently the device would raise an interrupt that is received by the VMM and forwarded to the respective VM. In the case of transmission, the reverse process of reception occurs.

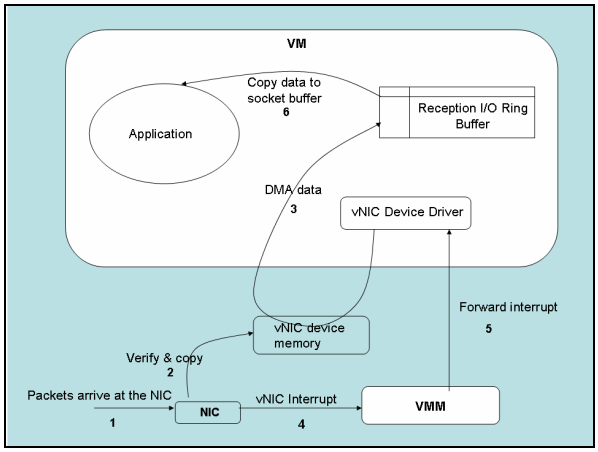


Figure 4: Network packet reception workflow.

Concurrent device access by multiple VMs is enabled by the supporting multiple DMA channels on the device with MSI. The proposed NIC architecture assumes an intelligent device that offloads tasks like virtual device interface identification, DMA channel arbitration based on priority, etc.

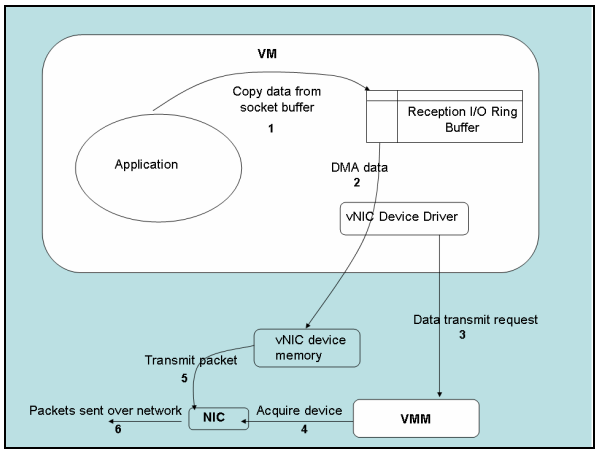


Figure 5: Network packet transmission workflow.

Each VM will now install a device driver that understands the virtual device interface. This device driver is similar to the standard device driver of the

non-virtualized server but works with a restricted device address. Also, since the device is exported for native VM access, the device driver uses the I/O page translation tables that are setup by the VMM during the virtual device initialization.

3.3 Layered Queuing Network (LQN) modeling

The proposed architecture involves changes in the hardware and system software accessing and using the hardware. In order to evaluate the end-to-end application performance based on these architecture-OS interaction changes, we need a setup to capture the architecture parameters in terms of the interacting component service times and the associated queuing delays when sharing software components and/or devices. LQN models are layered queuing network models that capture such software and device contentions when multiple processes share a common software or device. Using method of layers (MOL) on such models one can make performance estimates and studies of the modeled system. Various issues like which server in the system is a bottleneck, what kind of throughput or response times are sustainable in the given setup, what are the server and device utilizations for a given workload, etc., can be studied in detail using LQNs. For further details on LQN modeling and MOL refer [7] and [8] respectively. Figure 6 is a diagrammatic representation of the device and software sharing in the proposed architecture. In the picture, solid lines show software interaction, and dashed lines indicate usage of a device, like a processor or NIC card, by any of the software component.

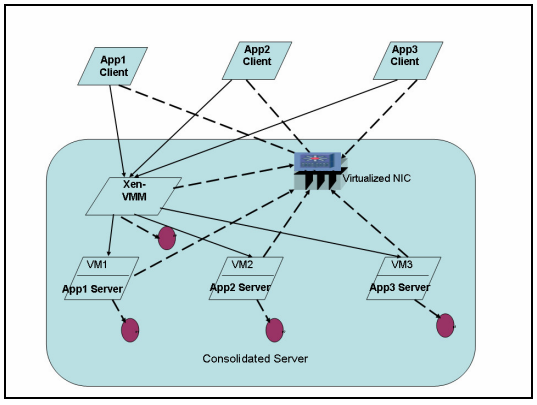


Figure 6: Software and device contention model for proposed I/O device sharing architecture.

Merging of lines at a software component or a device indicates contention. LQNs allow for intuitive modeling of the system of interest from the interaction workflow. For this study we generate the LQN model manually using the LQNDEF [9] software developed at the *RADS* lab of *Carleton University*. In the model each functional unit is modeled as an entity and the interactions across these entities are modeled as synchronous or asynchronous communication links based on the actual implementation in the system.

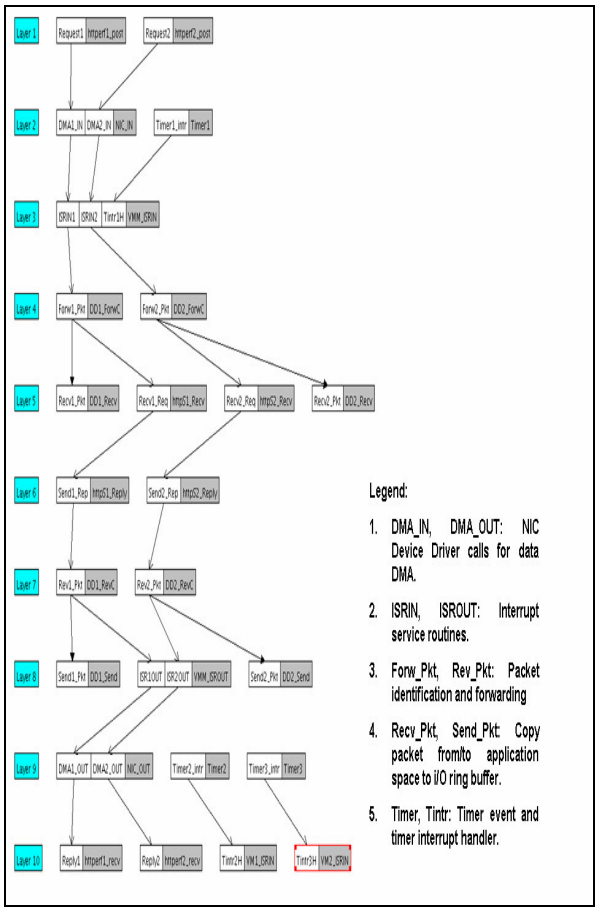


Figure 7: LQN model generated for a consolidated Xen server incorporating the proposed architecture. The server is modeled to be hosting two virtual machines, each catering to an httperf stream

Synchronous communication is used where the requesting entity can service requests serially. For example, when the IDD/VM is receiving data from the device buffer into I/O ring buffer it is a synchronous and a blocked operation. This blocking causes queuing delays at the shared device or software and is captured using synchronous communication link in the LQN model. In the case of multiple kernel entities, like device driver DMA transmit and receive calls, only one of them can be active at a time. We represent such

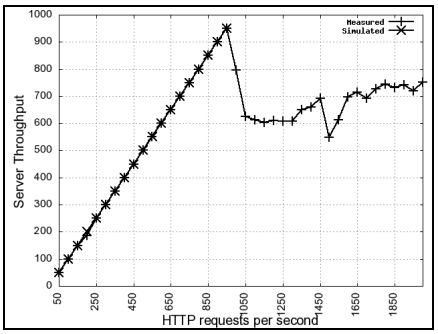
software components as tasks within an entity, to capture serialization. For example, between the timer and device interrupt, only one can be serviced at a time. The LQN model follows the interaction workflow and each entity of the model represents receive or transmit component of the workflow functional element. We model the LQN as an open queuing network model and measure the performance in terms of the maximum throughput achieved. The LQN model used for evaluating the proposed architecture is depicted in Figure 7. The service times [Table 1] of the entities of the LQN model are arrived at using the xenoprof[10] kernel profiler. All the service times specified are for a single http request and are measured in seconds. We use the httperf request rate to represent the mean arrival rate of the workload for the LQN. We make one assumption to simplify the LQN model; while in reality every *http get* request is broken into a sequence of packets that are passed through the various layers of OS, we model it as a single service request. This assumption tends to give optimistic throughputs in the simulation results since it does not capture the packet queuing delays. As can be observed from the validation graphs in Figure 8 for the application throughput, the deviation of the simulation results from the observed measurements is less than 10% and is thus suitable for evaluation of the architecture. One element that is incorporated in the LQN model and not shown in the workflow is the system timer interrupt using the server element "Timer". This element is introduced in the LQN to account for the queuing delays accrued while the OS is handling timer interrupts.

Table 1: Execution service time demands for entries used in LQN model.

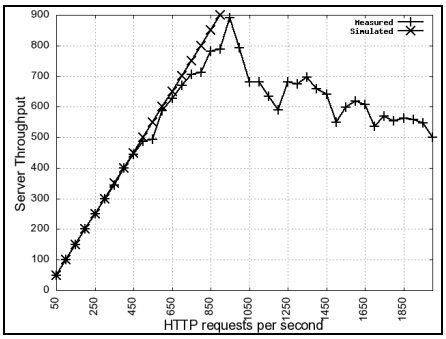
Task Name	Entry Name	Phase 1	Phase 2
httperf1_post	Request1	1e-10	0
NIC_IN	DMA1_IN	9.24e-05	0
NIC_IN	DMA2_IN	9.24e-05	0
VMM_ISRIN	ISRIN1	1e-10	0
VMM_ISRIN	ISRIN2	1e-10	0
VMM_ISRIN	Tintr1H	4.7783e-05	0
DD1_Recv	Recv1_Pkt	2.3297e-05	0
httpS1_Recv	Recv1_Req	0.00021069	0
httpS1_Reply	Send1_Rep	0.00021069	0
DD1_RevC	Rev1_Pkt	1e-10	3.7767e-05
DD1_Send	Send1_Pkt	2.3297e-05	0
VMM_ISROUT	ISR1OUT	1e-10	0
VMM_ISROUT	ISR2OUT	1e-10	0
NIC_OUT	DMA1_OUT	9.24e-05	0
NIC_OUT	DMA2_OUT	9.24e-05	0
httperf1_recv	Reply1	1e-10	0
Timer1	Timer1_intr	1e-10	0
DD1_ForwC	Forw1_Pkt	3.7767e-05	1e-10
httperf2_post	Request2	1e-10	0
httpS2_Recv	Recv2_Req	0.00021069	0
httpS2_Reply	Send2_Rep	0.00021069	0
httperf2_recv	Reply2	1e-10	0
VM1_ISRIN	Tintr2H	4.7783e-05	0
VM2_ISRIN	Tintr3H	4.7783e-05	0
Timer2	Timer2_intr	1e-10	0
Timer3	Timer3_intr	1e-10	0
DD2_ForwC	Forw2_Pkt	3.7767e-05	1e-10
DD2_Recv	Recv2_Pkt	2.3297e-05	0
DD2_RevC	Rev2_Pkt	1e-10	3.7767e-05
DD2_Send	Send2_Pkt	2.3297e-05	0

4. Evaluation of Architecture

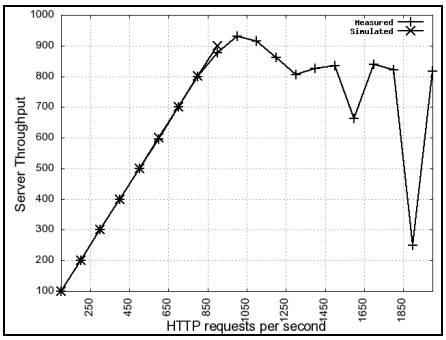
To evaluate the proposed architecture we use the parasrvn[9] simulator of the LQNS software distribution from *Carleton University*. First we generate and validate the LQN model for the existing network I/O workflow in Xen with the experimental data. As can be observed from Figure 8 a,b, and c, LQN simulation results are almost in accordance to the observed values.



a: httperf throughput for non-virtualized server



b: httperf throughput for one-VM consolidated server



c: httperf throughput for two-VMs consolidated server

Figure 8: LQN model validation for httperf throughput against experimental data for the http server hosted on a non-virtualized and virtualized Xen VM for a single-core system.

The cut-off httperf request rate for the simulated results corresponds to the stage where the server is saturated. The experimental data collected was for a single core machine and the LQN model is validated for such a system. In order to understand the benefits of the existing Xen-virtualization architecture on a multi-core system, we extend the LQN model to a multi-core server by hosting the VMM/IDD and each of the VMs on an independent core. The expected difference between a single core and multi-core environment in terms of VMM overheads are VM context switching for the single core server and timer scheduling across multiple cores for the multi-core server. Each of these overheads is about 10% of the observed experimental CPU utilization values and one effect compensates the other. We have not included these effects into the LQN model to keep it simple. Figure 9 and Figure 10 below depict the results of achievable throughput and server CPU utilization, respectively, for a multi-core server with two VMs consolidated. In both the graphs, the curves for the VMs are overlapping.

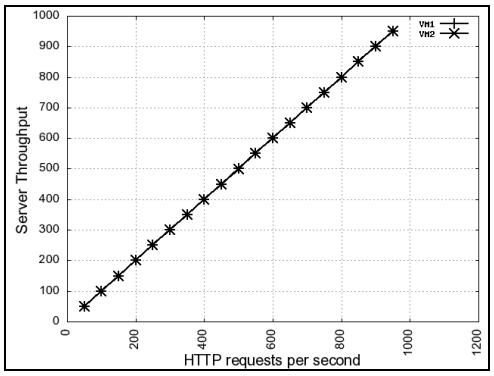


Figure 9: httperf throughput graph with two VMs consolidated on a multi-core Xen server with the VMM/IDD, and the VMs pinned to an independent core.

The throughput graph for both the VMs is similar and appears overlapped in the chart. As can be noted from Figure 9, 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 achievable per stream is 950requests/s as against 450requests/s in the case of single-core hosting both the VMs and the IDD. Moving to a multi-core server the consolidated throughput, including both the streams, at the NIC doubles. But, for the maximum throughput, we observe that the Xen-IDD CPU utilization saturates [Figure 10]. This indicates that further increase in throughput is not possible since the IDD's CPU does not have any computing power left.

Figure 11 and Figure 12 show these statistics for a similar situation but with the proposed architecture. As we can observe, the maximum throughput achievable per stream increases to 1500 req/s, which is an increase of about 60% more throughput.

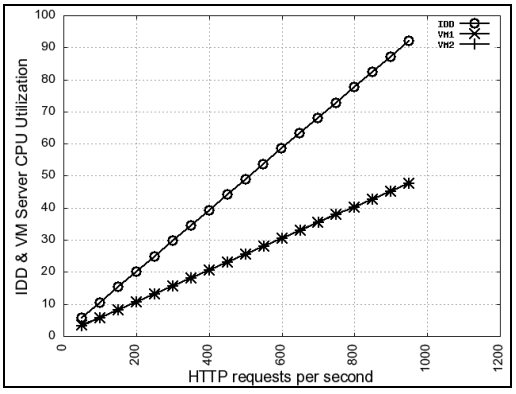


Figure 10: http server CPU utilization graph for two VMs consolidated on a multi-core Xen server with the VMM/IDD, and the VMs pinned to an independent core

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 the VMs and the VMM, we observe that the CPU utilization by the VMM is very small. The reason for this is that, the NIC is now offloading the identity of the destination of the packet from the IDD and this identification happens due to MSI based interrupts which execute at hardware speeds.

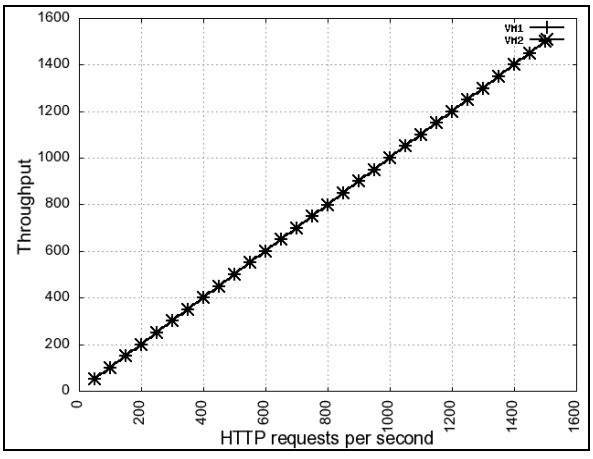


Figure 11: Maximum achievable httperf throughput for a multi-core consolidated Xen server hosting two VMs incorporating the proposed I/O virtualization architecture.

Hence a very low value of service time is assigned for this service in the LQN model. Also, in the existing model, bridging software that takes care of routing the packets to a VM, has a substantial overhead. This is done away with in the proposed architecture.

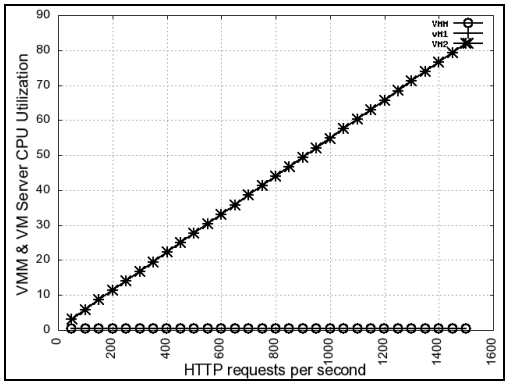


Figure 12: CPU Utilization of VMM and VM for the consolidated multi-core Xen server incorporating the proposed I/O virtualization architecture and hosting two VMs.

The net result is improved throughput, reduced virtualization overhead, and reduction of VMM resource consumption on behalf of VMs. We also notice that the CPU utilization by the VMM is almost constant which results in eliminating the crosstalk effect 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. These results of performance improvements predicted by the LQNs model in this paper are comparable with the improvements achieved and reported in [11] and [12]. The reason that the reported improvements in this paper are slightly lower (60% versus 70%) than what is reported in literature is due to the fact that we are reporting performance improvements at the application level, while in [11] and [12] the performance is measured at the interface. It is understandable that moving from the interface to the application there will be software overheads involved due to processing across various network layers.

5. Conclusion

In this paper we have explored the issues, from the point of view of enabling QoS guarantees, involved in sharing I/O devices in virtualized systems. We established the issues of loss of performance due to crosstalk and loss of usable bandwidth existing in Xen architecture for a NIC that is shared across two virtual machines. We then proposed a new I/O device virtualization architecture to overcome these issues. We used layered queuing network models to implement and evaluate the proposed architecture. We found that simple changes in the I/O device architecture and the associated system lead to improvements in application throughputs of up to 60%. We also found that the performance improvement predicted by the LQN model is comparable to that of reported real implementations. Hence, the proposed architecture can be easily extended to other I/O devices like disk/storage and memory and LQN model used to evaluate the applicability and improvement in overall application performance.

6. 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] 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.

[3] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. War_eld, and M. Williamson, "Safe hardware access with the Xen virtual machine monitor." *1st Workshop on OASIS*, Oct 2004.

[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] Nils Radtke, "Linux Ethernet Bridge + Netfilter HOWTO", available online at http://www.linux.org/docs/ldp/howto/Ethernet-Bridge-netfilter-HOWTO.html

[6] L. Cherkasova and R. Gardner, "Measuring CPU overhead for I/O processing in the Xen virtual machine monitor." In USENIX Annual Technical Conference, Apr 2005.

[7] 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.

[8] J.A. Rolia and K. C. Sevcik, "The Method of Layers", IEEE Transactions on Software Engineering, Vol. 21, No.8, Aug 1995.

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

[10] Menon, J. R. Santos, Y Turner, and G. Janakiraman, "Xenoprof - Performance profiling in Xen" http://xenoprof.sourceforge.net/xenoprof_2.0.txt

[11] Himanshu Raj and Karsten Schwan, High performance and scalable I/O virtualization via self-virtualized devices, HPDC'07, p179-188.

[12] Willmann, P., Shafer, J., Carr, D., Menon, A., Rixner, S., Cox, A. L., Zwaenepoel, W. Concurrent direct network access for virtual machine monitors. HPCA 2007 (February).