Software Techniques to Improve Virtualized I/O Performance on Multi-Core Systems

Guangdeng Liao[1], Danhua Guo[1], Laxmi Bhuyan[1], Steve R King[2]
[1]Department of Computer Science and Engineering
University of California, Riverside
{gliao, dguo, bhuyan}@cs.ucr.edu
[2]Communication Technology Lab, Intel Corporation
steven.r.king@intel.com

ABSTRACT

Virtualization technology is now widely deployed on high performance networks such as 10-Gigabit Ethernet (10GE). It offers useful features like functional isolation, manageability and live migration. Unfortunately, the overhead of network I/O virtualization significantly degrades the performance of network-intensive applications. Two major factors of loss in I/O performance result from the extra driver domain to process I/O requests and the extra scheduler inside the virtual machine monitor (VMM) for scheduling domains.

In this paper we first examine the negative effect of virtualization in multi-core platforms with 10GE networking. We study virtualization overhead and develop two optimizations for the VMM scheduler to improve I/O performance. The first solution uses cache-aware scheduling to reduce inter-domain communication cost. The second solution steals scheduler credits to favor I/O VCPUs in the driver domain. We also propose two optimizations to improve packet processing in the driver domain. First we re-design a simple bridge for more efficient switching of packets. Second we develop a patch to make transmit (TX) queue length in the driver domain configurable and adaptable to 10GE networks. Using all the above techniques, our experiments show that virtualized I/O bandwidth can be increased by 96%. Our optimizations also improve the efficiency by saving 36% in core utilization per gigabit. All the optimizations are based on pure software approaches and do not hinder live migration. We believe that the findings from our study will be useful to guide future VMM development.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—Scheduling

General Terms

Measurement, Performance, Design, Experiments.

Keywords

Virtualization, VMM Scheduler, 10GE, Xen, Multi-Core.

ANCS'08, November 6-7, 2008, San Jose, CA, USA.

1. INTRODUCTION

Virtualization technology is experiencing a surge of interest from both academia and industry for functional isolation, manageability and live migration on volume servers. In such environments, the virtual machine monitor (VMM) virtualizes not only the machine's computing resources but also network facilities. Multiple operating systems running concurrently in different virtual machines (VM) enjoy the illusion of ownership of network resources. As more and more network-active servers are consolidated, we see a migration toward high performance networks in the virtualized endpoint server.

Although traditional Ethernet based network architectures such as Gigabit Ethernet lag in performance as compared to other high performance networks (e.g., InfiniBand [17], Quadrics [27], Myrinet [3]), they continue to be the most widely used network architecture nowadays. This trend is mainly attributed to the low component cost and backward compatibility with the existing Ethernet infrastructure [9]. As of 2006, Gigabit Ethernet-based clusters make up 176 (or 35.2%) of the top-500 supercomputers [30]. Thus, as performance pressure on the consolidated network grows, we strongly believe that industry will quickly migrate to 10GE networks to accommodate the increasing workload.

Unfortunately, the overhead of network I/O virtualization can significantly affect the performance of network intensive applications [22, 23, 24], especially under high speed networks. We conduct experiments on an Intel machine with two quad-core processors and 10GE networking to evaluate the virtualization impact on server networking performance. Our machine used Xen [2] as the VMM. In our experiments, a guest domain (DomU) only achieves 2.3Gbps bandwidth, compared to 5.5Gbps in

Linux, while consuming many more CPU cycles. Some researches [6, 25] are trying to allow domains to directly access hardware to improve bandwidth. These approaches take advantage of new Ethernet NIC features like multiple TX/RX queues. However, the direct access approach inhibits live migration and portability, which are the two major incentives for virtualization in high end servers [2, 5, 8]. Therefore, accelerating virtualized I/O processing in a hardware independent manner becomes an important problem.

We show that two major factors of loss in virtualized I/O performance result from the extra driver domain to process I/O requests and the extra scheduler inside the virtual machine monitor for scheduling domains. The VMM must schedule Dom0 and DomU instantly and in the correct order, otherwise increased latency from scheduling delay degrades bandwidth [7]. In this paper we conduct experiments to study the impact of the VMM scheduler and Dom0 on a multi-core platform with 10GE networking. We then develop two optimizations for the default VMM scheduler: cache-aware scheduler and a credit-stealing policy. Since the VMM directly copies all received packets from Dom0 to DomU in the contemporary I/O split model, our cache-aware scheduler attempts to schedule the DomU on the core, which shares last level cache with the core Dom0 currently residing in. This approach provides DomU with improved cache locality when accessing the packets, and substantially reduces inter-domain communication cost. The credit-stealing policy is a technique for the VMM scheduler to favor VCPUs in Dom0 that intensively process received packets (called as I/O VCPUs). In this technique, the VMM dynamically steals some credits for I/O VCPUs from other idle Dom0's VCPUs instead of fairly allocating credits among all of them. With this optimization, we increase the likelihood that the I/O VCPUs are running while receiving the NIC interrupt requests, thus accelerating packet processing.

Besides improving the VMM scheduler, we also identify packet processing overhead in Dom0 by instrumenting the source code at the component level. Our profiling shows that packet switching is one of the main bottlenecks. Thus we design a simplified bridge which retains the same user/kernel interface as the original bridge but reduces switch processing time by 10X for each packet. Additionally, in our experiments, we notice some losses of data within TCP streams resulting in TCP retransmissions. Our deeper investigation reveals that the retransmissions result from queue overflow in the network backend driver connecting the bridge to the DomU. Hence, we develop a patch to make backend queue length configurable and adaptable to 10GE networks.

A combination of all the above techniques can substantially improve the capability of virtualized systems to process high speed network traffic. Our experimental results reveal that the new virtualized I/O can achieve 4.5 Gbps corresponding to an increase of 96%. Additionally, we find that the efficiency for packet processing improves with a savings of 36% in core utilization per gigabit. Since all our optimizations are software based, they do not inhibit live migration and portability of virtual machines. Although we designed our techniques under 10GE network, they are also applicable to a low-end network like 1GE network.

The remainder of this paper is organized as follows. The background necessary to understand our work is described in section 2. Section 3 presents our experimental setup. Virtualization performance with a 10GE network is illustrated in section 4. The two optimizations of the VMM scheduler on multi-core architectures are described in section 5, followed by the two optimizations for Dom0 in section 6. Section 7 combines all techniques and presents the results. Finally section 8 presents the related work followed by our conclusion and future work in section 9.

2. BACKGROUND

2.1 Virtualization

Virtualization is a broad term that refers to the abstraction of physical computer resources. A typical virtualized platform consists of a software virtual machine monitor that "virtualizes/abstracts" the physical resource of the platform and provides a simulated environment that appears to the operating system as hardware. The most recent virtualization solutions are based on either total virtualization or para-virtualization. This total virtualization [28] demands relatively complex implementation of hypervisor which depends on techniques like ring compression and binary patching to enable this environment. This complexity and performance overhead in the hypervisor can be reduced by making the host hardware and the guest operating system aware of virtualization. The first type is referred to as hardware-assisted virtualization. Recently processor vendors like Intel and AMD are adding hardware support for virtualization [1, 14, 16] which reduces the burden on the hypervisor by providing the virtualization and isolation hooks in the hardware.

While hardware supported virtualization makes the hardware aware of virtualization, para-virtualization makes the guest OS aware of virtualization. Xen [2, 5, 8] is the most popular open source para-virtualized virtual machine monitor based on Linux. In this paper, we focus on the para-virtualized system Xen. It does not require changes to the application binary interface (ABI), and hence no modifications are required to guest applications. In particular, Xen exposes a hypercall mechanism that virtual machines must use to perform privileged operations, an event notification mechanism to deliver virtual interrupts and other notifications to virtual machines, and a shared memory based device channel for transferring I/O messages among virtual machines. While the OS must be ported to

this virtual machine interface, this approach leads to better performance than the approach based on pure virtualization.

2.2 Xen I/O Virtualization

The Xen I/O architecture has evolved from Hypervisor contained device drivers (Direct I/O) to a Split I/O [5, 8]. The primary goal of the Split I/O architecture is to provide isolation from misbehaved device drivers. Virtual machines in Xen usually don't have direct access to hardware. Each device driver is expected to run in a driver domain (Dom0), which hosts a backend driver to serve access requests from guest domains (DomU). The network architecture used in Xen is shown in Figure 1.

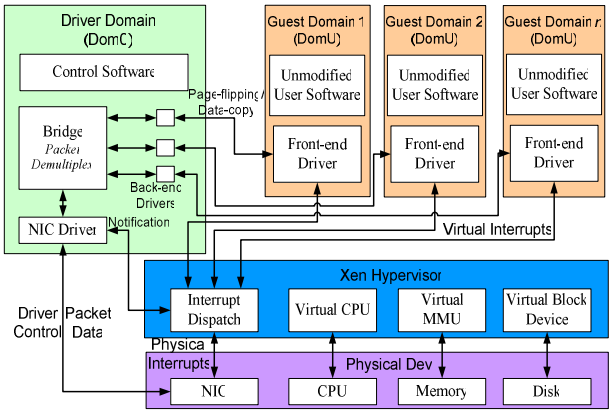

Figure 1. Xen I/O Split

Xen provides each DomU with a number of virtual network interfaces called Front-ends (FE), which are used by the DomU for all its network communications. Corresponding to each FE in a DomU, a backend Interface (BE) is created in the Dom0, which acts as the proxy for that FE. The FE and BE are connected to each other over an I/O channel. The recent I/O channel implements both data copy and page flip mode in receiving side with data copy as the default setting. However, during transmitting packets, only page flip mode is supported. Once inter-domain packet movements get finished, the virtual interrupt is sent to notify the target DomU of the packet. BE for each DomU is bridged with the real NIC driver in Dom0 by Linux Bridge module [20]. It functions as an internal crossover Ethernet bridge to switch the received packets to the corresponding attached ports through their MAC addresses. Finally real NIC driver in Dom0 directly access NIC to transmit or receive packets through external network. Compared to Linux, there are two extra queues introduced by Dom0. One of them is between Linux Bridge and BE, the other one is used to connect FE and BE.

2.3 VMM Scheduler

The Xen functions as an abstraction layer of the real physical devices. As a result, scheduling in virtualization is based on Virtual CPUs (VCPU) because Physical CPUs (PCPU) are transparent to domains. Each domain can be arbitrarily allocated with multiple VCPUs. Besides the default credit scheduler, Xen also keeps its legacy scheduler Simple Earliest Deadline First (SEDF) [19]. SEDF provides weighted CPU sharing in an intuitive way and uses real-time algorithms to ensure real time guarantees. However, it lacks global load-balancing on multiprocessors and is becoming obsolete. In this paper we focus on the default credit scheduler [4], a proportional fair share CPU scheduler built to achieve load balance on SMP hosts. Its overall objective is to allocate the processor resources fairly. The scheduler organizes a local run queue of online runnable VCPUs for each PCPU and always picks a workload (VCPU) from the head of the queue to run. This queue is sorted by VCPU priority. A VCPU's priority can be one of three values: OVER, UNDER and BOOST. OVER, UNDER represent whether or not this VCPU has used up its fair share of CPU resource in the ongoing accounting period. The BOOST state provides a mechanism for domains to achieve low I/O response latency. All the VCPUs in BOOST state are placed in front of those in UNDER state in the runqueue, while those with OVER state are kept in the tail portion. Based on the predefined weight, each domain is initially allocated a corresponding credit which is fairly shared among all the VCPUs that are affinitized to the domain. As a VCPU runs, it consumes credits. Every so often, a system-wide accounting thread re-computes how many credits each active domain has earned and bumps the credits.

When it comes to multi-core architecture, there are a few twists while the scheduler functions. First of all, when there is not a VCPU of priority UNDER on a PCPU's local run queue, the scheduler will search other PCPUs for one. This load balancing ensures each domain receives its fair share of PCPU resources system-wide. Before a PCPU goes idle, the scheduler will look on other PCPUs to find any runable VCPU. This guarantees that no PCPU idles when there is runnable work in the system. Secondly, VCPU migration might happen based on priority difference for event notification. Whenever an event is notified to a target VCPU while it is idle, the scheduler tickles the designated PCPU and re-evaluates to see if the target VCPU preempts the current running VCPU. If there are at least two runnable VCPUs in that PCPU, the scheduler would migrate some of them to the idlers in the system to achieve load balance. Last but not the least, the scheduler checks the state of the current running VCPU during each timer interrupt and redistributes the PCPU if necessary. The running VCPU will be migrated to the online neighbor PCPU with the most idling neighbors PCPU. This policy distributes work across distinct sockets first and then distinct cores in the same socket.

3. EXPERIMENTAL TESTBED

Our experimental testbed consists of two Intel Clovertown machines which function as a system under test (SUT) and a stressor respectively. Each of them is a two-processor (or socket) platform based on the quad-core Intel Xeon processor 5300 series with 8 MB of L2 cache per processor [13]. Its CPU layout is illustrated in Figure 2. As we can see from the figure, each processor has two 4MB shared L2 caches. It is equipped with 16GB DRAM and 1333 MHz system bus. The SUT and stressor are hosted by Xen 3.1.3 and Vanilla Linux kernel respectively. Xen is configured with data copy mode and the credit scheduler is at default mode. To conduct virtualization experiments in 10GE network, we connect two machines via two Intel® PRO/10 GbE SR Server Adapters [12]. They connect to hosts through PCI-Express x8, a 16+16 Gigabit/s full-duplex I/O fabric that is fast enough to keep up with the 10+10 Gigabit/s full-duplex network port. Since the virtualized system has not supported TCP/IP offloading engine (TOE) and Jumbo Frames yet, all the experiments in this paper are conducted without TOE support and with an MTU of 1500 Bytes. We retain the default settings in network adapter's driver without specific performance tuning on interrupt coalescing, write combing etc. All protocol and system relevant settings are at default.

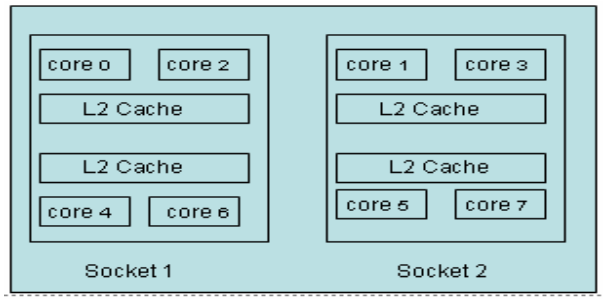

Figure 2. Intel Xeon Clovertown Machine

In all the experiments, Iperf [11] is selected as the micro-benchmark. The micro-benchmark allows us to quickly and easily identify the performance benefits of our optimizations. We run the experiments with eight parallel connections (the number of cores in our SUT) for various message sizes for measuring network bandwidth and the corresponding core utilization (The core utilization can reach up to 800% in our systems corresponding to two quad-core processors). In virtualized environment, the number of VCPUs in Dom0 is set to 8 (the number of cores in our SUT) and only one DomU is used for our experiments. The VMM scheduler is configured as credit scheduler by default, unless otherwise stated.

4. BASELINE ANALYSIS ON 10GE

In this section, we first conduct experiments by creating one DomU to evaluate the performance of baseline virtualized I/O in the SUT and then investigate the impact from Dom0 and VMM schedulers on virtualized I/O bandwidth.

4.1 Virtualization Performance Overhead

We conduct experiments with the default settings to study the virtualized I/O overhead under 10GE network. The result is illustrated in Figure 3, where bars and lines represent the bandwidth and core utilization respectively. The results of Dom0 is measured by running Iperf on a driver domain with a guest domain idling. The results of DomU are collected by running Iperf on a guest domain. The core utilization of DomU in the figure represents the sum of driver and guest domain's utilizations.

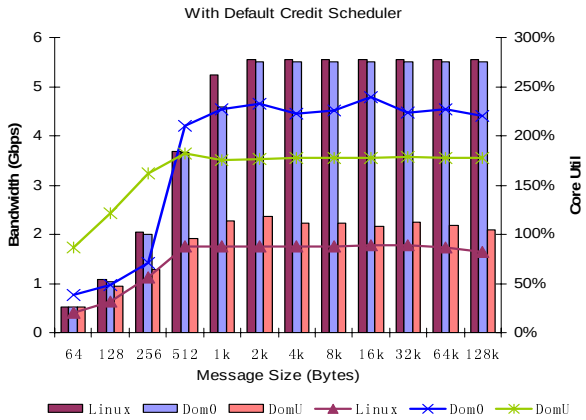

Figure 3. Bandwidth & Core Utilization

From the figure, some key observations are made as follows: 1) virtualized I/O (or DomU) can not achieve the same bandwidth as Linux even with higher core utilization. According to our results, virtualized networking suffers about 65% bandwidth degradation. We obtained only 2.3 Gbps of application bandwidth but 2X consumption of the core utilization compared to the non-virtualized case, 2) different from DomU, Dom0 can achieve the same bandwidth as Linux although with significantly higher core utilization than DomU. It is mainly because Dom0 directly manages hardware and network processing in Dom0 performs in a similar way to Linux. These observations point out that the current I/O split model mainly contributes to the performance loss and is not efficient enough to process packets. It obtains much lower bandwidth but with significantly higher core utilization than Linux. Since virtualization introduces an extra Dom0 to process packets and an extra VMM scheduler for scheduling domains, they are two major factors of I/O performance loss.

4.2 Domain's CPU Overhead

The overhead of Dom0 is unclear but critical for system designers to understand the behavior of processing packets. We study the breakdown of CPU overheads between Dom0 and DomU in virtualized I/O processing with various

message sizes, which are illustrated in Figure 4. Based on the figure, Dom0 consumes significantly more CPU cycles than DomU where network applications reside, especially for receiving small messages. It motivates us to optimize the overhead incurred in Dom0.

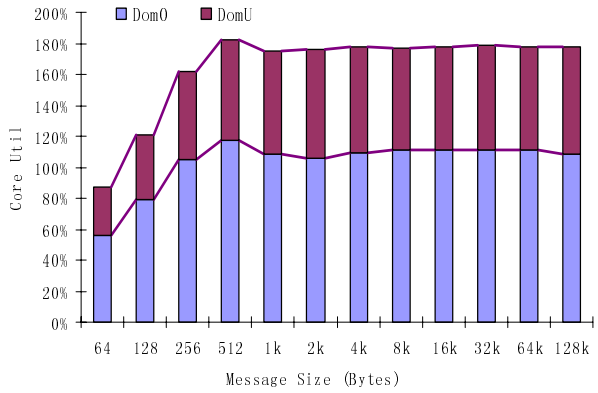

Figure 4. CPU Overhead Distribution

4.3 Impact of VMM Scheduler

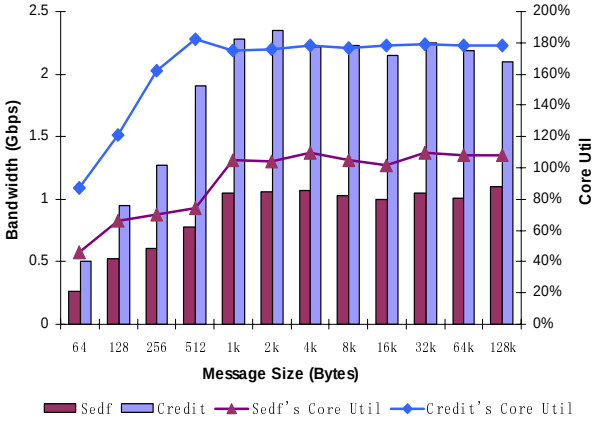

Figure 5. Impact of VMM Schedulers

Virtualized system introduces an extra scheduler inside the VMM for scheduling different domains. However, there is no single and easy method for quantifying the impact of VMM schedulers on network performance. Since Xen is a unique virtualized system to provide multiple VMM schedulers like SEDF and the default credit scheduler, we carry out a comparative evaluation of them on multi-core platforms to project VMM scheduler's impact. The experimental results including bandwidth and core utilization are illustrated in Figure 5. It shows that SEDF can not utilize the multiple cores and only achieves 1.1Gbps bandwidth, mainly because it lacks global load-balancing on multi-core platforms. This explains why SEDF has been replaced by credit scheduler and is becoming obsolete on multi-core platforms [4]. The credit scheduler performs better than SEDF in Fig. 5, but still is unable to get high bandwidth for I/O intensive applications. All of them point out that the VMM scheduler can drastically impact the performance of I/O intensive applications running on domains, and motivate us to revisit its design and possible optimization.

5. OPTIMIZATIONS OF SCHEDULER

The default credit scheduler, a proportional fair share CPU scheduler, is designed for load-balancing workloads on multi-core platforms. However, it is unaware of core topology in multi-core systems, where some of cores are sharing a last level cache while others are sitting in different sockets. It blindly migrates the VCPU running on PCPU with high workloads to PCPU with lightweight workloads. Moreover, the number of VCPUs in Dom0 is configured to the same number of PCPUs, and each VCPU is assigned with the same credit even though some of them are idling. All of these behaviors significantly degrade virtualized network performance. In this section, we develop two optimizations: a cache-aware scheduler for reducing inter-domain communication cost and credit-stealing for favoring I/O VCPU in Dom0. In order to compare the efficiency of packet processing across our optimizations, the metric of core utilization is normalized to per gigabit in the following sections.

5.1 Cache-Aware Scheduler

To make the best use of the resources and to make inter-core communication efficient, cores in a physical package share some of the resources. Our SUT has two CPU cores sharing the L2 cache which is called Intel Advanced Smart Cache [13], as shown in Figure 2. Each processor has four cores in a physical package with two last level (L2) caches. Each L2 cache is shared by two cores. Further, as more logic gets integrated into the processor package, more resources will be shared by the cores on the die. The current credit scheduler in virtual machine monitor is designed for SMP load balance, but is not cache-aware and can not co-schedule the two VCPUs with data sharing on the two cores sharing L2 cache (a.k.a. cache domain). Since Dom0 is designed for serving I/O requests to de-multiplex packets and move packets to designated DomU (I/O DomU), there is intense data sharing between Dom0 and I/O DomU, especially when the data copy mode is configured. Co-scheduling Dom0 and I/O DomU in the cache domain will give I/O DomU a free ride to access the data in the cache and then result in more efficient inter-domain communication.

Implementation:

In order to co-schedule Dom0 and I/O DomU, the first step is to identify them in the VMM. Currently we identify them by counting how often I/O events of boosting VCPUs are triggered during each time slice. If the number of triggers exceeds a threshold (default 150), both the boosting and the

boosted VCPUs are considered as I/O VCPUs (in receive side, boosting VCPU is I/O VCPU in Dom0). Note that our extension of the scheduler is only based on VCPUs with intense I/O operations, and doesn't sacrifice the system-wide load-balance on multi-core platforms. After the identification of I/O VCPUs, the VMM scheduler always intelligently schedules boosting and boosted VCPUs to the cores sharing same L2 cache.

In default credit scheduler, when an event is notified to a target VCPU while it is idle, it is awaken with the state of BOOST. Then other idle PCPUs and PCPU hosting the VCPU are tickled to re-evaluate where the VCPU will be running. In cache-aware scheduler, instead of tickling all idle PCPUs, boosted VCPU is inserted into the runqueue of PCPU sharing L2 cache with the PCPU currently hosting boosting VCPU. An example is shown in Figure 6. The left side in the figure is the original system state where boosting VCPU and one running VCPU are sitting in the same cache domain and boosted VCPU is running on the core 4. Cache-aware scheduler will automatically migrate boosted VCPU into the same cache domain as boosting VCPU to take advantage of shared cache. The running VCPU is preempted into the core 4 for securing the system level load balance. The system state after migration is shown in the right side of the figure.

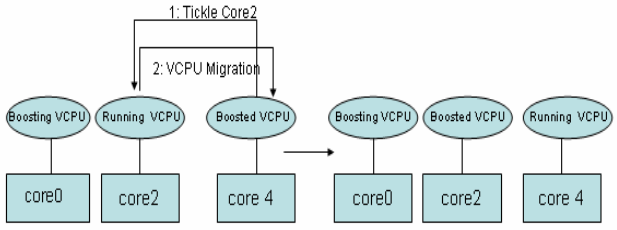

Figure 6. An Example of Cache-Aware Scheduler

Additionally, VCPU migration in current scheduler also occurs when a VCPU remains BOOST for a while and some PCPUs are idle. It chooses the target PCPU with the largest number of idle neighbors in its grouping. This option will distribute workload across distinct packages first and result in maximum resource utilization since there is no shared resource contention. However, virtualized I/O processing with data sharing between Dom0 and I/O DomU will suffer heavy inter-package communication penalty from this mechanism. Cache-aware scheduler dynamically migrates the boosted VCPU and boosting VCPU to the same cache domain when this migration is triggered.

Although our technique might preempt the running VCPU on the PCPU, the preempted VCPU could be migrated into other PCPUs to sustain system-level workload balance on multi-core platforms.

Experiments:

We conducted experiments to study the benefits of our cache-aware scheduler over the default version in terms of both bandwidth and core utilization per gigabit as a function of message sizes. The result is depicted in Figure 7, where "*Default*" represents the conventional credit scheduler and "*Opt A*" is for our cache-aware scheduler. Our extended scheduler increases the bandwidth by 13%, and also saves 11% in core utilization per gigabit when message size is greater than 1 KBytes. With smaller messages, a slightly higher bandwidth is achieved with lower core utilization. The results reveal that our cache-aware scheduler reduces the overhead of inter-domain communication in virtualized packet processing.

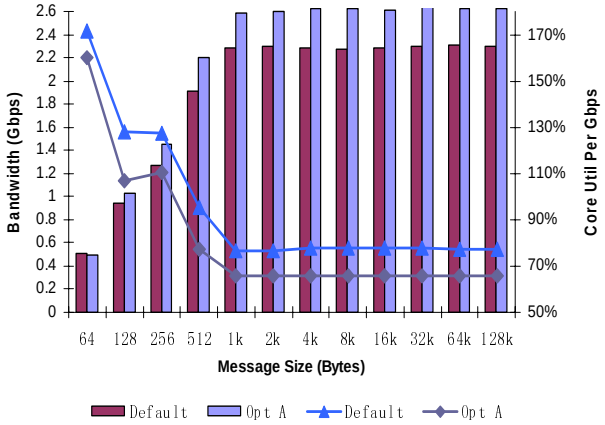

Figure 7. Cache-Aware Scheduler

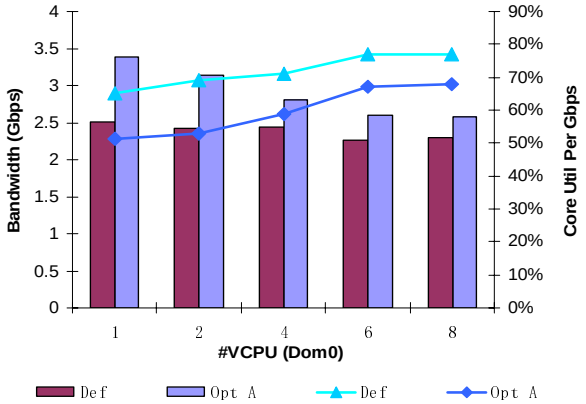

Figure 8. Impact of the Number of VCPUs in Dom0

We also study the impact of the number of VCPUs in Dom0 on virtualized network bandwidth with and without cache-aware scheduler in Figure 8. The 64 KBytes message is used in our experiments. The figure shows that both the schedulers perform better with less VCPUs in Dom0. It is also observed that our cache-aware scheduler increases the bandwidth by 35% with the savings of 21% in core utilization per gigabit when the number of VCPUs is 1. However, the benefits over default scheduler reduce as the

number of VCPUs increases from 1 to 8. All of the above observations point out that the number of VCPUs in Dom0 affects I/O's behavior and has a greater negative impact with cache-aware scheduler.

5.2 Credit-Stealing for I/O VCPU in Dom0

The number of VCPUs in Dom0 is configured by default as the number of cores in the platform. In credit scheduler, all VCPUs affiliated to the same domain are allocated fairly with the same credit. However, all of the interrupts from NIC are usually directly directed to a specific VCPU to improve the cache locality of interrupt processing in a non-virtualized environment. This credit allocation mechanism results in performance degradation in virtualized environment mainly because more VCPUs in Dom0 lead to less shared credit for each VCPU. I/O VCPU can not be allocated with sufficient computing resources to satisfy packet processing. We propose to dynamically and temporarily steal some credits from other idle VCPUs to favor I/O VCPUs during each time slice while I/O VCPUs are busy with processing packets. The principle to steal credits is formalized in the following equation:

$$Steal = Credit(Idle\_VCPUs)/(2 * Num(IO\_VCPUs))$$

where $Steal$ means the stolen credit for each I/O VCPU, $Credit(Idle\_VCPUs)$ is for the credit of all idle VCPUS. $Num(IO\_VCPUs)$ represents the number of I/O VCPUs. It shows that each idle VCPU's credit is dynamically cut in half to favor I/O VCPUs to eliminate their burden while working with intensive NIC interrupt requests. Since our policy steals credits from idle VCPUs, it does not hurt the overall system performance.

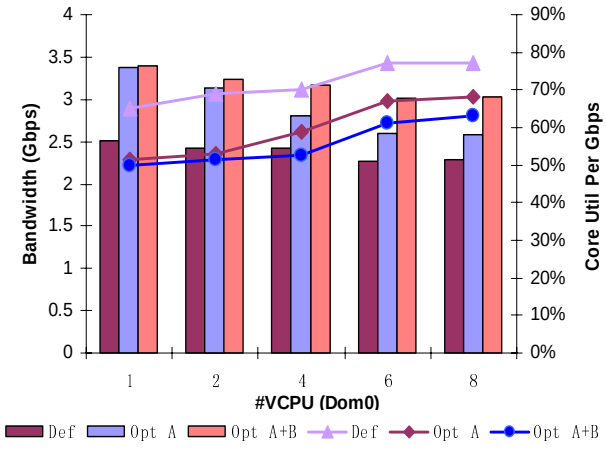

Figure 9. Credit-Stealing for I/O VCPU in Dom0

Experiments:

We used the 64 KBytes messages to study the network bandwidth achieved by credit-stealing along the number of VCPUs in Dom0. The experimental results are shown in Figure 9. Similar to our previous figures, the normalized core utilization is also illustrated by lines. "*Def*" represents the default credit scheduler and "*Opt A*" is for our extended cache-aware scheduler. "*Opt A+B*" is the combinational scheduler with both two optimizations. As seen from the figure, the policy for stealing credit further improves the network performance by 18% and saves 5% in core utilization per gigabit when the number of VCPUs is greater than 4. Without any knowledge from users or administrators, it will dynamically adapt to favor I/O processing while receiving intensive I/O requests and then lead to a higher bandwidth by wasting less computing resources. With the combination of cache-aware scheduler, our extended scheduler increases performance by 31% and saves 15% in core utilization per gigabit with 8 VCPUs.

Since our two optimizations are software extensions of the VMM scheduler, they are generic to improve network performance in a hardware independent manner, and are applicable to both 1GE and 10GE network.

6. OPTIMIZATONS IN DOM0

This section is to study the overhead incurred in Dom0 with 10GE network. In order to analyze the component level overhead along packet processing, we develop our own tool based on Intel Performance Counter [15]. We instrument the VMM and Dom0 along with the path to record the current time-stamp and retired instructions at major points. Our experiment indicates that Linux Bridge and I/O channel are two major bottlenecks during packet processing.

Component	Cycles	Instructions
Linux Bridge	**11K**	**4K**
I/O Channel (Data Copy Model)	8.5K	1.5K

Table 1. Overhead of Important Components

As shown in Table 1, Linux Bridge executes 4K instructions and consumes about 11K cycles to switch one packet. It is mainly because Linux Bridge must comply with the Netfilter interface which is used to integrate filter rules to filter received/transmitted packets. The I/O Channel using data copy model is another big overhead. It is mainly contributed to the expensive entry/exit to the VMM and the real packet copy between domains. All of them are mandatory in the current I/O model. We also notice that there are some packet losses within TCP streams resulting in TCP retransmissions. In subsection 6.1, we design a simplified bridge to improve packet switching, followed by the configurable TX queue length of backend (BE) engine (Fig. 1) for ameliorating packet retransmissions in subsection 6.2.

6.1 Packet-Switching Optimization

Linux Bridge is a way to connect two Ethernet segments together in a protocol independent way. Packets are forwarded based on Ethernet address, rather than IP address (like a router). The Linux bridge code implements a subset of the ANSI/IEEE 802.1d standard. The code for bridging has been integrated into 2.4 and 2.6 kernel series [20]. In order to simplify the VMM design, Xen takes advantage of the existing Linux Bridge component in Linux Kernel to serve as an arbitrator/multiplexer/de-multiplexer. Linux Bridge is used to switch packets to designated ports. Although this approach can greatly simplify the virtualization design, it adds extra high overhead while switching packets.

Fortunately, our function level profiling of Linux Bridge indicates that pure packet switching function requires only 600 cycles, and Jhash algorithm [18] used for multiplexing packets by hashing MAC addresses only consumes 120 cycles. It motivates us to design a simplified bridge tailored for packet switching in virtualized environment. However, it must retain the same user/kernel interface as original bridge so that the user space bridge utility still works in virtualization environment. Since bridge utilities in user space are being used by domain management tool residing in Dom0 to create/destroy BE, the new bridge should comply with the original user/kernel interface to avoid interference with the current workable system. The new design is required to keep bridge as simple as possible with respect to packet switching's performance and scalability.

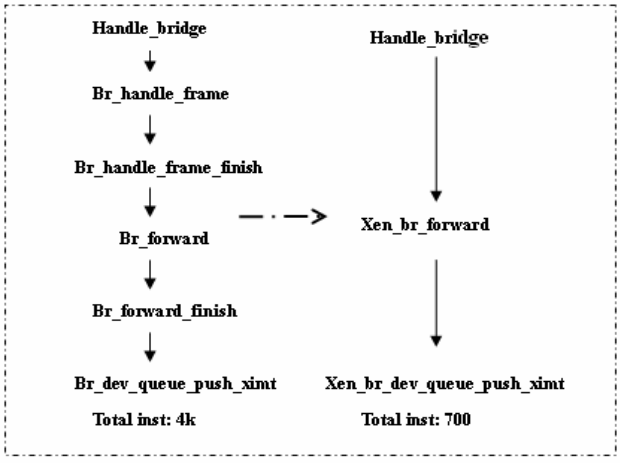

Figure 10. Linux Bridge Vs Our Tailored Bridge

Packet processing path of both Linux Bridge and our tailored bridge are shown in Figure 10. It shows that we bypass most of the functions introduced by Netfilter interface and re-implement the internal interfaces to minimize extra function costs except the bridge itself. The Jhash algorithm is still adopted in our design. Our prototype is implemented as a new feature of Linux Bridge to take advantage of its existence in mainstream kernel. We study its packet switching overhead and find that executed instructions are reduced from 4K to 700, and the switching packet cycles are reduced significantly by 10X from 11K to 1.2K.

We also evaluate our bridge by running the whole session Iperf to study its impact on bandwidth and core utilization, which is shown in Figure 11. "*Opt C*" in the figure is for our simplified bridge. It is observed that our simple bridge prototype can increase the bandwidth by 7% and save 10% in core utilization per gigabit when message size is greater than 1 KBytes. With small messages, a slightly higher bandwidth can be achieved by our bridge with the savings of 13% in core utilization per gigabit. All the results reveal that our simple bridge is a helpful software approach to optimize I/O processing. Similar to the two optimizations for the VMM scheduler, our simplified bridge is also a generic approach and is applicable to both 1GE and 10GE network.

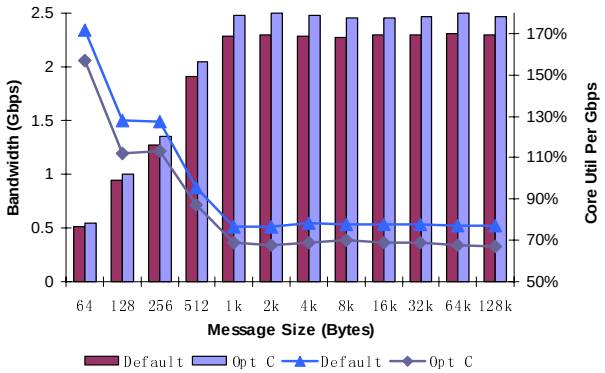

Figure 11. Simple Bridge Optimization

6.2 Configurable TX Queue Length of BE

Since TCP is a network protocol providing reliable in-order delivery of a stream of bytes, frequent data loss incurring packet retransmissions would result in the significant performance degradation. It is important to understand the TCP behavior while processing packets in virtualized environment. In our experiments, we notice that there are some losses of data within TCP streams, resulting in TCP retransmissions and therefore creating gaps during data transfer. We study the data loss by a detailed analysis with the tool tcpdump [31] and observe some transmission gaps of greater than 60 *ms* caused by TCP retransmissions that do not happen in Linux.

By looking into the source code for packet processing, we found that two extra queues are introduced in virtualized I/O: an outgoing queue of BE to queue frames from Linux Bridge and a RX queue in FE of DomU to host received packets from BE. The maximum number of frames that can be queued on the BE's outgoing queue is statically fixed to

32 (It is the trade-off point between bandwidth and latency. A longer queue introduces a higher processing latency and hence might offset the gains of bandwidth from the longer queue). Although this value works well under 1GE network, a 10X in network speed needs a larger queue to avoid packets being dropped. We develop a patch to make this parameter configurable during run-time to users and study its impact on bandwidth. The result is illustrated in Figure 12 when the queue length is varied value from 32 to 512. We notice that 128 is the best value under 10GE network. It achieves 3.3Gbps bandwidth and saves the corresponding core utilization. Compared to the default system, the technique increases bandwidth by 43% and saves 5% in core utilization per gigabit. Additionally, we also study the impact of RX queue's length in FE and find that the current value 256 works well and does not trigger any data loss.

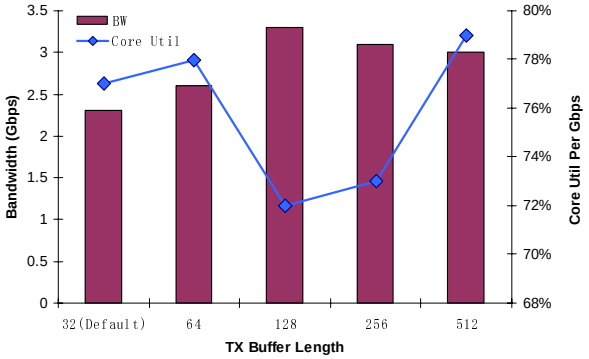

Figure 12. Configruable TX Queue Len

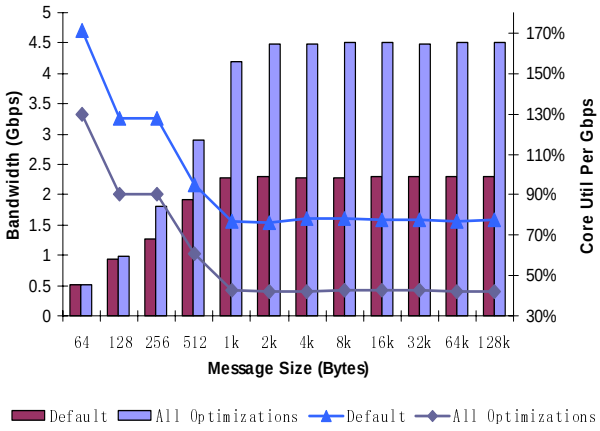

Figure 13. Combination of All Techniques

7. COMBINATION OF OPTIMIZATIONS

Two extensions for the VMM scheduler and two optimizations for Dom0 were proposed and evaluated by individually in the previous sections. This section evaluates the combined effect of all the above optimizations in terms of network bandwidth and core utilization per gigabit. We incorporate all the optimizations into virtualized system and compare the results with the default system for varied message sizes, which is illustrated in Figure 13. The bars represent the bandwidth and the lines represent the core utilization per gigabit. It is observed that our combined techniques can increase the network bandwidth by 96% to 4.5 Gbps, and also save 36% in core utilization per gigabit when the message size is greater than 1 KBytes. With small messages, a slightly higher bandwidth is achieved but 20% core utilization is saved. In our experiment, we also observed that the total core utilization consumed by Dom0 is reduced from 105% to 84% by using all the optimizations.

8. RELATED WORK

With the advent of multi-core architecture and high-speed networks such as 10GE, G.N *et al.* [26] studied the network stack's interaction with applications in multi-core environments. However, they just focused on some high performance network stacks and did not investigate the virtualized I/O with a multi-core server under 10GE network. With respect to virtualization overhead analysis, several previous papers [22, 23, 24] measured the impact of virtualization overhead on micro or macro benchmark, but they did not consider the impact of the VMM scheduler, which deteriorates on 10GE network. D.O *et al.* [7] studied the relationship between the VMM scheduler and I/O performance. However, they focused on the fairness of I/O performance with 1GE network and lacked of the consideration of the VMM scheduler on mainstream multi-core systems where behaves significantly different from uni-core systems.

Since some useful features from virtualization including ease of management, functional isolation and live migration can be very beneficial to the manageability of high performance applications: Liu *et al.* [21] adopted virtualization technology for HPC and allowed each domain to directly access the high performance network. However, they targeted to the high performance network InfiniBand rather than the Gigabit Ethernet Network. In Ethernet Network, some researches including NetXen [25] and Crossbow [6] are trying to address the performance issues by taking advantage of the new Ethernet NIC features like multiple TX/RX queues to allow domains to directly access the hardware. However, they are designed to work with the new 10GE network adapters and hence inhibit the features of portability and live migration, which are the two major incentives for deploying virtualization in high end servers. Similarly, hardware-assisted direct I/O technology [16] is proposed for full virtualized system to allow guest OS to directly access PCI devices. However, it also heavily relies on hardware and full virtualization itself has much higher overhead than para-virtualized system. Software support is also far immature. Receive Side Scaling [29], a technique in NIC for mapping each connection to a specific core instead of being processed by random idlers to improve network

processing scalability, can not also be directly applied to virtualized environment due to an extra mapping of virtual CPU to physical CPU. Our software optimizations work in a hardware independent manner and are applicable to both 1GE and 10GE network.

9. CONCLUSION AND FUTURE WORK

This paper analyzes the performance of virtualization in 10GE network with a multi-core server. We found that virtualization under 10GE network adds significant performance overhead to network packet processing. We proposed two optimizations for the scheduler inside the VMM to eliminate the scheduler's impact on performance, based on cache locality and credit stealing. We also proposed two techniques for Dom0 to accelerate packet processing by re-designing a simplified bridge tailored for switching packets and adjusting the queue length of backend to minimize the data loss. Our combined optimizations not only dramatically increase bandwidth by 96% but also significantly improve the efficiency with a savings of 36% in core utilization per gigabit. All the optimizations work in a hardware independent manner and are applicable to both 1GE and 10GE network. Using these optimizations, VMM designers can understand how to better serve the virtualized I/O processing in multi-core systems.

In future, we would like to study the impact of architectural features on virtualized network performance such as Direct Cache Access (DCA) [10] of directly pushing data into cache, and exploit some micro-architectural changes to further optimize packet processing performance for 10GE network. All the patches for the VMM scheduler, Linux Bridge and configurable TX queue length are under code review and will be released to public soon.

10. ACKNOWLEDGEMENTS

This work is supported in part by Intel research grant.

11. REFERENCES

[1] AMD64 Virtualization "Pacifica" Technology, Secure Virtual Machine Architecture Reference Manual, May 2005.
[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In 19th SOSP, Oct 2003.
[3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. K. Su. Myrinet: A Gigabit-per-Second Local Area Network. IEEE Micro '95.
[4]Credit scheduler. http://xen.org/files/summit_3/sched.pdf.
[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines, OSDI.
[6] Crossbow. http://opensolaris.org/os/project/crossbow/.
[7] D, Ongaro., A. L, Cox., S, Rixne. 2008. Scheduling I/O in virtual machine monitors. VEE 2008.
[8] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the Xen virtual machine monitor. In 1st OASIS, Oct 2004.
[9] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda. Performance Characterization of a 10-Gigabit Ethernet TOE. HotI, 2005.
[10] Ram Huggahalli, Ravi Iyer, Scott Tetrick. "Direct cache access for high bandwidth network I/O", 32nd International Symposium on Computer Architecture, 2005. ISCA'05 Proceedings.
[11] Iperfbenchmark. http://dast.nlanr.net/Projects/Iperf/.
[12] Intel 10 Gigabit Ethernet Controllers http://download.intel.com/design/network/prodbrf/317796.pdf.
[13] Intel Core 2 Extreme quad-core processor. http://www.intel.com/products/processor/core2XE/.
[14] Intel Virtualization Technology Specification for the IA-32 Intel Architecture, April 2005.
[15] Intel Software Developer's Manual Vol.
3B. http://www.intel.com/products/processor/manuals/.
[16] Intel VT-d.
http://www.intel.com/technology/itj/2006/v10i3/2-io/5-platform-hardware-support.htm.
[17] Infiniband. http://www.infinibandta.org.
[18] Jhash. http://www.burtleburtle.net/bob/hash/doobs.html.
[19] I. M. Leslie, D. Mcauley, R. Black, T. Roscoe, P. T. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. IEEE .
[20] Linux Bridge. http://bridge.sourceforge.net/.
[21] J. Liu, W. Huang, B. Abali and DK Panda, "High Performance VMM-Bypass I/O in Virtual Machines", USENIX Annual Technical Conference, June 2006.
[22] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing Performance overheads in the Xen Virtual Machine Environment, VEE'05.
[23] A.Menon, J. R. Santos, Y. Turner, and G. Janakiraman, "Xenoprof - Performance profiling in Xen".
[24] A. Menon, A. Cox, W. Zwaenepoel, Optimizing Network Virtualization in Xen, 2006 USENIX Annual Technical Conference.
[25] Netxen, http://www.netxen.com/index1.html.
[26] G.Narayanaswamy, P.Balaji, W.Feng, An Analysis of 10-Gigabit Ethernet Protocol Stacks in Multicore Environments in HotI'07.
[27] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. TheQuadrics Network (QsNet): High-Performance Clustering Technology.In HotI '01.
[28] M. Rosemblum and T. Garfinkel. Virtual Machine Monitors: Current Technology and Future trends. IEEE computer, 38(5): 39-47, 2005.
[29] Scalable Networking: Eliminating the Receive Processing Bottleneck. Microsoft WinHEC April 2004.
[30] Top500 supercomputer list. http://www.top500.org.
[31] Tcpdump. http://www.tcpdump.org.