High Performance Virtual Machine Migration with RDMA over Modern Interconnects

Wei Huang #1, Qi Gao #2, Jiuxing Liu *3 , Dhabaleswar K. Panda #4

#*Computer Science and Engineering, The Ohio State University*
2015 Neil Avenue, Columbus, OH 43210, USA
1huanwei@cse.ohio-state.edu
2gaoq@cse.ohio-state.edu
4panda@cse.ohio-state.edu

**IBM T. J. Watson Research Center*
19 Skyline Drive, Hawthorne, NY 10532, USA
3jl@us.ibm.com

Abstract— **One of the most useful features provided by virtual machine (VM) technologies is the ability to migrate running OS instances across distinct physical nodes. As a basis for many administration tools in modern clusters and data-centers, VM migration is desired to be extremely efficient to reduce both migration time and performance impact on hosted applications.**

Currently, most VM environments use the Socket interface and the TCP/IP protocol to transfer VM migration traffic. In this paper, we propose a high performance VM migration design by using RDMA (Remote Direct Memory Access). RDMA is a feature provided by many modern high speed interconnects that are currently being widely deployed in data-centers and clusters. By taking advantage of the low software overhead and the one-sided nature of RDMA, our design significantly improves the efficiency of VM migration. We also contribute a set of micro-benchmarks and application-level benchmark evaluations aimed at evaluating important metrics of VM migration. The evaluations using our prototype implementation over Xen and InfiniBand show that RDMA can drastically reduce the migration overhead: up to 80% on total migration time and up to 77% on application observed downtime.

I. Introduction

Recently, virtual machine (VM) technologies are experiencing resurgence in both industry and academia. They provide desirable features to meet demanding requirements of computing resources in modern computing systems, including server consolidation, performance isolation and ease of management. Migration is one of the most important features provided by modern VM technologies. It allows system administrators to move an OS instance to another physical node without interrupting any hosted services on the migrating OS. It is an extremely powerful cluster administration tool and serves as a basis for many modern administration frameworks which aim to provide efficient online system maintenance, reconfiguration, load balancing and proactive fault tolerance in clusters and data-centers [2], [16], [24], [26]. As a result, it is desirable that VM migration be carried out in a very efficient manner, with both short migration time and low impact on hosted applications.

The state-of-the-art VM technologies such as Xen [4] and VMware [24] achieve VM migration by transferring the memory pages of the guest OS from the source machine to the destination host over the TCP socket and resuming execution at the destination. While migrating over the TCP socket ensures the solution can be applicable to the majority of industry computing environments, it can also lead to suboptimal performance due to high protocol overhead, heavy kernel involvement and extra synchronization requirements of the two sided operations, which may overshadow the benefits of migration.

Meanwhile, recent high speed interconnects, such as InfiniBand [10], Myrinet [15] and Quadrics [21], provide features including OS-bypass communication and Remote Direct Memory Access (RDMA). OS-bypass allows data communication to be directly initiated from process user space; on top of that, RDMA allows direct data movement from the memory of one computer into that of another. With very little software overhead, those communication models allow data to be transferred in a highly efficient manner. As a result, they can benefit VM migration in various aspects. First, with extremely high throughput offered by high speed interconnects and RDMA, the time needed to transfer the memory pages can be reduced significantly, which leads to an immediate save on total VM migration time. Further, data communication over OS-bypass and RDMA does not need to involve CPU, caches, or context switches. This allows migration to be carried out with minimum impact on guest operating systems and hosted applications.

In this paper, we study RDMA based virtual machine migration. We analyze the challenges to achieve efficient VM migration over RDMA, including protocol design, memory registration, non-contiguous data transfer, network QoS, etc. We carefully address these challenges to fully leverage the benefits of RDMA. This paper also contributes a set of micro-benchmarks and application-level benchmark evaluations, which reflect several important requirements on VM migration posed by real-world usage scenarios. Evaluations with our prototype implementation of Xen migration over InfiniBand show that RDMA based protocols are able to significantly improve the migration efficiency. For example, compared with the original Xen migration over TCP/IP over InfiniBand (IPoIB) [9], our design over InfiniBand RDMA reduces the impact of migration on SPEC CINT 2000 Benchmarks [22] by an average of 54% when the server is lightly loaded, and an average of 70% when it is heavily loaded.

The rest of the paper is organized as follows: we start with a brief overview of VM migration and InfiniBand architecture as background in Sections II and III. We analyze the potential benefits of RDMA based VM migration and identify several key challenges towards an efficient design in Section IV. Then, we address the detailed design issues in Section V and carry out performance evaluations in Section VI. Last, we discuss related work in Section VII and conclude the paper in Section VIII.

II. Virtual Machine Migration

Xen [4] is a popular virtual machine technology originally developed at University of Cambridge. Figure 1 illustrates the structure of a physical machine hosting Xen. The Xen hypervisor (the VMM) is at the lowest level and has direct access to the hardware. Above the hypervisor are the Xen domains (VMs) running *guest OS* instances. Each guest OS uses a pre-configured share of physical memory. A privileged domain called *Domain0 (or Dom0)*, which is created at boot time, is allowed to access the control interface provided by the hypervisor and performs the tasks to create, terminate or migrate other guest VMs *(User Domain or DomU).*

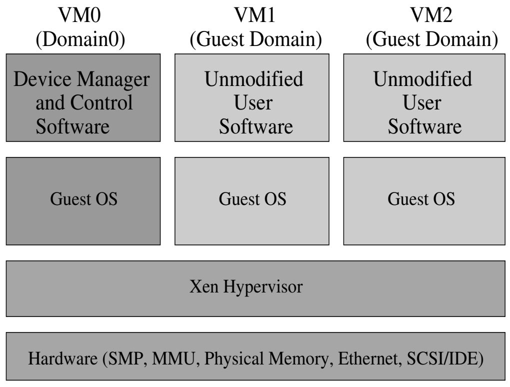


Fig. 1. The structure of the Xen virtual machine monitor

When migrating a guest OS[1], Xen first enters the *pre-copy* stage, where all the memory pages used by the guest OS are transferred from the source to pre-allocated memory regions on the destination host. This is done by user level *migration helper processes* in Dom0s of both hosts. All memory pages of the migrating OS instance (VM) is mapped to the address space of the *helper processes*. After that the memory pages are sent to the destination host over TCP/IP sockets. Memory pages containing page tables need special attention that all machine dependent addresses (*machine frame numbers* or *mfn*) are translated to machine independent addresses (*physical frame numbers* or *pfn*) before the pages are sent. The addresses will be translated back to *mfn* at the destination host. This ensures transparency since guest OSes reference memory by *pfn*. Once all memory pages are transferred, the guest VM on the source machine is discarded. And the execution will resume on the destination host.

[1]For Xen, each domain (VM) hosts only one Operating System. Thus, in this paper, we do not necessarily distinguish among migrating a VM, a domain, and an OS instance.

Xen also adopts live migration [3], where the pre-copy stage involves multiple iterations. The first iteration sends all the memory pages, and the subsequent iterations copy only those pages dirtied during the previous transfer phase. The pre-copy stage terminates when the page dirty rate exceeds the page transfer rate or when the number of iterations exceeds a predefined value. In this way, the only observed downtime by the hosted applications is at the last iteration of pre-copy, when the VM is shutdown to prevent any further modification to the memory pages.

III. InfiniBand Architecture and RDMA

InfiniBand [10] is an emerging interconnect offering high performance and features such as OS-bypass and RDMA. RDMA semantics can be used to directly read (RDMA read) or modify (RDMA write) the contents of remote memory. RDMA operations are one sided and do not incur software overhead on remote side. Before RDMA operations can take place, the target side of the operation must register the memory buffers and send the remote key returned from the registration to the operation initiator. The registration helps InfiniBand to get the DMA addresses of the memory buffers used in user processes. It also avoids faulty program from polluting memory on the target machines. InfiniBand supports non-contiguity on the initiators (RDMA read with scatter or RDMA write with gather). However, the target buffers of RDMA operations must be contiguous.

IV. Motivation

In this section we look at the potential benefits of migration over RDMA, which motivate the design of RDMA based migration. We also analyze several design challenges to fully exploit the benefits of RDMA.

A. Benefits of RDMA based Migration

Besides the increase of bandwidth, RDMA can benefit virtual machine migration mainly from two aspects.

First, RDMA allows the memory to be directly accessed by hardware I/O devices without OS involvement. It indicates that the memory pages of the migrating OS instance can be directly sent to the remote host in a zero-copy manner. This avoids the TCP/IP stack processing overhead. Also for VM migration, it reduces context switches between the migrating VM and the privileged domain, which hosts the migration helper process.

Second, the one sided nature of RDMA operations can alleviate the burden on the target side during data transfer. This further saving on the CPU cycles is especially important in some cases. For instance, one of the goals of VM technology is server consolidation, where multiple OSes are hosted in one physical box to efficiently utilize the resources. Thus, in many cases a physical server may not have enough CPU resources to handle migration traffic without degrading the hosted application performance.

Direct memory access and the one sided nature of RDMA can significantly reduce the software involvement during migration. This reduced overhead is critical especially in performance-sensitive scenarios, such as for load balancing or proactive fault tolerance.

B. *Design Challenges*

Though RDMA has the potential to greatly improve the VM migration efficiency, we need to address multiple challenges to fully exploit the benefits of RDMA. Now we take a closer look at those challenges. Our description here focuses on Xen migration and InfiniBand RDMA. However, the issues are common to other VM technologies and RDMA capable interconnects.

Design of efficient migration protocol over RDMA: As we have mentioned in Section II, normal data pages can be directly transferred during migration, but page table pages need to be pre-processed before being copied out. Our migration protocol should be carefully designed to efficiently handle both types of memory pages. Also, RDMA write and RDMA read both can be utilized for data transfer, but they have different impact on the source or destination hosts. How to minimize such impact during migration needs careful considerations.

Memory Registration: InfiniBand requires the data buffers to be registered before they can be involved in data transfer. Earlier research [12] in related areas proposed two solutions. One is to copy the data into pre-registered buffers (copy-based approach). The other is to register the user data buffers on the fly (zero-copy approach). However, neither of these two approaches works well in our case. Copy-based approach will consume CPU cycles and pollute data caches, suffering the same problem as TCP transfer. Zero-copy approach requires registering the memory pages that belong to a foreign VM, which is not currently supported by the InfiniBand driver for security reasons.

Non-contiguous Transfer: Original Xen live migration transfers the memory pages in page granularity. Each time the source host only sends one memory page to the destination host. This may be fine for TCP/IP communication. However, it causes under-utilization of network link bandwidth when transferring pages over InfiniBand RDMA. It is more desirable to transfer multiple pages in one RDMA operation to fully utilize the link bandwidth.

Network QoS: Though RDMA avoids most of the software overheads involved in page transfer, the migration traffic contends with other applications for network bandwidth. It is preferable to explore an intelligent way that minimizes the contention on network bandwidth, while utilizing the network bandwidth efficiently.

V. Detailed Design Issues and Solutions

In this section we present our design of virtual machine migration over RDMA. We first introduce the overall design of RDMA based migration protocols in Section V-A. Then we explain in detail how we address other design challenges in the later sections.

A. *RDMA based Migration Protocols*

As we have mentioned, there are two kinds of memory pages that need to be handled during migration. Normal memory pages will be transferred to the destination host directly, and the page table pages will have to be translated to use machine independent *pfn* before being sent. Translating the page table pages will have to consume CPU cycles, while other pages can be directly sent using RDMA.

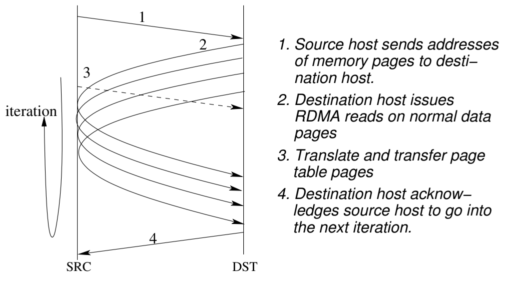
Both RDMA read and RDMA write operations can be used to transfer the memory pages. We have designed protocols based on each of them. Figure 2 is a simplified illustration of RDMA related traffic between the migration helper processes in one iteration of the pre-copy stage. Actual design uses the same concept, but is more complex due to other issues such as flow control. Our principle is to issue RDMA operations to send normal memory pages as early as possible. While the transfer is taking place, we start to process the page table pages, which requires more CPU cycles. In this way, we overlap the translation with data transfer, and achieve minimal total migration time. We use send/receive operations instead of RDMA to send the page table pages because of two reasons. First, the destination host needs to be notified when the page table pages have arrived, so that it can start translating the page tables. Using send/receive does not require explicit flag messages to synchronize between the source and destination hosts. Also, the number of page table pages is small, so most migration traffic is still transferred over RDMA.

As it can be seen, RDMA read protocol requires more work be done at the destination host while RDMA write protocol puts more burden on the source host. Thus, we dynamically select the suitable protocol based on runtime server workloads. At the beginning of the pre-copy stage, the source and destination hosts exchange load information and the node with lighter workloads will initiate RDMA operations.

B. *Memory Registration*

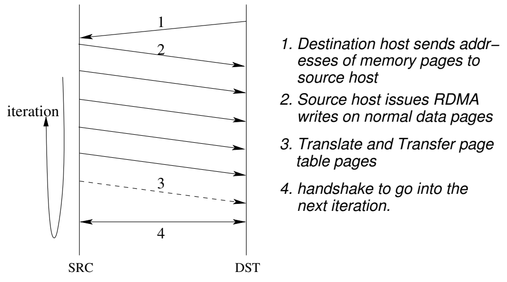
As indicated in Section IV-B, memory registration is a critical issue because none of the existing approaches, either copy based send or registration on the fly, works well here. We use different techniques to handle this issue based on different types of memory pages.

For page table pages, the migration helper processes have to parse the pages in order to translate between machine dependent *machine frame number (mfn)* and machine independent *physical frame number (pfn)*. Thus, there will be no additional cost to use a copy-base approach. On the source host, the migration helper process writes the translated pages directly to the pre-registered buffers and then the data can be sent out to the corresponding pre-registered buffers on the destination. On the destination host, the migration helper process reads the



1
2
3
4
iteration
SRC
DST
1. Source host sends addresses of memory pages to destination host.
2. Destination host issues RDMA reads on normal data pages
3. Translate and transfer page table pages
4. Destination host acknowledges source host to go into the next iteration.

(a) Migration over RDMA read



1
2
3
4
iteration
SRC
DST
1. Destination host sends addresses of memory pages to source host
2. Source host issues RDMA writes on normal data pages
3. Translate and Transfer page table pages
4. handshake to go into the next iteration.

(b) Migration over RDMA write

Fig. 2. Migration over RDMA

data from the pre-registered buffers and writes the translation results into the new page table pages.

For other memory pages there will be additional cost to use a copy based approach. And the migration helper process cannot register the memory pages belonging to the migrating VM directly. Fortunately, InfiniBand supports direct data transfer using hardware addresses in kernel space, which allows memory pages addressed by hardware DMA addresses to be directly used in data transfer. The hardware addresses are known in our case, by directly reading the page table pages (*mfn*). The only remaining issue now is that the helper processes in Xen are user level programs and cannot utilize this kernel function. We make modifications to InfiniBand drivers to extend this functionality to user level processes and hence bypass the memory registration issue. Note that this modification does not raise any security concerns because we only export the interface to user processes in the control domain (Dom0), where all programs running in this domain are trusted to be secure and reliable.

C. *Page Clustering*

In this section, we first analyze how Xen organizes the memory pages of a guest VM, and then propose a "page-clustering" technique to address the issue of network under-utilization caused by non-contiguous transfer. As shown in Figure 3, Xen maintains an address mapping table which maps machine independent *pfn* to machine dependent *mfn* for each guest VM. This mapping can be arbitrary and the physical layout of the memory pages used by a guest VM may not be contiguous. During migration, a memory page is copied to a destination memory page corresponding to the same *pfn*, which guarantees application transparency to migration. For example, in Figure 3, physical page 1 is copied to physical page 2 on the destination host, because their corresponding *pfn* are both 3. Xen randomly decides the order to transfer pages to better estimate the page dirty rate. The non-contiguous physical memory layout together with such randomness makes it very unlikely that two consecutive transfers involve contiguous memory pages so that they can be combined.

We propose *page clustering* to serve two purposes: first, to send as many pages as possible in one RDMA operation

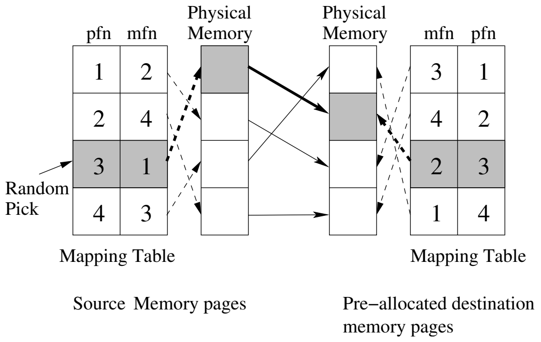
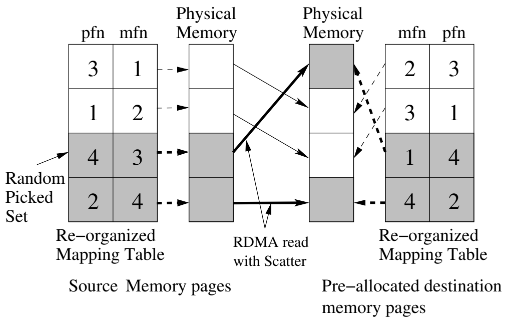


Fig. 3. Memory page management for a "tiny" OS with four pages

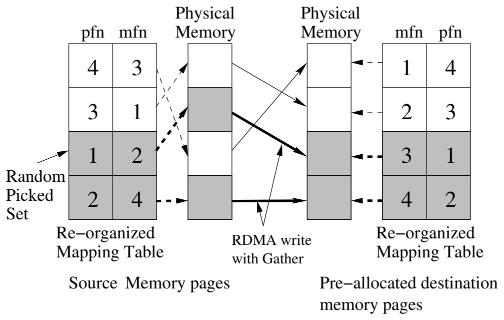
to efficiently utilize the link bandwidth; second, to keep a certain level of randomness in the transfer order for accurate estimation of page dirty rate. Figure 4(a) illustrates the main idea of *page clustering* using RDMA read. We first reorganize the mapping tables based on the order of *mfn* at the source host. Now contiguous physical memory pages correspond to contiguous entries in the re-organized mapping table. In order to keep randomness, we cluster the entries of the re-organized mapping tables into multiple sets. Each set contains a number of contiguous entries, which can be transferred in one RDMA operation under most circumstances (We have to use multiple RDMA operations in case that a set contains the non-contiguous portion of physical memory pages used by the VM). Each time we randomly pick a set of pages to transfer. As shown in the figure, with sets of size two the whole memory can be transferred within two RDMA read operations. The size of each set is chosen empirically. We use 32 in our actual implementation. Note that the memory pages on the destination host need not be contiguous, since InfiniBand supports RDMA read with scatter operation. RDMA write protocol also uses the similar idea, except that we need to reorganize the mapping tables based on the *mfn* at the destination host to take advantage of RDMA write with gather, as shown in Figure 4(b).

D. *Network Quality of Service*

By using RDMA based schemes we can achieve minimal software overhead during migration. However, the migration



(a) RDMA read



(b) RDMA write

Fig. 4. Re-organizing mapping tables for page-clustering

traffic will unavoidably consume a certain amount of network bandwidth, thus may affect the performance of other hosted communication-intensive applications during migration.

To minimize network contention, Xen uses a dynamic adaptive algorithm to limit the transfer rate of the migration traffic. It always starts from a low transfer rate limit at the first iteration of pre-copy. Then the rate limit is set to a constant increment to the page dirty rate of the previous iteration, until it exceeds a high rate limit, when Xen will terminate the pre-copy stage. Although the same scheme can be used for RDMA based migration, we would like a more intelligent scheme because RDMA provides much higher network bandwidth. If there is no other network traffic, limiting the transfer rate unnecessarily prolongs the total migration time. We want the pre-copy stage to be as short as possible if there is enough network bandwidth, but to alleviate the network contention if other applications are using the network.

We modify the adaptive rate limit algorithm used by Xen to meet our purpose. We start from the highest rate limit by assuming there is no other application using the network. After sending a batch of pages, we estimate the theoretical bandwidth the migration traffic should achieve based on the average size of each RDMA operation. If the actual bandwidth is smaller than that (the empirical threshold would be 80% of the estimation), it probably means that there are other applications sharing the network, either at the source or destination host. Then we reduce the rate of migration traffic by controlling the issuance of RDMA operations. We control the transfer rate under a pre-defined low rate limit, or a constant increment to the page dirty rate of the previous round, whichever is higher. If this rate is lower than the high rate limit, we try to raise the rate limit after sending a number of pages. If there is no other application sharing the network at the time, we will be able to achieve a full bandwidth. In this case, we will keep sending at the high rate limit. Otherwise, we will remain at the low rate limit some more time before try to raise the limit again. Because RDMA transfers require very little CPU involvement, its throughput depends mainly on the network utilization. Thus, our scheme works well to detect the network contention, and is able to efficiently utilize the link bandwidth when there is less contention on network resources.

VI. Evaluation

In this section we present our performance evaluations, which we design to address various important metrics of VM migration. We first evaluate the basic migration performance with respect to total migration time, migration downtime and network contentions. Then we examine the impact of migration on hosted applications using SPEC CINT2000 [22] and NAS Parallel Benchmarks [18]. Finally we evaluate the effect of our adaptive rate limit mechanism on network QoS.

A. Experimental Setup

We implement our RDMA based migration design with InfiniBand OpenFabrics verbs [20] and Xen-3.0.3 release [26]. We compare our implementation with the original Xen migration over TCP. To make a fair comparison, all TCP/IP related evaluations are carried over IP over InfiniBand (IPoIB [9]). Though not shown in the paper, we found that migration over IPoIB always achieves better performance than using the GigE control networks of the cluster. And in all our evaluations except in Section VI-E, we do not limit the transfer rate for either TCP or RDMA based migration.

The experiments are carried out on an InfiniBand cluster. Each system in the cluster is equipped with dual Intel Xeon 2.66 GHz CPUs, 2 GB memory and a Mellanox MT23108 PCI-X InfiniBand HCA. Xen-3.0.3 with the Linux 2.6.16.29 kernel is used on all computing nodes.

B. Basic Migration Performance

In this section we examine the basic migration performance. We first look at the effect of page-clustering scheme proposed in Section V-C. Figure 5 compares the total time to migrate VMs with different sizes of memory configurations. We compare four different schemes: migration using RDMA read or RDMA write, and with or without page-clustering. Because page-clustering tries to send larger trunks of memory pages to utilize link bandwidth more efficiently, we observe that it can constantly reduce the total migration time, up to 27% in case of migrating a 1GB VM using RDMA Read. For RDMA write, we do not see as much benefit of page-clustering as RDMA read. This is because for messages around 4KB, InfiniBand has more optimized RDMA write performance, the bandwidth improvement from sending larger messages becomes smaller. Since page-clustering constantly shows better performance, we use page-clustering in all our later evaluations.

Next we compare the total migration time achieved over IPoIB, RDMA read and RDMA write operations. Figure 6 shows the total migration time needed to migrate a VM with varied memory configurations. As we can see, due to the increased bandwidth provided by InfiniBand and RDMA, the total migration time can be reduced by up to 80% by using RDMA operations. RDMA read based migration has slightly higher migration time; this is because InfiniBand RDMA write operation typically provides a higher bandwidth.

Figure 7 shows a root-to-all style migration test. We first launch multiple virtual machines on a source node, with each using 256 MB of memory. We then start migrating all of them to different hosts at the same time and measure the time to finish all migrations. This emulates the requirements posed by proactive fault tolerance, where all hosted VMs need to be migrated to other hosts as fast as possible once the physical host is predicted to fail. We also show the migration time normalized to the case of migrating one VM. For IPoIB, there is a sudden increase when the number of migrating VMs reaches 3. This is because we have two CPUs on each physical host. Handing three migration traffic leads to contention on not only network bandwidth, but CPU resources as well. For migration over RDMA, we observe almost linear increase of the total migration time. RDMA read scales the best here because it puts the least burden on the source physical host, so that the contention on network is almost the only factor affecting the total migration time in this case.

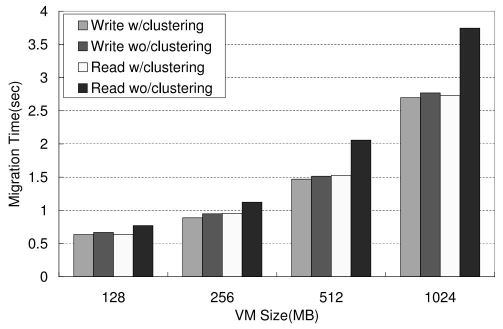


Fig. 5. Benefits of page-clustering

We have been evaluating total migration time that may be hidden from applications through live migration. With live migration, the application will only perceive the migration downtime. The migration downtime mainly depends on two factors. First is the application hosted on the migrating VM. The faster application dirties memory pages, the more memory pages may need to be sent in the last iteration of the pre-copy

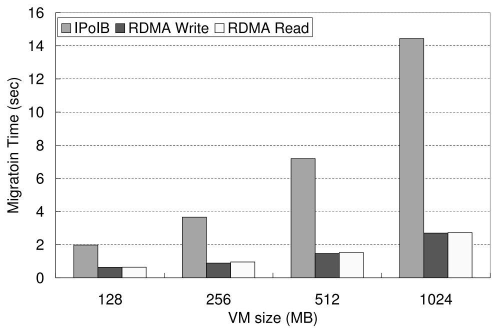


Fig. 6. Total migration time

stage, which prolongs the downtime. Second is the network bandwidth, a higher bandwidth shortens the time spent in the last pre-copy iteration, resulting in shorter downtime. To measure the migration downtime, we use a latency test. We start a ping-pong latency test over InfiniBand with 4 bytes messages between two VMs and then migrate one of the VMs. The worst round-trip latency observed during migration can be considered as a very accurate approximation of the migration downtime, because a typical round-trip over InfiniBand will take less than 10 μs.

We conduct the test while having a process continuously tainting a pool of memory in the migrating VM. We vary the size of pool to emulate applications dirtying the memory pages at different rates. Only RDMA read results are shown here because RDMA write performs very similarly. As shown in Figure 8, downtimes of migrating over RDMA or IPoIB are similar in case of no memory tainting. This is because the time to transfer the dirty pages in the last iteration is very small compared with other migration overhead such as re-initializing the device drivers. While increasing the size of the pool, we see a larger gap of the downtime. Due to the high bandwidth achieved through RDMA, the downtime can be reduced drastically, up to 77% in case of tainting a pool of 256MB memory.

In summary, due to increased bandwidth, RDMA operations can significantly reduce the total migration time and migration downtime in most cases. Low software overhead also gives RDMA extra advantages while handling multiple migration tasks at the same time.

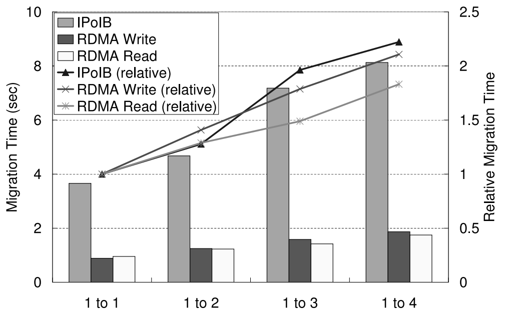


Fig. 7. “Root-to-all” migration

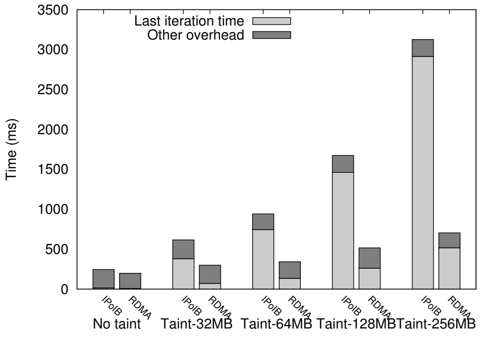


Fig. 8. Migration downtime

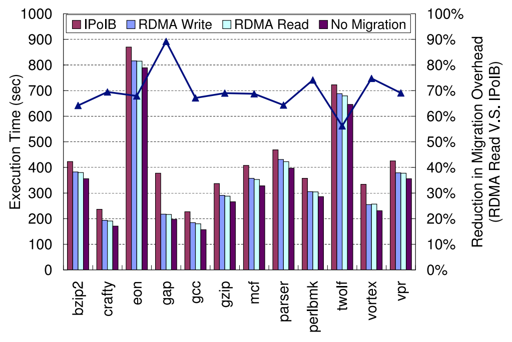


Fig. 10. SPEC CINT 2000 (1 CPU)

C. *Impact of Migration on Hosted Applications*

Now we evaluate the actual impact of migration on applications hosted in the migrating VM. We run SPEC CINT 2000 [22] benchmarks in a 512MB guest VM and migrate the VM back and forth between two different physical hosts. Because CINT is long running application, we migrate the VM eight times to enlarge the impact of migration. As we can see in Figure 9, live migration is able to hide the majority of the total migration time to the hosted applications. However, even in this case, RDMA based scheme is able to reduce the migration overhead over IPoIB by an average of 54%.

For the results in Figure 9, we have 2 CPUs on each host, providing enough resources to handle the migration traffic while the guest VM is using one CPU for computing. As we have mentioned in Section IV-A, in real production VM-based environment, we may consolidate many servers onto one physical host, leaving very few CPU resources to handle migration traffic. To emulate this case, we disable one CPU on the physical hosts and conduct the same test, as shown in Figure 10. We observe that migration over IPoIB incurs much more overhead in this case due to the contention on CPU resources, while migration over RDMA does not have much more overhead than the 2 CPU case. Compared with migration over IPoIB, RDMA-based migration reduces the impact on applications by up to 89%, an average of 70%.

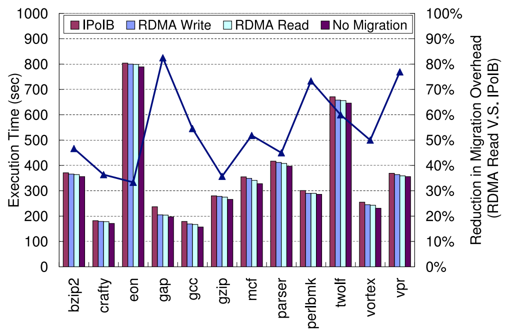


Fig. 9. SPEC CINT 2000 (2 CPUs)

Migration will affect the application performance not only on the migrating VM, but also on the other non-migrating VMs on the same physical host as well. We evaluate this impact in Figure 11. We first launch a VM on a physical node, and run SPEC CINT benchmarks in this VM. Then we migrate another VM to and from that physical host in 30 seconds interval to study the impact of migrations on the total execution time. We use one CPU in this experiment. We observe the same trend that migration over RDMA reduces the overhead by an average of 64% compared with IPoIB. Here we also show the hybrid approach. Based on server loads, the hybrid approach automatically chooses RDMA read when migrating VM out of the host and RDMA write when migrating VM in. Table 1 shows the sample counts of total instructions executed in the privileged domain, total L2 cache misses and total TLB misses during each benchmark run. For RDMA based migration we show the percentage of reductions compared to IPoIB. All of these costs are directly contributing to the overhead of migration. We observe that RDMA based migration can reduce all the costs significantly. And the hybrid scheme reduces the overhead further compared to RDMA read. RDMA write scheme, by which server has less burden when migrating VMs in but more when migrating VMs out, shows very similar number to RDMA read. Thus we omit RDMA write data for conciseness.

In summary, RDMA based migration can significantly reduce the migration overhead observed by applications hosted on both the migrating VM and the non-migrating VMs. This is especially true when the server is highly loaded and has less CPU resources to handle the migration traffic.

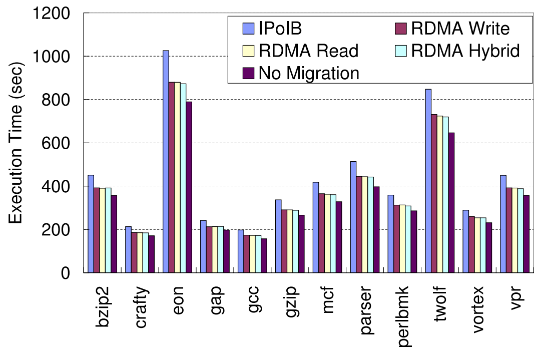
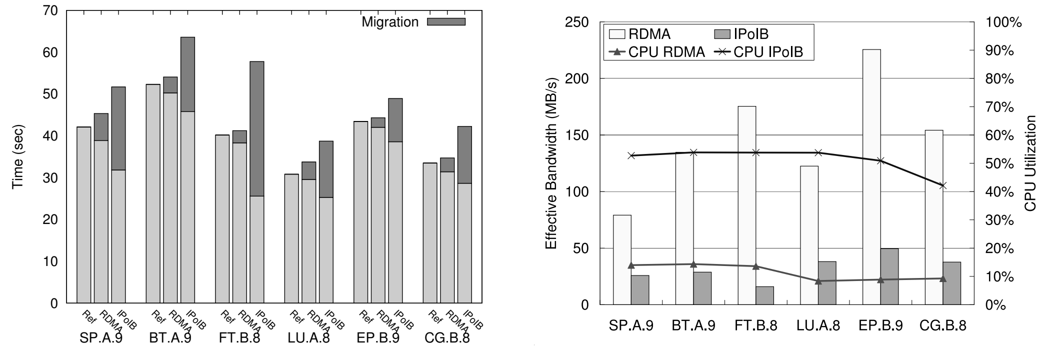


Fig. 11. Impact of migration on applications in a non-migrating VM

TABLE I

SAMPLE INSTRUCTION COUNT, L2 CACHE MISSES AND TLB MISSES (COLLECTED USING XENOPROF [13])

Profile		bzip2	crafty	eon	gap	gcc	gzip	mcf	parser	perlbmk	twolf	vortex	vpr
Inst. Count	IPoIB	24178	9732	58999	12214	9908	16898	21434	28890	19590	47804	14688	23891
	RDMA Read	-62.7%	-61.1%	-62.5%	-62.0%	-58.4%	-60.6%	-61.8%	-63.3%	-62.5%	-62.5%	-62.40%	-62.06%
	RDMA Hybrid	-64.3%	-63.6%	-65.4%	-63.7%	-59.6%	-62.6%	-63.4%	-65.5%	-63.7%	-64.7%	-64.37%	-64.19%
L2 Cache Miss	IPoIB	12372	1718	5714	3917	5359	2285	31554	8196	3523	27384	5567	16176
	RDMA Read	-10.8%	-36.9%	-56.8%	-15.4%	-13.4%	-43.6%	-3.8%	-21.7%	-28.4%	-12.6%	-13.87%	-10.10%
	RDMA Hybrid	-11.1%	-39.6%	-58.8%	-15.0%	-15.7%	-45.3%	-4.0%	-22.6%	-28.5%	-14.8%	-14.03%	-9.81%
TLB Miss	IPoIB	46784	153011	789042	27473	69309	33739	42116	59657	82135	216593	71239	67562
	RDMA Read	-69.9%	-10.2%	-11.0%	-61.9%	-19.2%	-68.1%	-69.9%	-66.1%	-33.2%	-31.1%	-28.48%	-49.78%
	RDMA Hybrid	-73.0%	-10.5%	-10.4%	-64.5%	-19.9%	-70.4%	-73.1%	-68.4%	-34.1%	-32.0%	-29.65%	-51.78%



(a) Total execution time

(b) Effective bandwidth and Dom0 CPU utilization

Fig. 12. Impact of migration on NAS Parallel Benchmarks

D. Impact of Migration on HPC Applications

Several recent work shows the feasibility of VM-based environments for HPC parallel applications [7], [16]. Thus, we also study the impact of migration on Parallel HPC applications. We conducted an experiment using NAS Parallel Benchmarks (NPB) [18], which are derived from the computing kernels common on Computational Fluid Dynamics (CFD) applications.

We use MVAPICH, a popular MPI implementation over InfiniBand [14]. The benchmarks run with 8 or 9 processes on separate VMs, each on different physical hosts and using 512MB of memory. We migrate a VM once during the execution to study the impact of migration on the total execution time, as shown in Figure 12(a). Here RDMA read is used for migration, because the destination host has lower load than the source host. As we can see, the overhead caused by migration is significantly reduced by RDMA, an average of 79% compared with migration over IPoIB. We also mark out the total migration time, which is not directly reflected in the increase of total execution time because of live migration. We observe that IPoIB has much longer migration time due to the lower transfer rate and the contention on CPU resources. In HPC it is very unlikely that people will spare one CPU for migration traffic. Thus we use only one CPU on each host in this experiment. As a result, the migration overhead here for TCP/IPoIB is significantly higher than reported by other relevant studies as in [8], [16].

Figure 12(b) further explains the gap we observed between migration over IPoIB and RDMA. As we can see, while migrating over IPoIB, the migration helper process in Dom0 uses up to 53% of the CPU resources but only achieves an effective migration throughput up to 49MB/s (calculated by dividing the memory footprint of the migrating OS by the total migration time). Migrating over RDMA, in contrast, is able to deliver up to 225MB/s while using a maximum of 14% of the CPU resources.

E. Impact of Adaptive Limit Control on Network QoS

In this section we demonstrate the effectiveness of our adaptive rate limit mechanism described in Section V-D. We set the high limit of page transfer rate to be 300 MB/s and the low limit to be 50 MB/s. As shown in Figure 13, we first start a bi-directional bandwidth test between two physical hosts, where we observe around 650MB/s throughput. At the 5th second, we start to migrate an 1GB VM between these two hosts. As we can see, the migration process first tries to send pages at the higher rate limit. However, because of the bi-directional bandwidth test, it is only able to achieve around 200 MB/s, which is less than the threshold (80%). The migration process then detects the network contention and starts to send pages at the lower rate. Thus, from the Bidirectional bandwidth test we observe an initial drop, but very quickly the throughput comes back to 600MB/s level. The migration process tries to get back to the higher rate several times between the 5th and the 15th seconds, but immediately detects that there is still network contention and remains at the lower rate. At the 15th second we stop the bandwidth test, after that the migration traffic detects that it is able to achieve a reasonable high bandwidth (around 267 MB/s), thus keeps sending pages at the higher rate.

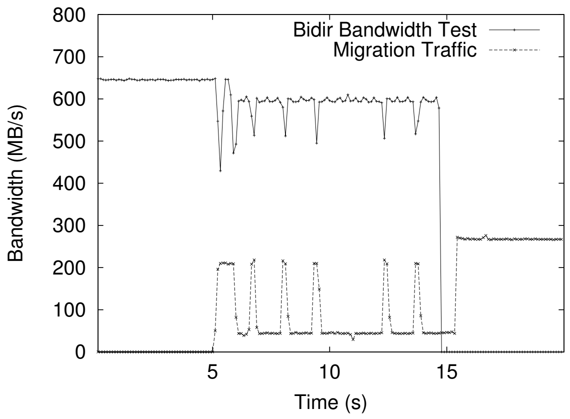


Fig. 13. Adaptive rate limit control

VII. Related Work

In this paper we discussed improving virtual machine migration using RDMA technologies. Our work is built on top of Xen live migration [3]. Other popular virtual machine technologies include VMware workstation [24] and VMware ESX server [25]. VMware also supports guest OS migration through VMotion [19]. Though the source code is unavailable, from published documents we believe that they use similar approaches as Xen. Thus, our solution can be applicable in this context also.

Our work complements industry and research efforts that use VM migration to support data-center management, such as VMware VirtualCenter [24] and Xen Enterprise [26]. Also, with the low overhead of Xen para-virtualization architecture, researchers have been studying the feasibility of High Performance Cluster [7], [8] or Grid Computing [5] with virtual machines. Mueller et al. [16] have proposed proactive fault tolerance for HPC based on VM migration. Our work can seamlessly benefit those efforts that the migration can take advantage of high speed interconnects, which leads to much better efficiency in their proposed solutions.

Travostino et al. [23] studied VM migration over MAN/WAN. And Nakashima et al. [17] applied RDMA mechanisms to VM migration over UZURA 10 Gb Ethernet-NIC. Even though our general approach (optimizing memory page transfer over new network technologies) is similar to theirs, our work is different in multiple aspects. First, we work on OpenFabrics Alliance (OFA) [20] InfiniBand stack, which is an open standard and is widely used. The detail design challenges differ and we believe that our work is general enough to be applied to more computing systems environments. Second, we address extra optimization issues such as page clustering and network QoS. Finally, we design thorough evaluations at both micro-benchmarks and application-level benchmarks. Besides total migration time and observed application downtime, we focus on various important metrics reflecting the requirements of VM migration posed by real world usage scenarios.

Our work aims to benefit VM migration by using the OS-bypass and one-sided feature of RDMA. Exploiting the benefits of RDMA has been widely studied in communication subsystems, file systems, or storage areas [11], [12], [28]. Compared to these works, we work in a specialized domain that minimizing system resource consumption is critical and the migration process handles data that belongs to other running OS instances, which leads to different research challenges.

Several studies have suggested that TCP-offload engines can effectively reduce the TCP processing overhead [1], [6], [27]. VM migration can benefit from these technologies. Because of the two sided synchronous model of the socket semantics, however, we still cannot avoid frequent context switches between the migrating domain and the control domain which hosts the migration helper process. Thus, we believe RDMA will still deliver better performance in handling migration traffic. The detailed impact of TCP-offload engines is an interesting topic and will be one of our future research directions.

VIII. Conclusions and Future Work

In this paper, we identify the limitations of migration over the TCP/IP stack, such as lower transfer rate and high software overhead. Correspondingly, we propose a high performance virtual machine migration design based on RDMA. We address several challenges to fully exploit the benefits of RDMA, including efficient protocol designs, memory registration, non-contiguous transfer and network QoS. Our design significantly improves the efficiency of virtual machine migration, in terms of both total migration time and software overhead. We evaluate our solutions over Xen and InfiniBand through a set of benchmarks that we design to measure the important metrics of VM migration. We demonstrate that by using RDMA, we are able to reduce the total migration time by up to 80%, and migration downtime by up to 77%. We also evaluate the impact of VM migration on hosted applications. We observe that RDMA can reduce the migration cost on SPEC CINT 2000 benchmarks by an average of 54% when the server is lightly loaded, and an average of 70% on a highly loaded server.

In future, we will continue working on exploiting the benefits of high speed interconnects for VM management. We will explore more intelligent QoS schemes to further reduce the impact of VM migration on the physical host, e.g., taking advantages of hardware QoS mechanisms to reduce the contention on network traffic. We plan to analyze in detail the impact of TCP-offload engines on virtual machine migration traffic. Also, based on current work, we plan to explore virtual machine save/restore over remote memory to benefit fault-tolerance frameworks depending on such functionalities.

Acknowledgments

This research is supported in part by an IBM PhD Scholarship, and the following grants and equipment donations to the Ohio State University: Department of Energy's Grant #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation grants #CNS-0403342 and #CCF-0702675; grants from Intel, Mellanox, Sun, Cisco, and Linux Networx; and equipment donations from Apple, AMD, IBM, Intel, Microway, Pathscale, Silverstorm and Sun.

References

[1] Annie P. Foong, Thomas R. Huff, Herbert H. Hum, Jaidev P. Patwardhan, and Greg J. Regnier. TCP performance re-visited. In *Proceedings International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, TX, March 2003.

[2] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, 15(4):412–447, 1997.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *Proceedings of 2nd Symposium on Networked Systems Design and Implementation*, 2005.

[4] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, Oct. 2003.

[5] R. Figueiredo, P. Dinda, and J. Fortes. A Case for Grid Computing on Virtual Machines. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS), May 2003.*, 2003.

[6] D. Freimuth, E. Hu, J. LaVoie, R. Mraz, E. Nahum, P. Pradhan, and J. Tracey. Server Network Scalability and TCP Offload. In *USENIX 2005*, 2005.

[7] W. Huang, J. Liu, B. Abali, and D. K. Panda. A Case for High Performance Computing with Virtual Machines. In *International Conference on Supercomputing (ICS)*, 2006.

[8] W. Huang, J. Liu, M. Koop, B. Abali, and D. Panda. Nomad: Migrating OS-bypass Networks in Virtual Machines. In *the 3rd ACM/USENIX Conference on Virtual Execution Environment (VEE'07)*, June 2007.

[9] IETF IPoIB Workgroup.
http://www.ietf.org/html.charters/ipoib-charter.html.

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

[11] J. Liu, D. K. Panda, and M. Banikazemi. Evaluating the Impact of RDMA on Storage I/O over InfiniBand. In *SAN-03 Workshop (in conjunction with HPCA)*, Feb. 2004.

[12] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *17th Annual ACM International Conference on Supercomputing*, June 2003.

[13] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel. Diagnosing Performance Overheads in the Xen Virtual Machine Environment. In *Proceedings of the 1st ACM/USENIX Conference on Virtual Execution Environments (VEE'05)*, June 2005.

[14] MVAPICH Project Website. http://mvapich.cse.ohio-state.edu.

[15] Myricom, Inc. Myrinet. http://www.myri.com.

[16] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive Fault Tolerance for HPC with Xen Virtualization. In *Proceedings of the 21st Annual International Conference on Supercomputing (ICS'07)*, Seattle, WA, June 2007.

[17] K. Nakashima, M. Sato, M. Goto, and K. Kumon. Application of RDMA Data Transfer Mechanism over 10Gb Ethernet to Virtual Machine Migration. IEICE technical report. Computer systems,Vol.106, No.287(20061006) pp. 1-6 (In Japanese).

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

[19] M. Nelson, B.-H. Lim, and G. Hutchins. Fast Transparent Migration for Virtual Machines. In *Proceedings of USENIX 2005*, Anaheim, California.

[20] Open Fabrics Alliance. http://www.openfabrics.org.

[21] Quadrics, Ltd. QsNet. http://www.quadrics.com.

[22] SPEC CPU 2000 Benchmark. http://www.spec.org/.

[23] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath, and P. Y. Wang. Seamless live migration of virtual machines over the MAN/WAN. *Future Gener. Comput. Syst.*, 22(8):901–907, 2006.

[24] VMware. http://www.vmware.com.

[25] C. Waldspurger. Memory resource management in VMware ESX server. In *the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[26] XenSource. http://www.xensource.com/.

[27] H. youb Kim and S. Rixner. TCP Offload through Connection Handoff. In *Proceedings of EuroSys 2006*, Leuven, Belgium, April 2006.

[28] W. Yu, S. Liang, and D. K. Panda. High Performance Support of Parallel Virtual File System (PVFS2) over Quadrics. In *International Conference on Supercomputing (ICS-05)*, 2005.