Redesigning Xen's Memory Sharing Mechanism for Safe and Efficient I/O Virtualization

Kaushik Kumar Ram, Jose Renato Santos, Yoshio Turner

HP Laboratories
HPL-2010-39

Keyword(s):
No keywords available.

Abstract:
Xen's memory sharing mechanism, called the grant mechanism, is used to share I/O buffers in guest domains' memory with a driver domain. Previous studies have identified the grant mechanism as a significant source of network I/O overhead in Xen. This paper describes a redesigned grant mechanism to significantly reduce the associated overheads. Unlike the original grant mechanism, the new mechanism allows guest domains to unilaterally issue and revoke grants. As a result, the new mechanism makes it simple for the guest OS to reduce the number of grant issue and revoke operations that are needed for I/O by taking advantage of temporal and/or spatial locality in its use of I/O buffers. Another benefit of the new mechanism is that it provides a unified interface for memory sharing, whether between guest and driver domains, or between guest domains and I/O devices using IOMMU hardware. We have developed an implementation of the new grant mechanism that fully supports driver domains, but not yet IOMMUs. The paper presents performance results using this implementation which show that the new mechanism reduces per-packet overhead by up to 31% and increases network throughput by up to 52%.

External Posting Date: March 21, 2010 [Fulltext] Approved for External Publication
Internal Posting Date: March 21, 2010 [Fulltext]

Published and presented at the Second Workshop on I/O Virtualization (WIOV '10), March 13, 2010, Pittsburgh, PA, USA.

© Copyright The Second Workshop on I/O Virtualization (WIOV '10), 2010.

Redesigning Xen's Memory Sharing Mechanism for Safe and Efficient I/O Virtualization

Kaushik Kumar Ram
Rice University
kaushik@rice.edu

Jose Renato Santos
HP Labs
joserenato.santos@hp.com

Yoshio Turner
HP Labs
yoshio.turner@hp.com

ABSTRACT

Xen's memory sharing mechanism, called the grant mechanism, is used to share I/O buffers in guest domains' memory with a driver domain. Previous studies have identified the grant mechanism as a significant source of network I/O overhead in Xen. This paper describes a redesigned grant mechanism to significantly reduce the associated overheads. Unlike the original grant mechanism, the new mechanism allows guest domains to unilaterally issue and revoke grants. As a result, the new mechanism makes it simple for the guest OS to reduce the number of grant issue and revoke operations that are needed for I/O by taking advantage of temporal and/or spatial locality in its use of I/O buffers. Another benefit of the new mechanism is that it provides a unified interface for memory sharing, whether between guest and driver domains, or between guest domains and I/O devices using IOMMU hardware. We have developed an implementation of the new grant mechanism that fully supports driver domains, but not yet IOMMUs. The paper presents performance results using this implementation which show that the new mechanism reduces per-packet overhead by up to 31% and increases network throughput by up to 52%.

1. INTRODUCTION

The *Grant Mechanism* in the Xen virtualization platform provides controlled memory sharing between virtual machines (domains) [4, 8]. The grant mechanism allows a source domain to control which of its memory pages can be accessed by a specified destination domain. In addition, it allows the destination domain to validate that the shared memory pages belong to the source domain.

A primary use of the grant mechanism is to share I/O buffers in guest domains' memory with a driver domain. The driver domain is a dedicated virtual machine which hosts physical device drivers and performs I/O on behalf of the guest domains. For example, for network I/O the driver domain needs write access to I/O buffers in the guest domains' memory to copy the contents of arriving packets that are destined for the guest domain. For packets that are transmitted by the guest domain, the driver domain needs read access to guest domain I/O buffers to read packet headers and determine where to route the packets.

Previous work [15, 17] has shown that the grant mechanism incurs significant overhead when performing network I/O, and has also shown that most of this overhead is incurred in the driver domain. This is mostly due to the overheads of grant hypercalls and of the high cost of page mapping/unmapping operations executed in these hypercalls.

This paper presents a new grant reuse scheme whereby the guest domain can greatly reduce the number of grant issue and revoke operations that are needed for I/O by taking advantage of temporal and/or spatial locality in its utilization of I/O buffers. The guest domain can issue a grant for a page containing I/O buffers, then use the page several times for I/O, and finally revoke access to that page. In contrast, in the existing implementation every I/O involves grant issue and revoke operations. Consequently, the grant reuse scheme reduces the number of grant hypercalls and page mapping/unmapping operations needed for I/O.

To support the grant reuse scheme, this paper introduces a new grant mechanism which replaces the existing grant mechanism in Xen. Whereas the existing grant mechanism requires the guest domains to coordinate with the driver domain to revoke a grant, the key idea of the new grant mechanism is to enable the guest domains to unilaterally issue and revoke grants, except in error conditions. By breaking this dependency, the new grant mechanism avoids the need for a handshake protocol between the guest and driver domains in order to revoke the grant to a page, as would be needed with the existing grant mechanism. More generally, using the new grant mechanism to control memory sharing between two arbitrary guests has the advantage that each guest can stop sharing its pages with its peer at any time. In particular, each guest domain can forcibly revoke its grants in case its peer misbehaves.

This paper additionally shows that the new grant mechanism provides a unified interface that can extend the control of memory sharing to I/O devices using new IOMMU hardware. This allows guest domains to use the new grant mechanism to protect a guest domain's memory from incorrect device DMA operations, for both pass-through device access (i.e., direct I/O) [11, 13, 19] and when using an intermediary driver domain [8].

While parts of the new grant mechanism are similar to the partial mechanism described in our previous work [15], this paper contributes key enhancements that complete the mechanism – in particular, to support network transmit in addi-

This paper appeared at the Second Workshop on I/O Virtualization (WIOV '10), March 13, 2010, Pittsburgh, PA, USA.

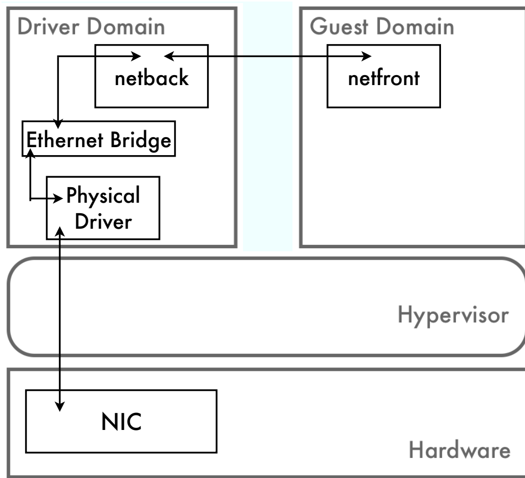

Figure 1: Xen's driver domain-based network I/O virtualization architecture

tion to receive, and to support mapping buffers into the driver domain address space.

Finally, this paper presents an experimental performance evaluation of a full implementation of the new grant mechanism on modern hardware (but without IOMMU support as yet). The evaluation demonstrates that the new mechanism reduces per-packet overhead by up to 31% and increases the network throughput by up to 52%. While this paper only explores the use of the new grant mechanism for network I/O, we believe that it can completely replace the existing mechanism in Xen. The new mechanism is no less general than the existing mechanism in Xen, and the ability to unilaterally revoke grants provides greater robustness against non-cooperative peers.

The rest of the paper is organized as follows. Section 2 reviews necessary background information on Xen's I/O architecture, the existing grant mechanism, and IOMMUs. Section 3 presents the design of our new grant mechanism. Section 4 presents an experimental evaluation of the performance of the new grant mechanism. Section 5 presents related work, and Section 6 presents conclusions.

2. BACKGROUND

2.1 Xen Grant Mechanism and I/O Architecture

Figure 1 shows Xen's driver domain-based network I/O architecture. The driver domain needs to have read access to I/O buffers in the guest domain's memory in order to read packet headers for transmitted packets, and it needs write access to copy received packets to the guest domain. To provide this access, the grant mechanism is used to share I/O buffers in the guest domain's memory with the driver domain. The grant mechanism allows the driver domain to access guest I/O buffers in a safe manner and the guest domain to limit the memory pages shared with the driver domain to only those containing buffers being used for I/O operations.

The grant mechanism interface is illustrated in Figure 2. A

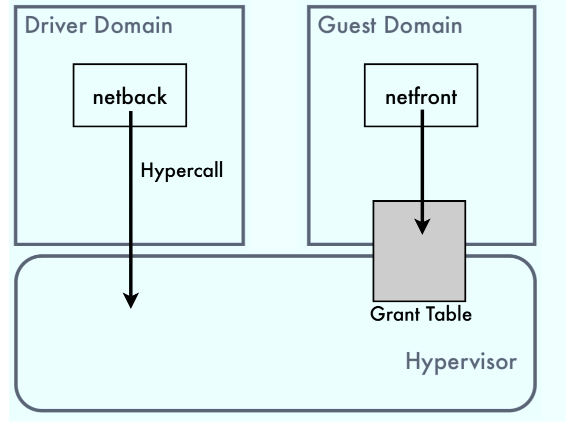

Figure 2: Existing Grant Mechanism interface in Xen

domain shares one of its memory pages with another domain in two stages. In the first stage, the source domain allocates a *grant reference* for the memory page it desires to share. The grant reference points to an unique entry in a *grant table* which is shared between the guest domain and the hypervisor. The grant entry contains the shared memory page address, the destination domain id, and the access permissions. The guest domain fills the grant table entry using simple memory writes, and then passes the grant reference to the destination domain. In the second stage, the destination domain uses the grant reference to access the shared memory. This stage requires hypervisor intervention. The destination domain invokes a hypercall, to enter the hypervisor, passing the grant reference as an argument. The hypervisor first checks whether the grant reference is valid. Then it reads the source domain's grant table entry and checks whether the domain that invoked the hypercall is the intended destination domain and also whether the shared memory page is owned by the source domain. If these tests pass then the hypervisor pins the memory page and finally maps the page within the destination domain address space. *Page pinning* ensures that the page ownership does not change while the destination domain has access to the page. Now the destination domain can safely access the shared memory page.

To stop sharing a page, the destination domain issues another hypercall. The hypervisor first unmaps the page from the destination domain address space and then unpins the shared memory page. Subsequently, the source domain can revoke the grant reference for the memory page by invalidating the corresponding grant table entry (again, through simple memory write operations).

In the standard network I/O model in Xen, the guest domain creates grants for its receive/transmit buffers to provide shared access to the driver domain. The driver domain then issues the grant hypercalls to map the guest pages in its address space. Once the I/O has completed, the driver domain again issues hypercalls to unmap the guest pages[1]. Then the driver domain notifies the guest domain that the

[1]Typically, receiving a network packet involves only a single hypercall. The hypervisor then maps the guest page, copies the packet, and finally unmaps the page.

I/O operations have completed. This also serves as a notification that the guest domain can revoke the grant for the corresponding page. Thus, a grant is issued and revoked for each and every I/O operation, leading to significant performance overhead for memory sharing.

2.2 IOMMU Overview

I/O Memory Management Units (IOMMUs) provide address translation for I/O devices [2, 3]. All memory accesses from devices, through DMA, undergo address translation using an I/O Page Table (or IOMMU Table). A DMA operation fails if a valid translation does not exist in the I/O page table. Thus, IOMMUs can protect against incorrect or malicious memory accesses from I/O devices.

In virtualized systems, an IOMMU is particularly needed when guest domains have direct access to devices (pass-through devices), in order to restrict the device to access only the memory of the guest domain to which it is assigned. Each device has its own IOMMU Table, which can be configured with valid translations for exactly all the pages that belong to the guest domain accessing that device. In this mode of operation, which we refer to as coarse grain protection, the IOMMU table is mostly static and does not change unless the set of pages assigned to the guest domain changes. To provide a higher level of protection against buggy device drivers or to enable user-level drivers, the set of valid IOMMU translations can be limited to only a small set of pages which contain I/O buffers that need to be accessed by the device. In this mode, which we refer to as fine grain protection, the guest domain needs to invoke the hypervisor to add pages to the IOMMU table before programming a device DMA operation, and to remove the page translations after the operation completes [5, 6].

3. REDESIGNING THE GRANT MECHANISM

A grant reuse scheme can significantly reduce grant overheads by reusing the same grant for multiple I/O operations. A guest domain can issue a grant for a page containing I/O buffers, then use the page several times for I/O, and finally revoke access to the page. Thus, under the reuse scheme the overhead of the grant hypercalls and the mapping/unmapping operations is not incurred on every I/O operation.

For the grant reuse scheme, a source domain should be able to revoke a grant when the corresponding memory buffer is re-purposed. As an example, suppose a guest OS shares a granted page with a driver domain for network I/O, and then later the guest OS re-purposes the page, for instance to assign the page to a user-level process. Before re-purposing the page, the guest OS should be able to revoke the grant in order to prevent subsequent access to the page by the driver domain. In general, a source domain needs the flexibility to revoke a grant from within various OS subsystems running in the source domain. Using the existing grant mechanism in Xen, this would require the source domain to carry out a protocol handshake with the destination domain via an inter-domain I/O channel in order to revoke the grant. This handshake protocol prevents the source domain from completing the grant revocation until the destination domain unmaps the page and notifies the source domain that the unmapping is complete.

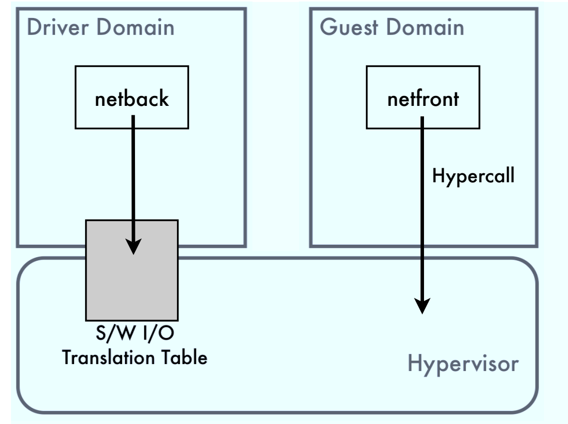

Figure 3: New Grant Mechanism interface in Xen

This paper introduces a new grant mechanism which breaks that dependency by allowing the source domain to unilaterally issue and revoke grants, except in error conditions. For I/O, this means that the guest domain can issue and revoke grants using a simple hypercall interface without requiring driver domain participation. This avoids the handshake protocol completely. Moreover, the new grant mechanism has the benefit of reducing the trust required between any two domains that are sharing memory. Either guest can unilaterally remove access privileges to its pages from the other guest without requiring the cooperation of the other guest to unmap the pages. This is particularly useful in case the other guest misbehaves.

It turns out that the guest interface to the new grant mechanism is very similar to an interface needed to add and remove pages from an IOMMU table when using the fine grain protection mode. Thus, the grant interface can be used to issue/revoke grants or add/remove entries to/from an IOMMU table. This unification simplifies I/O support in the guest OS, to share memory either with driver domains or with devices directly. For example, for guest domains running Linux this can be supported by a common implementation of the DMA API interface [5, 6].

3.1 New Grant Mechanism

The new grant mechanism is illustrated in Figure 3. In the new mechanism, the guest domain directly interacts with the hypervisor, via hypercalls, to issue and revoke grants. The hypervisor exposes to the driver domain the mapping of grant references to memory pages using a shared region of memory containing a *software I/O translation table*. This table is required only when guest domain pages are shared with a driver domain for device I/O. The table is unnecessary when sharing memory without involving an I/O device, or when a hardware IOMMU is used to provide address translations for I/O devices (Section 3.4).

3.1.1 Initialization

At initialization the guest domain issues a hypercall to create a new I/O translation table passing as arguments: the ID of the driver domain, and the size of the I/O translation table.

This hypercall returns a handle to the new I/O translation table which the guest domain communicates to the driver domain. The driver domain has one such translation table for each guest domain it supports.

After receiving the translation table handle from the guest domain, the driver domain issues a hypercall with the following arguments: the table handle provided by the guest domain, a virtual address at which to map the translation table, and a range of virtual addresses in kernel space to be used for mapping foreign pages granted by the guest domain. The hypervisor maps the translation table at the specified location in the driver domain address space, and it also records the specified virtual address range.

3.1.2 Page Sharing

When the guest domain wants to share a page, it invokes a hypercall with three arguments: a grant reference, the address of a guest page to be shared, and the translation table handle. The hypervisor first validates that the page belongs to that guest domain. Then it pins and maps the page into the driver domain address space using the virtual address range provided by the driver domain at initialization, and adds the machine address of the page to the software I/O translation table. The specific virtual address to use for this mapping in the virtual address range is given by the grant reference, which is used as a page number offset from the start of the virtual address range. Similarly, the guest domain revokes a grant by issuing a hypercall, in which the hypervisor unmaps and unpins the guest page from the driver domain address space.

The overhead of using grants in the driver domain is very low since the guest page is already mapped in the driver domain address space, and the machine address of the page can be directly read from the software I/O translation table. This table is indexed by a grant reference. Each entry contain two fields: (1) the machine address of the granted page, which is used by the driver domain to program a device DMA operation; and (2) the *status field*. Every time the driver domain uses a grant for I/O, it increments the *status field* to indicate to the hypervisor that the page is being used for an active I/O operation. When the I/O completes, the *status field* is decremented. If the *status field* is non-zero when the guest domain revokes the grant then the hypervisor returns an error to the guest domain, indicating that the page is still in use by an I/O device. However, this should not occur except in an error condition; either the guest domain is incorrectly trying to revoke a grant for a page that is being used for an active I/O operation, or the driver domain has incorrectly set the status field or is incorrectly using the page for I/O. In contrast, in the error-free case the guest domain can unilaterally revoke the grant whenever there is no pending I/O for that buffer.

3.2 Reuse of Grants

The cost of using the grant mechanism can be reduced by enabling reuse of the same grant across multiple I/O operations, taking advantage of locality in the use of I/O buffers, whether spatial locality (multiple I/O buffers sharing a page) or temporal locality. This allows a grant acquired for a guest memory page to be reused for future network I/O operations. A grant associated with a guest memory page can be revoked when the page is re-purposed to be used for non-I/O operations, provided all the related I/O operations have completed.

While different mechanisms can be used in different guest domains to keep track of grant use, the guest OS in our prototype uses a hash table for each associated software I/O translation table. Each entry in the hash table corresponds to a grant reference. If a grant is issued, the hash table entry contains the guest page frame number for that grant, and a reference counter which records the number of times the grant is being used. The guest domain checks whether a grant already exists for a page by looking up the hash table. If a grant does not exist, the guest domain adds the page to the hash table entry, initializes the reference counter and invokes a hypercall to issue the grant to the driver domain. If a grant already exists, which means the grant can be reused, the guest domain increments the reference counter and simply gives the corresponding grant reference to the driver domain. Thus, the grant overhead is negligible if the grant already exists.

Grant reuse is effective at reducing grant overhead only if the reused page remains mapped in the driver domain address space (mapping/unmapping is expensive). When the guest domain wishes to revoke a grant, the page must be unpinned and unmapped from the driver domain address space. In the proposed new mechanism, this can be done unilaterally by the guest domain without driver domain cooperation. In contrast, in the original grant mechanism, grants can be revoked only with the cooperation of the driver domain which has to issue the hypercall to unmap the guest pages from its address space.

While the new mechanism enables simple and unilateral grant revocation under the grant reuse scheme, the performance benefits are predominantly provided by the grant reuse scheme itself.

3.3 Promoting Temporal Locality

The new grant mechanism relies on locality in the use of I/O buffers to reduce performance overheads. To promote locality in the reuse of I/O buffers for network receive traffic, we create a pool of dedicated buffers in the guest domain for network I/O receive operations. The I/O buffers are allocated from the pool when needed for receive operations and are returned to the pool afterward. Recycling buffers from the pool promotes locality leading to a high degree of grant reuse.

All the memory in the pool is used only for network receive I/O. Therefore, the entire pool can remain granted persistently. However, as the number of pages in the pool increases, the guest domain can experience memory pressure. A mechanism is needed to shrink the pool when needed. Fortunately, Linux provides a slab cache mechanism that automatically adjusts its size under memory pressure. We exploit this mechanism and use a dedicated slab cache to implement the I/O buffer pool. The guest OS must revoke the grants associated with a guest memory page when it is released from the slab cache and therefore can be used for purposes other than I/O. This is especially important for pages which have been granted with read-write access (e.g.,

receive buffers).

3.4 Combining Grant and IOMMU Mechanisms

Both IOMMU hardware and Xen grant mechanism provide similar functionality for hardware and software components, respectively. An IOMMU provides translation of I/O addresses to machine memory addresses and controls which pages of a guest domain can be accessed by a hardware I/O device. The grant mechanism provides translation of grant references to machine memory addresses and controls which pages can be accessed by a driver domain software.

3.4.1 Unifying Memory Protection for Driver Domains and Direct I/O

The same hypercall interface in the new grant mechanism can be used by the guest domain to grant and revoke access to a memory page when performing I/O, regardless if the I/O operation is to be performed by a driver domain or directly by a physical device. This simplifies I/O support in the guest OS. In addition, this allows the same "grant" reuse mechanism to be used both for driver domains and directly accessed I/O devices, providing performance benefits for both cases.

If a guest domain has direct access to a pass-through device, the grant issue hypercall causes the hypervisor to add the specified page to the device IOMMU table; otherwise, if the guest domain is using a driver domain, the hypercall simply issues the grant as described earlier. Similarly, the grant revoke hypercall causes the page to be removed from the IOMMU table or from the software I/O translation table.

3.4.2 Using IOMMU Hardware with Driver Domains

While the grant mechanism protects guest domains' memory from improper access by software in driver domains, it cannot protect against improper access by the I/O devices that are programmed by the driver domain OS. To provide such protection, the hypervisor can setup an IOMMU table to check and translate addresses for DMA accesses performed by the I/O device. In this case, all granted guest pages must be added to the IOMMU table to enable the I/O device to access the pages. The new grant mechanism can be extended such that when a guest domain grants I/O access to one of its memory pages, the grant issue hypercall adds the guest page to the device IOMMU table in addition to mapping the page into the driver domain address space. When the guest domain revokes a grant, the hypervisor removes the corresponding page from the device IOMMU table in addition to unmapping the page from the driver domain.

The software I/O translation table in the new grant mechanism is not needed when the IOMMU hardware is used with driver domains. The driver domain OS can rely on the IOMMU table to do the address translations and therefore does not need the machine addresses to program the device DMA operations. Further, the status field need not be maintained by the driver domain because when the guest domain revokes a grant, the associated page is removed from the IOMMU table immediately. The IOMMU will then block any pending or subsequent accesses by the device to that page.

Unlike the software I/O translation table, there is only one IOMMU table per I/O device. So in this case, the hypervisor will assign distinct regions of the address space of the IOMMU table to each guest domain that is supported by the driver domain. Each region will have the same number of entries as the software I/O translation table and will provide translations for the grant references of the corresponding guest domain. To allow the driver domain to perform I/O operations using its local buffers, one additional region of the IOMMU address space is reserved to map all the local driver domain memory. This provides the driver domain with coarse grain IOMMU protection for its local pages, while providing fine grain protection for guest domains.

The driver domain OS computes an I/O address for a granted guest domain page as the sum of: (a) the base address of the IOMMU region associated with the corresponding guest domain, and (b) an offset equal to the grant reference. The driver domain OS uses this computed I/O address to program the device DMA operations, and the IOMMU translates the I/O address used in the DMA operation to the memory address of the guest page.

If a driver domain supports multiple devices, separate IOMMU tables can be used to provide finer protection. For example, using separate tables allows pages accessible by a network device to be segregated from pages accessible by a block device. In this case only the IOMMU table associated with the corresponding guest virtual device needs to be updated when a grant is issued or revoked.

4. EVALUATION

This section presents experimental results that quantify the benefits of the new grant mechanism. The new grant mechanism was implemented in the Xen hypervisor[2] and in a paravirtualized (PV) Linux domain[3]. The implementation provides full support for network I/O with driver domains, but does not yet support IOMMUs.

4.1 Experimental Setup and Methodology

The netperf TCP stream microbenchmark [1] is used in all the experiments to generate network traffic. The experiments include scenarios with network traffic in the transmit and receive direction between guest domains and a NIC, and network traffic between guest domains running on the same physical host. The new VMDq support is also used to receive packets [7, 15]. The CPU at the transmit side of the connection is not a resource bottleneck in any of the experiments. OProfile [12] is used to determine the number of CPU cycles spent when processing network packets.

The experiments are run on two server machines connected directly to each other using a 10 Gigabit CX4 Ethernet cable. The server running Xen has a 2.66 GHz Intel Core i7 quad-core CPU (with hyper-threading enabled), 6 GB of memory, and a 10 GbE Intel 82598EB Ethernet NIC (with VMDq support [9]). The external server has a 3.0 GHZ AMD Athlon dual-core CPU, 8 GB of memory, and a 10 GbE

[2] `xen-unstable.hg, changeset 17823:cc4e471bbc08`

[3] `linux-2.6.18.8-xen.hg, changeset 572:5db911a71eac`

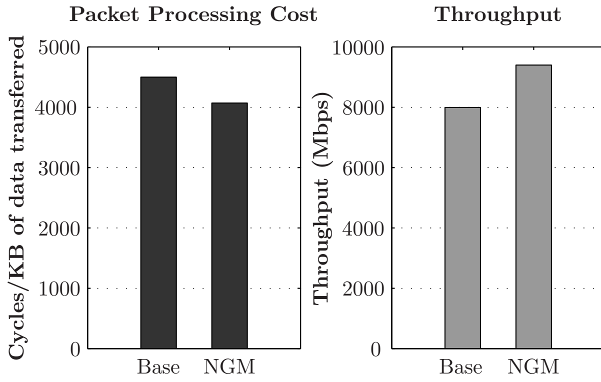

Figure 4: Impact of the New Grant Mechanism in receive path

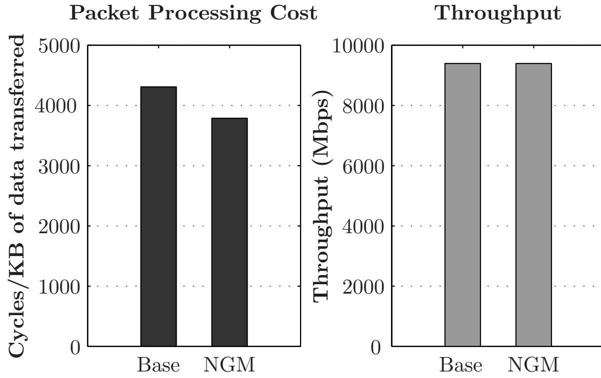

Figure 5: Impact of the New Grant Mechanism in receive path with VMDq Support

Intel 82598EB Ethernet NIC. The Intel Xen server is configured with up to two PV Linux guest domains, and one dedicated PV Linux driver domain in addition to the privileged domain 0. The driver domain and the guest domain(s) are each configured with 1 GB of memory and a single virtual CPU (to eliminate potential multi-CPU guest domain scheduling issues [14]). The external AMD server ran an Ubuntu distribution of native Linux kernel v2.6.30.3 which was configured to run only on a single CPU core.

4.2 Experimental Results

Figures 4 through 7 compare the packet processing cost (CPU cycles per KB of data transferred) and throughput (Mbps) in all experiments with and without the new grant mechanism (NGM). We observe a reduction in packet processing cost in all cases, with a maximum reduction of 31% in the transmit experiment (Figure 6). This results in higher throughput being achieved in two cases. In the receive experiment (without VMDq support) (Figure 4) there is a 18% increase in throughput, from 8.0 Gbps to 9.4 Gbps and in the inter-domain experiment (Figure 7) there is a 52% increase, from 17.8 to 26.9 Gbps. In both these cases, the driver domain CPU is initially a resource bottleneck. This limits the rate at which the guest domain can receive packets since the guest domain is not able to utilize its CPU core to the maximum extent possible. The new grant mechanism eliminates this resource bottleneck by reducing the packet processing cost in the driver domain. In turn, this enables the guest domain to make better use of its CPU core to

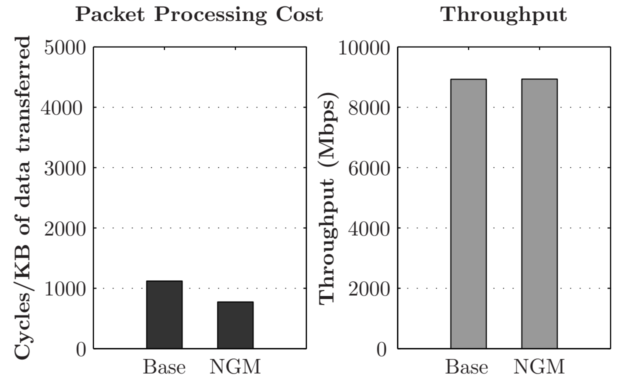

Figure 6: Impact of the New Grant Mechanism in transmit path

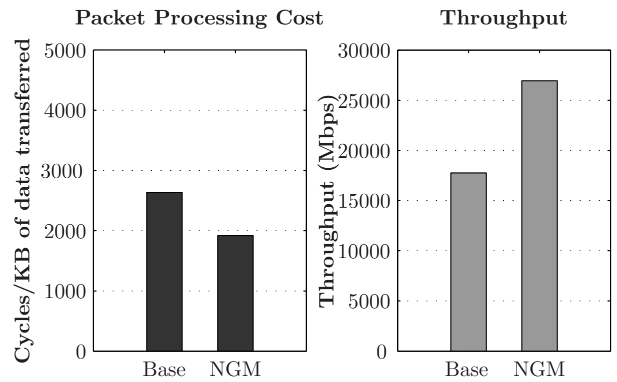

Figure 7: Impact of the New Grant Mechanism on Inter-Domain Networking

receive at a much higher rate until the guest domain CPU saturates and becomes the resource bottleneck. Interestingly, the case of inter-domain traffic shows a much greater increase in throughput. In this case, once again the driver domain CPU is initially the resource bottleneck. Unlike the receive case, however, here the driver domain suffers from grant overheads on both the transmit and receive sides. As grant reuse reduces overhead for both transmit and receive, the processing cost and throughput improve more significantly in this case than in the other cases. In the final two cases, there is no increase in throughput either because the CPU at the receiver is a bottleneck (Figure 6) or because the line rate (9.414 Gbps) is achieved even without the new grant mechanism (Figure 5).

Figure 8 compares the packet processing cost breakdown from the receive experiment. Here, the total packet processing cost is divided in two parts: the driver domain CPU (*dd*) and the guest domain CPU (*guest*). Each of these parts is further divided into the domain kernel (*kernel*) and the hypervisor (*xen*) components. Almost all the reduction in the total packet processing cost (shown earlier in Figure 4) comes from the *dd-xen* component. This is because, in the new grant mechanism, the driver domain does not issue hypercalls to have the hypervisor map and unmap the guest pages. Instead, the guest domain directly calls the hypervisor to perform these tasks. This implies that the cost of mapping and unmapping the guest pages should now be accounted for in the *guest-xen* component. But there is

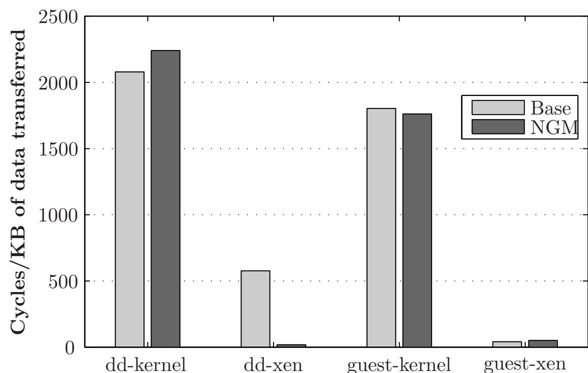

Figure 8: Breakdown of Packet Processing Costs from external receive experiment

no increase in the *guest-xen* component since a very high percentage (99%) of the grants are reused in the guest domain. There is also a slight increase in the *dd-kernel* component. This is attributed to the cost of synchronizing access to the software I/O translation table with the hypervisor using memory barriers. However, this increase is more than offset by the larger decrease in the *dd-xen* component. Similar breakdowns are not shown for other cases since they are very similar to the one shown here.

5. RELATED WORK

A grant reuse scheme to reduce memory sharing overheads during network I/O was originally discussed in [17]. That work did not implement the scheme or specify a design, but just estimated the potential benefits by running experiments with the grant mechanism disabled. Subsequently, we proposed a first version of a new grant mechanism with grant reuse [15]. That first version supported network traffic on the receive path, but only for NICs with VMDq support, and it lacked support for transmit – specifically, it lacked the ability to map a guest page into driver domain memory. In essence, the previously proposed mechanism could only be used by a guest domain to grant a driver domain access to program DMA operations to guest memory pages. In contrast, the design presented in this paper provides these previously missing functions and also describes unification of the new grant mechanism with IOMMUs.

Early studies of IOMMU performance demonstrated a high cost of setting up and removing page mappings for every I/O operation [6]. Subsequent work showed that mappings can be reused to reduce the overheads associated with IOMMUs, but without necessarily resorting to the coarse grain protection model [18]. The new grant mechanism proposed in this paper provides a unified approach to reuse both grants and IOMMU mappings.

Mechanisms have been proposed to support fast and cheap inter-domain communication using static shared memory channels between the communicating domains.
XenSocket [20] provides a standard POSIX socket level API to establish the shared communication channel. However, applications have to be rewritten to take advantage of this new socket level API. *XWay* [10] is another mechanism which, unlike XenSockets, provides full binary compatibility for applications using TCP sockets. Both of these mechanisms avoid the memory sharing overheads by setting up *static* shared memory channels during initialization and then transferring data over them. The new grant mechanism proposed in this paper enables efficient *dynamic* sharing of memory.

Mechanisms for controlling memory sharing have been developed for environments distinct from server virtualization. For example, the network protocol RDMA (Remote Direct Memory Access) allows hosts to "advertise" their memory buffers to enable remote hosts to read and/or write the memory [16]. Advertisement involves providing remote hosts with access keys called "steering tags" which bear a resemblance to grant references. Like the new grant mechanism described in this paper, the RDMA protocol allows hosts that advertise memory to remove access rights from remote hosts by invalidating the corresponding steering tags.

6. CONCLUSION

This paper describes a redesigned grant mechanism which improves the efficiency of network I/O virtualization while preserving the safety of the original mechanism. The new mechanism allows domains to unilaterally revoke grants to their local memory. In turn, this facilitates grant reuse for I/O to take advantage of temporal and/or spatial locality in guest OS usage of I/O buffers. This paper also shows that the new mechanism provides a unified interface for controlled memory sharing with driver domains and with I/O devices using new IOMMU hardware. The new mechanism is observed to reduce the per-packet overhead by up to 31% and also increase the throughput by up to 52%.

While the evaluation in this paper focuses on the benefits of the new grant mechanism for network I/O operations, the mechanism also can be used in more general scenarios. In particular, any memory sharing scenario that exhibits significant temporal and/or spatial locality should benefit from the new mechanism. For example, it should be possible to optimize block I/O operations using the new mechanism. Moreover, the new grant mechanism provides the full generality of the existing grant mechanism in Xen, while also providing higher robustness in the presence of misbehaving peers.

7. REFERENCES

[1] The Netperf benchmark. `http://www.netperf.org`.

[2] D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger, G. Regnier, R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, and J. Wiegert. Intel virtualization technology for directed I/O. *Intel Technology Journal*, 10(3), August 2006.

[3] AMD Corporation. AMD I/O Virtualization Technology (IOMMU) Specification. `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf`, Feb. 2007.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the 19th Symposium on Operating Systems Principles*, pages 164–177. ACM, October 2003.

[5] M. Ben-Yehuda, J. Mason, O. Krieger, J. Xenidis,

L. Van Doorn, A. Mallick, J. Nakajima, and E. Wahlig. Utilizing IOMMUs for virtualization in Linux and Xen. In *OLS '06: Proceedings of the Ottawa Linux Symposium*, July 2006.

[6] M. Ben-Yehuda, J. Xenidis, M. Ostrowski, K. Rister, A. Bruemmer, and L. Van Doorn. The price of safety: Evaluating IOMMU performance. In *OLS '07: Proceedings of the Ottawa Linux Symposium*, June 2007.

[7] S. Chinni and R. Hiremane. Virtual machine device queues. `http://software.intel.com/file/1919`, 2007.

[8] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williams. Safe hardware access with the Xen virtual machine monitor. In *OASIS '04: Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT Infrastructure*, October 2004.

[9] Intel Corporation. Intel 82598 10 GbE Ethernet controller open source datasheet. Revision 2.5, 2008.

[10] K. Kim, C. Kim, S. Jung, H. Shin, and J. Kim. Inter-domain socket communications supporting high performance and full binary compatibility on Xen. In *VEE '08: Proceedings of the International Conference on Virtual Execution Environments*, pages 11–20. ACM, March 2008.

[11] K. Mansley, G. Law, D. Riddoch, G. Barzini, N. Turton, and S. Pope. Getting 10 Gb/s from Xen: Safe and fast device access from unprivileged domains. In *Euro-Par 2007 Workshops: Parallel Processing*, pages 224–233, August 2007.

[12] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the Xen virtual machine environment. In *VEE '05: Proceedings of the International Conference on Virtual Execution Environments*, pages 13–23. ACM, June 2005.

[13] Neterion, Inc. Neterion X3100 series. `http://www.neterion.com/products/pdfs/X3100ProductBrief.pdf`, 2009.

[14] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling I/O in virtual machine monitors. In *VEE '08: Proceedings of the International Conference on Virtual Execution Environments*, pages 1–10. ACM, March 2008.

[15] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner. Achieving 10 Gb/s using safe and transparent network interface virtualization. In *VEE '09: Proceedings of the International Conference on Virtual Execution Environments*, pages 61–70. ACM, March 2009.

[16] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia. RFC 5040: A remote direct memory access protocol specification, October 2007.

[17] J. R. Santos, Y. Turner, G. Janakiraman, and I. Pratt. Bridging the gap between software and hardware techniques for I/O virtualization. In *USENIX'08: Proceedings of the USENIX Annual Technical Conference*, pages 29–42. USENIX Association, June 2008.

[18] P. Willmann, S. Rixner, and A. L. Cox. Protection strategies for direct access to virtualized I/O devices. In *USENIX'08: Proceedings of the USENIX Annual Technical Conference*. USENIX Association, June 2008.

[19] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A. L. Cox, and W. Zwaenepoel. Concurrent Direct Network Access for virtual machine monitors. In *HPCA '07: Proceedings of the 13th International Symposium on High-Performance Computer Architecture*, pages 306–317. IEEE Computer Society, February 2007.

[20] X. Zhang, S. McIntosh, P. Rohatgi, and J. L. Griffin. XenSocket: A high-throughput interdomain transport for virtual machines. In *Middleware '07: Proceedings of the International Conference on Middleware*, pages 184–203. Springer-Verlag New York, Inc., August 2007.