InfiniBand RDMA based Remote Swapping for Virtualized Environments

Balazs Gerofi
Graduate School of Information Science and Technology
The University of Tokyo
Tokyo, JAPAN
bgerofi@il.is.s.u-tokyo-ac.jp

Yutaka Ishikawa
Graduate School of Information Science and Technology
The University of Tokyo
Tokyo, JAPAN
ishikawa@is.s.u-tokyo-ac.jp

ABSTRACT

With major hardware manufacturers (e.g., Intel) turning their attention towards high-performance interconnects, InfiniBand is expected to receive significantly wider adoption not only in the high-performance computing (HPC) community, but also in the data-center space. At the same time, cloud computing, particularly Infrastructure as a Service (IaaS), has transformed the way how online services are currently deployed, placing them over virtualized environments, i.e., Virtual Machines (VM). Virtualization helps to improve server consolidation, however, it puts an increased pressure on memory. Remote swapping is a technique that exploits a remote machine's RAM to support the needs of an application running in a memory scarce environment, utilizing the Operating System's (OS) swapping mechanism.

In this paper we investigate how advanced features of high-performance interconnects, such as Remote Direct Memory Access (RDMA) can be exploited at the Virtual Machine Monitor (VMM) layer for providing remote swapping functionality. Our solution is completely transparent to the application as well as to the operating system executed in the VM, therefore, allowing both legacy and closed-source systems to take advantage of RDMA. We present micro- and application-level measurements concerning two different guest operating systems, Linux and Windows Server 2003.

Categories and Subject Descriptors

C.4 [**Performance Of Systems**]: Performance attributes; D.4.2 [**Operating Systems**]: Storage Management—*Swapping*

General Terms

Design, Networking, Performance

Keywords

Virtualization; Hypervisor; RDMA; Remote swapping

1. INTRODUCTION

During the past ten years, the high-performance computing (HPC) community has been experiencing a transition to commodity clusters connected with high-performance interconnects such as Myrinet [9] and InfiniBand [1]. Particularly, InfiniBand has become commoditized and is now available in the form of low-cost PCI-Express devices. These interconnects provide sophisticated features, such as Remote Direct Memory Access (RDMA) and OS-bypass communication, which not only offer high bandwidth and low latency, but also help server scalability by using reduced memory copies and little CPU for network operations [30].

Furthermore, with Intel's recent acquisition of QLogic's [28] InfiniBand product lines, it is highly anticipated that InfiniBand will witness a substantially wider adoption, not only in the HPC domain, but also in the data-center space. In fact, several recent studies have already demonstrated the advantages of InfiniBand, focusing particularly on RDMA, for the data-center context [17, 25, 20, 30, 26, 16, 15].

At the same time, the latest computing trend of cloud computing, especially Infrastructure as a Service (IaaS), transformed the way how services are deployed in Internet warehouses. Most online services are now running over virtualized environments, i.e., virtual machines (VM), and it is predicted that 85% of cloud workloads executed on x86 architecture will be deployed over VMs by 2018 [8]. Virtualization offers various advantages including better server consolidation, resource isolation, hardware independency, and easy manageability [12].

However, while running multiple virtual machines results in better server consolidation, especially in terms of CPU usage, memory can be only virtualized in the manner of partitioning among the VMs. With the increase in the number of virtual machines, especially with the latest CPU trend of multi- and many-core CPUs and the growing memory demands of cloud applications, RAM quickly becomes the limiting factor. If the memory requirements of a virtual machine exceeds the amount allocated, the workload executed in the VM may experience severe performance degradation due to memory trashing [19, 10]. Sharing memory through data deduplication [23], or dynamically adjusting the VM's memory image via ballooning [31] are attempts for alleviating this problem, nevertheless, they do not offer a solution when most of the VMs are running out of memory.

Remote swapping is a technique that takes advantage of a remote node's memory in order to store swapped out pages there, instead of writing them to the local disk. It is driven by the observation that network interconnects have improved considerably faster than hard disk speed and it has been studied extensively in the literature [13, 14, 24, 22, 10]. Remote swapping is often implemented as a network block device, which is in turn used by the operating system for its swap partition.

In this paper, we investigate how advanced features of high-performance interconnects, such as RDMA, can be utilized for remote swapping at the virtual machine monitor (VMM) layer. We propose an RDMA based remote swapping mechanism, which we implement as a disk image driver for the QEMU-KVM [21] virtualization infrastructure. There are several advantages of the VMM level approach:

- Complete application and operating system transparency, possibly speeding up both legacy and closed-source systems;
- The ability for taking advantage of RDMA even for systems that otherwise have no support for InfiniBand;
- The inherited properties of the virtualization itself, such as easy configurability and manageability.

We provide micro- and application level benchmarks running over two different operating systems, Ubuntu Server (Linux) and Windows Server 2003. The rest of this paper is organized as follows. Section 2 provides background information on InfiniBand. Section 3 details the design of our approach, while Section 4 reveals some of the implementation details. In Section 5, we present experimental evaluation, Section 6 surveys related works and finally, Section 7 concludes the paper.

2. BACKGROUND

In this Section, we provide an overview of InfiniBand with special attention to its data-center specific aspects.

2.1 InfiniBand Overview

InfiniBand [1] is an industry standard network fabric that is originally designed for interconnecting nodes in high-end computing (HEC) clusters. It is a high-speed, general purpose I/O interconnect. The yearly growth rate of InfiniBand in the TOP500 systems is close to 30% [30], indicating a strong increase in adoption. The main features of InfiniBand are Remote Direct Memory Access (RDMA) and OS-bypass communication. RDMA allows software to remotely read or write memory contents of another remote process without any software involvement at the remote side, while OS-bypass enables a communicating process to directly talk to the network hardware without the involvement of the underlying operating system. It also exempts the OS to spend extra CPU cycles on communication protocol related computation, just like how it does so when transmitting over TCP/IP. These features are extremely powerful and can be used to implement high-performance communication protocols.

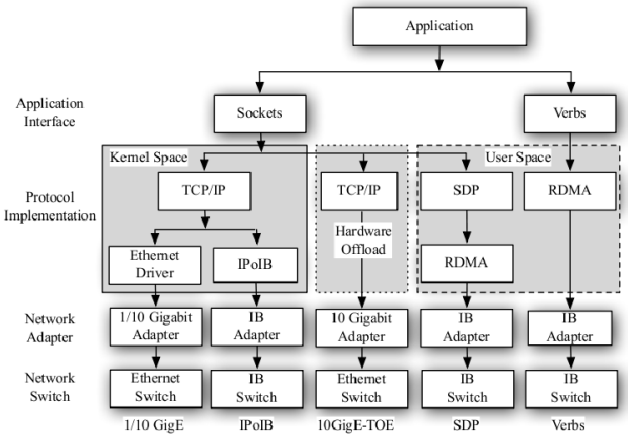


Figure 1: **Networking overview.**

Figure 1 provides an overview of the various networking layers including also InfiniBand. Software interaction with an InfiniBand Host Channel Adapter (HCA) can be performed mainly via three different interfaces. The lowest level, and most efficient way of programming the HCA is the *InfiniBand Verbs Layer*. Besides this, InfiniBand can also be accessed as a regular IP network interface using *InfiniBand IP Layer (IPoIB)* and through the *Socket Direct Protocol* [18] interface. We will be focusing on the Verbs and IPoIB in this paper, for which we provide more details in the following paragraphs.

The *InfiniBand Verbs Layer* is a low-level communication interface used to access the functionality provided by HCAs. This is illustrated in Figure 1 (on the right). Verbs that are used to transfer data work in a completely OS-bypassed manner. The verbs interface follows the *Queue Pair* (or communication end-points) model. Each queue pair has a certain number of work queue elements. Upper-level software places a work request on the queue pair that is then processed by the HCA. Work elements can represent send or receive requests, and also RDMA read and write requests. When a work element is completed, it is placed in the completion queue. Upper level software can detect completion by polling the completion queue.

InfiniBand also provides a driver for implementing the IP layer, often called *IP-over-IB* or *IPoIB* in short. This exposes the InfiniBand device as a regular network interface available from the system with an IP address. While Ethernet interfaces are presented as *eth0*, *eth1*, etc., IB devices are presented as *ib0*, *ib1* and so on. This interface is shown in Figure 1 (the second from the left). This interface does not provide OS-bypass and the operating system needs to process higher level protocols (e.g., TCP) the same way how it does over regular Ethernet interfaces. It is worth pointing out, that the advantage of IPoIB is that there is no need to the application to be modified or even recompiled, and it can directly take advantage of the higher performance provided by InfiniBand. However, RDMA provides much higher bandwidth which we will be also demonstrating in this paper against IPoIB.

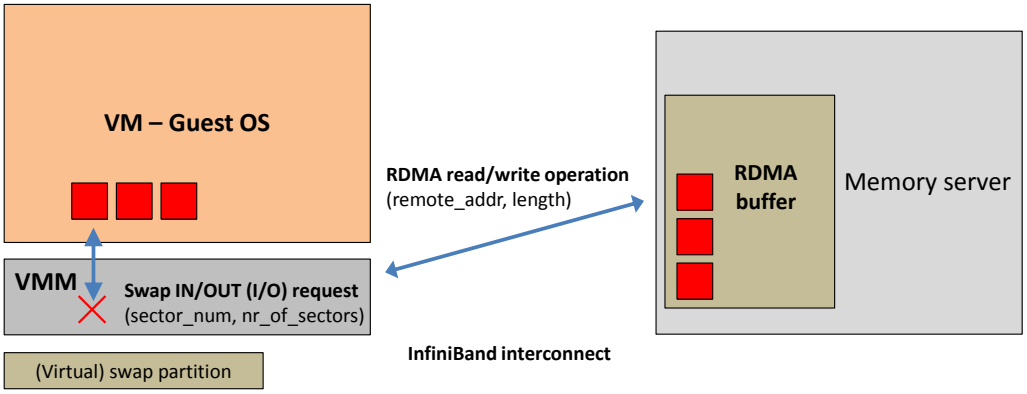


Figure 2: **RDMA based remote swapping overall design.**

3. DESIGN

Building a remote swapping mechanism at the virtual machine monitor layer brings up various design considerations. As it has been also pointed out in [10], one possible approach would be to make swapping decisions entirely at the virtual machine monitor layer. Because the VMM has full control over the mapping of the guest physical addresses to the corresponding physical page frames, it could easily modify mappings and select page frames to temporarily store on a remote memory server. When a page fault occurred on a guest physical address that is stored remotely, the VMM could transfer it back and update the mapping. One problem with this approach is that the VMM does not maintain information on page usages (by default) [31], which makes it difficult to choose victim pages [10]. Another problem, which has also plagued early virtual machine implementations [29], is *redundant paging*. In order to be able to determine which page to swap out the VMM would likely use some sort of LRU approximation based on page usage. However, while any LRU approximation algorithm can find frequently used pages, it cannot separate the infrequently used pages into pages that contain active data and unallocated pages that contain garbage. If an unallocated page is selected by VMM as a victim page, it will unnecessarily write back the page content. This problem stems from the fact that there would be two independent paging mechanism in the system, one in the guest OS and the other in the virtual machine monitor.

In order to overcome this issue, we chose to follow a rather simple approach and take advantage of the guest OS' swapping mechanism. Instead of introducing a new level of paging in the VMM, we monitor the swap device and catch the I/O requests of the guest OS. When the guest OS writes to the swap device, the blocks are sent to the remote machine and no local disk I/O is involved. Reads are also fullfiled from the remote machines' memory, respectively.

Additionally, to simplify things even more, we do not intend to build a second level of cache between the guest's buffer cache and the local disk, but instead we assume that the entire swap partition can be fit into the memory of the remote node(s). This decision also has to do with the nature how RDMA read and write works which will be explained below. Moreover, it eliminates the problem of the page placement policy [11] issue, i.e., the policy that defines whether a block should be transferred to the memory server or written to the local disk (in case the remote machine is just another level of cache).

3.1 RDMA

For security reasons, before executing any RDMA operations, it is required that the remote machine registers the memory buffer where data will be transferred to/from and informs the machine, which initiates the RDMA operation, of the buffer's remote key returned by the registration. As opposed to the design proposed in [22], where RDMA operations are performed by the memory server, we opted to initiate RDMA read/write by the VMM. This factor, together with the assumption that the swap device fits into remote memory, greatly simplifies the design of the entire remote swapping mechanism, because disk offsets can be directly translated to memory offsets on the remote host. Another advantage of this approach is the elimination of extra control messages before the actual RDMA operations are be performed, which are necessary in the architecture proposed by [22].

Instead, we load the disk image into the memory of the remote machine, register the buffer as RDMA capable memory area and exchange the remote keys in the initialization phase of the VM. As Figure 2 shows, the VMM intercepts the I/O operations and directly accesses the remote memory via RDMA operations. It is also worth pointing out, that we do need to maintain an RDMA capable memory area in the VMM, where we copy to/from the content of the swapped pages. The reason for such buffer is the fact that an RDMA capable buffers are pinned down in the memory, meaning that otherwise we would need to pin down the entire VM's memory image.

4. IMPLEMENTATION

As mentioned previously, we have implemented the RDMA based remote swapping in the QEMU-KVM virtualization infrastructure [21], as a disk image format driver. QEMU-KVM supports various disk images, which can be attached as block devices in the guest operating system.

We used the *raw* image driver as the base of our imple-

mentation and extended it to support RDMA transfer using OpenFabrics' native Infiniband verbs API [3]. The native InfiniBand verbs API allows direct user-level access from the *qemu-kvm* process to the IB host channel adaptor (HCA) resources while bypassing the operating system. At the IB verbs layer, we used the queue pair model for supporting channel-based communication semantics during the handshake between the VMM and the memory server, as well as for memory-based communication semantics, i.e., the RDMA operations.

5. EVALUATION

In this section we present performance evaluation of the RDMA based swapping mechanism compared to disk, Gigabit Ethernet and IPoIB based configurations.

5.1 Experimental Setup

Throughout our experiments the host machine of the replicated VM was equipped with a 4 cores Intel Xeon 2.2GHz CPU, with 2 hyperthreads per core (i.e., 8 hardware threads altogether), 6GBs of RAM and a 250GB SATA harddrive. Both the swap server and the client machines had an Intel 82546GB Gigabit Ethernet network interface and a Mellanox MT26428 InfiniBand QDR HCA.

The host machines run Ubuntu Server 9.10 on Linux kernel 2.6.37 and we used qemu-kvm 0.14.50 with kvm-kmod 2.6.37 as the basis of our implementation. The guest operating systems are Ubuntu Server 9.10 (Linux kernel 2.6.28) and Windows Server 2003, both 64bit versions. For the virtual machines in each experiment we used the KVM virtio disk and network drivers, both for the Linux and Windows guests. Intel's hardware MMU virtualization support, i.e. Extended Page Tables (EPT) were also enabled in all experiments.

As for the TCP/IP based experiments (both in case of Gigabit Ethernet and IPoIB), we use the Network Block Device [4] facility of the Linux kernel, through which we share a *ramdisk* on the memory server. The VMM accesses the block device via *nbd-client* which in turn communicates to *nbd-server* that shares the disk image located on ramdisk.

5.2 Raw Block Device Performance

Because we expose the swap partition as a block device to the guest operating system, we can actually perform raw disk speed measurements. In the first experiment we used the Linux *Bonnie++* hard disk benchmark to assess the performance of different configurations. Figure 3 illustrates the results we obtained.

There are six configurations we tested, host local disk, guest local disk, guest accessing a remote ramdisk via *nbd-client* (i.e., TCP) over Gigabit Ethernet, IP over InfiniBand and finally over our RDMA implementation. Bonnie++ provides several report types, from which we included four values, sequential write and sequential read for both per-character and per-block based operations. The first thing one can notice is that there is only a small performance difference between accessing the local disk from the host or from a KVM guest.

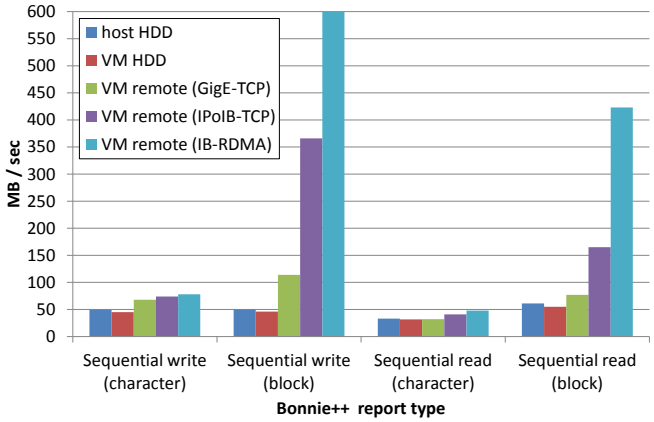


Figure 3: **Linux - Bonnie++ results over host HDD, guest HDD, remote RAM disk over GigE, IPoIB and RDMA.** *The block sequential write result for RDMA is truncated for clarity, the actual value is 1150 MB/s.*

Also, there is isn't that significant difference between the attained bandwidth when the benchmark uses operations per-character. Furthermore, for the per-character operations, it is also worth noticing that write performance is generally better than that for read. This is not true, however, for block-based access.

Nevertheless, for block-based access there are substantial differences. While we obtain around 50MB/s on the local disk for write, the GigE configuration yields around 110MB/s, which is close to the theoretical throughput of GigE. On the other hand, IPoIB attains more than 350MB/s and RDMA yields an unprecedent 1150MB/s for block based write performance, which is an order of magnitude better performance than that of GigE. A somewhat strange result can be seen for the read performance, both the IPoIB and RDMA based configurations attain significantly less throughput than in case of write, which we believe has probably to do with the Bonnie++ implementation, which we will also comment on below.

Figure 4 shows the results for Windows Server 2003 that we obtained with the Passmark Performance Test Suit [27]. There are three different tests, sequential write, sequential read, and random seek plus read-write, which all have results for asynchronous and synchronous mode. We do not present measurements for the host machine, because it runs Linux and we did not have another machine running the same Windows operating system. However, we do present measurements (executed in the guest) for the local hard disk, remote ramdisk over Gigabit Ethernet, IP over InfiniBand and InfiniBand RDMA.

First of all, a general observation, which isn't surprising that in all cases the asynchronous performance is better than that run in synchronous mode. We obtain similar results for the local hard disk performance as we did in the Linux case, however for Gigabit Ethernet remote ramdisk, Passmark reports slightly better performance than Bonnie++ in Linux. It is also generally true, that any of the remote ramdisk configurations provide much better random seek performance than

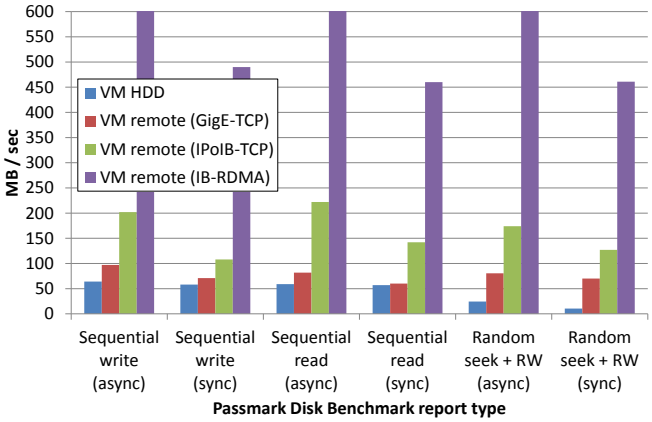


Figure 4: **Windows Server 2003 - Passmark Performance Test results over guest HDD, remote RAM disk over GigE, IPoIB and RDMA.** *The asynchronous sequential write, read, and random seek+RW results for RDMA are truncated for clarity, the actual values are 1132, 1128, and 1090 MB/s, respectively.*

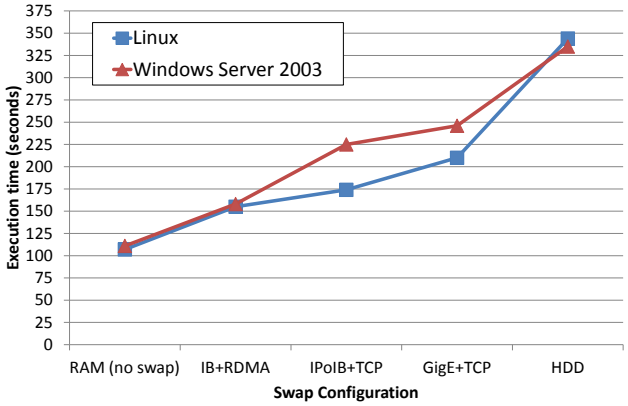


Figure 5: **Quick Sort Runtimes (1.2GB integer array) on Linux and Windows Server 2003 guests with various swap configurations.**

the local disk.

On the other hand, IPoIB based remote disk performs worse in this configuration. For instance the sequential write performance in Windows Server 2003 over IPoIB is around 200 MB/s while Bonnie++ reports 350MB/s for the similar scenario in Linux. Again, InfiniBand RDMA yields significantly better performance in the Windows Server case as well. However, an interesting observation is that both asynchronous read and write over RDMA attains over a Gigabyte/s throughput, while Bonnie++ reports the same value (around 1150MB/s) only for write performance. This might indicate that Bonnie++ accesses the disk differently for read and write, however, we did not verify this.

A final conclusion with respect to raw block device measurements, is that regardless the operating system (Linux or Windows Server), the RDMA based remote disk performs substantially better. In some cases it yields an order of magnitude better performance than IP over InfiniBand.

5.3 Quick Sort

We chose sorting as one of the high level benchmarks, because it is a frequently utilized routine in many applications and quick-sort in particular is often used in the literature for measuring remote swapping performance [22, 10]. In this experiment we configured the virtual machines to have 1GB RAM (unless stated otherwise), one virtual CPU and use the standard C library quick-sort routine to sort an array of integers that spawn 1.2 GBs.

We present measurements for both Linux and Windows Server 2003, where in each case we have five configurations. We run the first measurement having 2Gs of RAM in the VMs with swapping turned off entirely. Besides this, we have swapping to local hard disk, remote ramdisk over Gigabit Ethernet, IP over InfiniBand, and RDMA.

In order to make a fair comparison we created a file with random numbers and the tool reads this file so that both the Linux and the Windows execution sorts the same input. Moreover, we only measure the quick sort routine's execution time, i.e., excluding the duration required to read the input file.

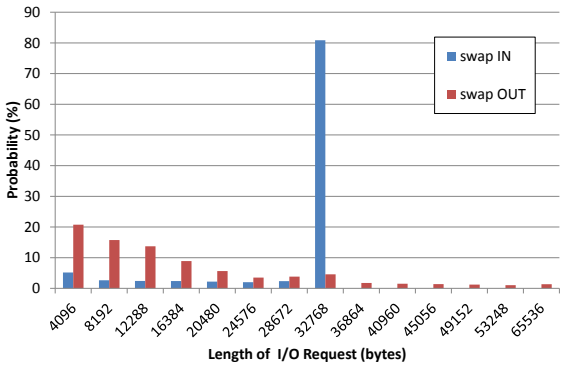
Figure 5 shows the results obtained both for Linux and Windows Server 2003. The first thing one can notice is that both the in RAM and the local hard disk swapping case show very close values, which confirms that we actually run the same algorithm. Another interesting observation, is that both for Linux and Windows Server 2003 the InfiniBand RDMA based remote swapping yields almost the same results, which is an improvement of 2.2X compared to the HDD swapping, corresponding to 70% of the performance of the in memory execution. As for the Gigabit Ethernet and the IP over InfiniBand case, they proved to be 41% and 51% better than the local disk execution for Windows, and 60% and 94% better for Linux, respectively. It is also worth pointing out that Linux performs 15% and 28% better than Windows Server 2003, for GigE and IPoIB, respectively. RDMA, on the other hand, is faster with a factor of 13% and 45% than IPoIB for Linux and Windows, respectively.

Table 1: **Number of bytes swapped IN and OUT during quicksort.**

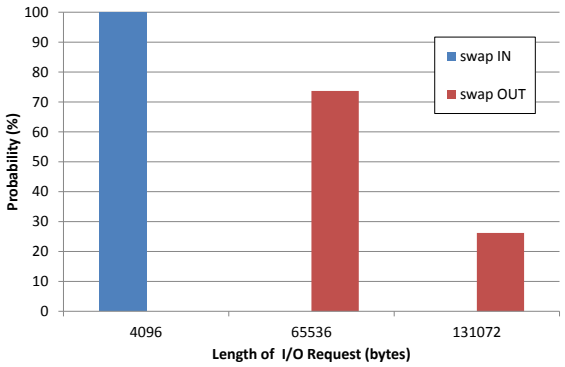
OS	swap IN	swap OUT
Linux 2.6.28	2.8 GB	3.4 GB
Windows Server 2003	1.5 GB	2.1 GB

One particular advantage of a VMM level investigation is the ability to gain insight how different operating systems work under similar workloads, as observed by the hardware. For example, we were curious to see how many bytes each of the OSes swap in and out during the quick sort execution, so we logged this information at the VMM level. Surprisingly, they behave rather differently. Table 1 demonstrates the results. As seen, Linux generates significantly more traffic than the Windows Server.

In order to clarify why this is happening, we also logged the



(a) Linux.



(b) Windows Server 2003.

Figure 6: **Swap I/O request length distribution.**

length of each I/O request and computed their distribution as a discrete probability density function. Figure 6 compares the values for Linux and Windows Server 2003. Note, that we only show values on the figures which have higher probability than 0.5% (i.e., not all the possible request lengths are displayed). As seen on Figure 6a, Linux reads 32KB from the swap partition in 80% of the cases, while Windows 2003 (shown on Figure 6b) issues requests almost exclusively in 4KB chunks. On the other hand, Linux has a wider distribution in terms of the length of write requests, while Windows Server mainly uses two values, 64KB and 128KB.

5.4 Fast Fourier Transformation

The last application we investigate is Fast Fourier Transformation (FFT) from the NAS Parallel Benchmarks Suit [2]. FFT is an often used algorithm in various contexts so we believe it is a good candidate for examination. On the other hand, FFT is very memory intensive, it touches its working set frequently, causing heave swapping behavior in case the physical memory is insufficient.

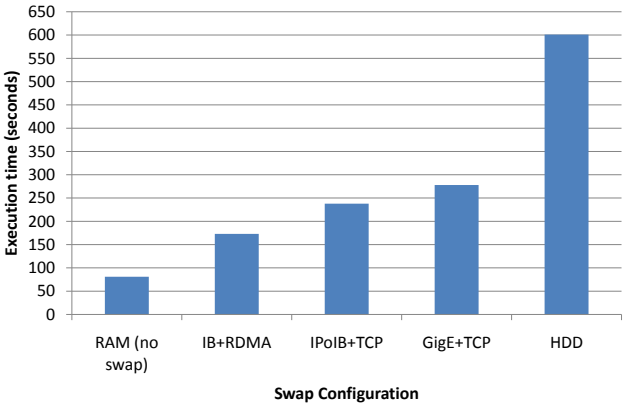


Figure 7: **NAS Parallel Benchmark FFT Class B execution time with various swapping strategies.**

We used the class B FT benchmark (over 1 vCPU), which has approximately 1.3GB working set. We set the VM to have 1.25GB memory and measure the execution for various swapping mechanism, such as local disk, ramdisk over GigE, IP over InfiniBand, and finally, InfiniBand RDMA. We also measure the execution time for the case where the entire working set fits into the physical RAM.

Figure 7 illustrates the results. As seen, in RAM execution takes approximately 80 seconds, while swapping to hard disk imposes a huge performance penalty, rendering the execution time to approximately 10 minutes. Even swapping over Gigabit Ethernet yields substantial improvements, running more than 2X faster than with disk based swapping. IPoIB improves the performance with around 16%, while RDMA adds another 37% speedup. After all, RDMA yields 3.4X faster execution than with swapping to local disk attaining around 46% of the in RAM execution's performance.

6. RELATED WORK

In this section, we survey related studies in the domain of remote memory utilization as well as other fields where RDMA proved to be advantageous in the data-center space.

6.1 Utilizing Remote Memory

The idea of utilizing remote nodes' memory in cluster environments have been studied extensively in the literature. Feeley et al. implemented the Global Memory Management (GMM) system [13], a network cache for swapped pages by modifying the memory management system of the DEC OSF/1 kernel. GMM considers remote memory as a cache of network swapped pages, where swapped out pages are also written to the local disk. Remote servers only cache clean pages and can arbitrarily drop pages according to their local memory needs. MOSIX [6], on the other hand, is a load balancing system built on top of a transparent job migration framework, where migration is utilized in order to avoid swapping to local disk.

Network RamDisk [14] uses the main memory of remote workstations as a faster-than-disk storage device. Various reliability policies are considered making the device tolerant to single workstation crashes. The Network RamDisk is implemented both on the Linux and the Digital Unix operating systems, as a block device driver without any modifications to the kernel code. Nswap [24] is a network swapping system for heterogeneous Linux clusters and networks of Linux machines. Nswap transparently provides network swapping to cluster applications. It is implemented as a loadable ker-

nel module and applications can take advantage of network swapping without having to recompile or link with special libraries. All the above mentioned studies, however, consider Ethernet or ATM based networks and require operating system level support in order to take advantage the memory of a remote machine. To the contrary, we provide a solution where an unmodified operating system can take advantage of the sophisticated features of high-performance interconnects in a completely transparent manner.

An InfiniBand based network block device has been proposed in [22], where RDMA operations are utilized. Besides the fact that this solution works on the operating system level, our remote memory access design is also different, because we initiate RDMA operations by the client machine (i.e., the VMM) as opposed to the memory server. Chen et al. proposed a remote swapping system at the virtual machine monitor level [10]. They provide an implementation in the Xen virtualization framework [7], but they only consider TCP over Ethernet and due to the Xen's limitations they were unable to investigate their solution's benefit for different operating systems.

6.2 RDMA in Data-centers

With the increasing adoption of high-performance interconnects in the data-center space, the benefits of RDMA have been studied in various contexts. Gu et al. implemented RAMS, an RDMA-based collaborative I/O cache system architecture for clustered servers [17]. They were motivated by the observation that significant CPU time is spent on intra-cluster communication for the expensive TCP protocol handling and showed substantial performance boost for a clustered Web server running over RAMS. Noronha et al. investigated how RDMA can be efficiently integrated into Network File System standard (NFSv4) [25]. They found that a carefully designed RDMA based implementation significantly outperforms the default TCP based solution.

Jose et al. extended the existing open-source Memcached software and made it RDMA capable [20]. Memcached is a key-value distributed memory object caching system, which is used widely in the data-center environment for caching results of database calls, API calls or any other data. They found that with RDMA, memcached's *Get* requests over a range of message sizes show better performance by a factor of five to ten even when compared to IP over InfiniBand.

In recent years, the MapReduce computing model (and its open-source implementation, Hadoop [5]) has emerged as a scalable model that is capable of processing petabytes of data. Sur et al. examined how RDMA can benefit the Hadoop Distributed File System (HDFS) [30], which lies at the heart of the Hadoop ecosystem and found that the impact is substantial. They showed improvements from 11% to 100% over multiple benchmarks.

In the context of pro-active fault-tolerance mechanism, an RDMA based process live migration has been proposed in [26]. Their protocol fully pipelines data writing, data transfer, and data read operations during different phases of a migration cycle, aggregates data writes on the migration source node and transfers data to the target node via high throughput RDMA transport. Experiments showed over 10X speedup compared to the baseline TCP implementation. Gerofi et al. demonstrated how RDMA can be utilized in the context of highly available virtual machines, focusing on memory intensive workloads over multiple vCPUs [16]. They found that for certain workloads, close to native VM performance can be achieved even for configurations with up 8 vCPUs.

7. CONCLUSIONS AND FUTURE WORK

High-performance interconnects are gaining adoption not only in the high-performance computing community, but also in the data-center space. On the other hand, virtualization has become the de facto infrastructure for deploying online services, mainly due to its advantages for better hardware consolidation.

However, running several virtual machines on the same physical machine puts an increasing pressure on the memory of the host. Remote swapping helps to ease this pressure by possibly utilizing a remote server's memory for the application that runs in a memory scarce environment.

In this paper we have investigated how to make use of advanced features of high-performance interconnects, such as Remote Direct Memory Access (RDMA) at the virtual machine monitor level in order to provide remote swapping functionality. Our main contribution is a solution that offers high performance remote swapping facility in an application and operating system transparent manner, allowing both closed-source and legacy software stacks to take advantage of the new features of an HPC interconnect.

We have implemented this remote swapping mechanism in the Linux KVM virtualization infrastructure and evaluated our solution over two different guest operating systems. Linux and Windows Server 2003, without the need of having native InfiniBand RDMA support in the guests. We have demonstrated the benefits of RDMA through various micro- and application-level benchmarks and showed that for instance an RDMA based remote disk performs multiple times faster than that over IPoIB. Also, application level benchmarks, such as Quick Sort of the Fast Fourier Transformation, benefit significantly when swapping is performed over RDMA.

In the future we intend to investigate how an RDMA based remote swapping mechanism could be utilized in a more dynamic manner, such as that the VMM could monitor the guest's behavior and switch to remote swapping when it's necessary, in a completely guest transparent manner. Another possible improvement that might alleviate the problem of small writes would be to aggregate writes and issue them together to the memory server, however this requires additional memory allocation in the VMM, which is against the primary assumption of running in a memory scarce environment.

8. ACKNOWLEDGMENTS

This work has been supported by the Core Research for Evolutional Science and Technology (CREST) project of the Japan Science and Technology Agency (JST).

9. REFERENCES

[1] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2.

[2] NASA. NAS Parallel Benchmarks. `http://www.nas.nasa.gov/Software/NPB`.

[3] OpenFabrics Alliance. `http://www.openfabrics.org`.

[4] Linux Network Block Device (TCP version). `http://nbd.sourceforge.net/`, 2010.

[5] Apache®. Hadoop. `http://hadoop.apache.org`, 2011.

[6] A. Barak, S. Guday, and R. G. Wheeler. *The MOSIX Distributed Operating System: Load Balancing for UNIX*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1993.

[7] 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 *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[8] T. J. Bittman and L. Leong. Virtual Machines Will Slow in the Enterprise, Grow in the Cloud, 2011.

[9] 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*, 15:29–36, February 1995.

[10] H. Chen, Y. Luo, X. Wang, B. Zhang, Y. Sun, and Z. Wang. A Transparent Remote Paging Model for Virtual Machines. In *International Workshop on Virtualization Technology*, IWVT '08, 2008.

[11] Z. Chen, Y. Zhou, and K. Li. Eviction based cache placement for storage caches. In *in USENIX Annual Technical Conference. Usenix*, pages 269–282, 2003.

[12] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[13] M. J. Feeley, W. E. Morgan, E. P. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath. Implementing global memory management in a workstation cluster. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, SOSP '95, pages 201–212, New York, NY, USA, 1995. ACM.

[14] M. D. Flouris and E. P. Markatos. The Network RamDisk: Using remote memory on heterogeneous NOWs. *Cluster Computing*, 2:281–293, October 1999.

[15] B. Gerofi, H. Fujita, and Y. Ishikawa. An Efficient Process Live Migration Mechanism for Load Balanced Distributed Virtual Environments. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, pages 197–206, Sept 2010.

[16] B. Gerofi and Y. Ishikawa. RDMA based Replication of Multiprocessor Virtual Machines over High-Performance Interconnects. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, pages 35–44, 2011.

[17] P. Gu and J. Wang. RAMS: a RDMA-enabled I/O cache architecture for clustered network servers. In *Proceedings of the international workshop on Storage network architecture and parallel I/Os*, pages 66–72, New York, NY, USA, 2004. ACM.

[18] InfiniBand Trade Association. Sockets Direct Protocol. `http://www.infinibandta.com`, 2011.

[19] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Geiger: monitoring the buffer cache in a virtual machine environment. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XII, pages 14–24, New York, NY, USA, 2006. ACM.

[20] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur Rahman, N. Islam, X. Ouyang, H. Wang, S. Sur, and D. Panda. Memcached Design on High Performance RDMA Capable Interconnects. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 743 –752, sept. 2011.

[21] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux Virtual Machine Monitor. In *Ottawa Linux Symposium*, pages 225–230, July 2007.

[22] S. Liang, R. Noronha, and D. Panda. Swapping to Remote Memory over InfiniBand: An Approach using a High Performance Network Block Device. In *Cluster Computing, 2005. IEEE International*, pages 1 –10, sept. 2005.

[23] G. Miłós, D. G. Murray, S. Hand, and M. A. Fetterman. Satori: enlightened page sharing. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09. USENIX Association, 2009.

[24] T. Newhall, S. Finney, K. Ganchev, and M. Spiegel. Nswap: A Network Swapping Module for Linux Clusters. In *Euro-Par 2003 Parallel Processing*, pages 1160–1169, 2003.

[25] R. Noronha, L. Chai, S. Shepler, and D. K. Panda. Enhancing the Performance of NFSv4 with RDMA. In *Proceedings of the Fourth International Workshop on Storage Network Architecture and Parallel I/Os*, pages 90–96, Washington, DC, USA, 2007. IEEE Computer Society.

[26] X. Ouyang, R. Rajachandrasekar, X. Besseron, and D. Panda. High Performance Pipelined Process Migration with RDMA. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 314 –323, may 2011.

[27] PassMark. PerformanceTest. `http://www.passmark.com/products/pt.htm`, 2012.

[28] QLogic Corp. ®. Infiniband. `http://www.qlogic.com/Pages/default.aspx`, 2011.

[29] L. H. Seawright and R. A. MacKinnon. VM/370 - A study of multiplicity and usefulness. *IBM Systems Journal*, 18(1):4 –17, 1979.

[30] S. Sur, H. Wang, J. Huang, X. Ouyang, and D. K. Panda. Can High-Performance Interconnects Benefit Hadoop Distributed File System? In *Micro Architectural Support for Virtualization, Workshop on*, MASVDC '10, 2010.

[31] C. A. Waldspurger. Memory resource management in VMware ESX server. *SIGOPS Oper. Syst. Rev.*, 36:181–194, December 2002.