Inter-rack Live Migration of Multiple Virtual Machines

Umesh Deshpande, Unmesh Kulkarni, and Kartik Gopalan
Computer Science, Binghamton University
{udeshpa1, ukulkar1, kartik}@binghamton.edu

ABSTRACT

Within datacenters, often multiple virtual machines (VMs) need to be live migrated simultaneously for various reasons such as maintenance, power savings, and load balancing. Such mass simultaneous live migration of multiple VMs can trigger large data transfers across the core network links and switches, and negatively affect the cluster-wide performance of network-bound applications. In this paper, we present a distributed system for inter-rack live migration (IRLM), i.e., parallel live migration of multiple VMs across racks. The key performance objective of IRLM is to reduce the traffic load on the core network links during mass VM migration through distributed deduplication of VMs' memory images. We present an initial prototype of IRLM that migrates multiple QEMU/KVM VMs within a Gigabit Ethernet cluster with 10GigE core links. We also present preliminary evaluation on a small testbed having 6 hosts per rack and 4 VMs per host. Our evaluations show that, compared to the default live migration technique in QEMU/KVM, IRLM reduces the network traffic on core links by up to 44% and the total migration time by up to 26%. We also demonstrate that network-bound applications experience a smaller degradation during migration using IRLM.

Categories and Subject Descriptors

D.4.0 [**Operating Systems**]:

General Terms

Design, Experimentation, Performance

Keywords

Virtual Machines, Operating Systems, Live Migration

1. INTRODUCTION

Virtual machines (VMs) have become one of the basic building blocks of datacenters due to cost savings, elasticity, and ease of administration. They are used to provide Infrastructure-as-a-Service (IaaS) to support cloud computing and high performance-computing (HPC) applications [17, 7]. Day to day operation of datacenters relies upon live migration of VMs to deal with load spikes during peak hours, to save power by packing VMs into fewer physical hosts, or to perform maintenance of the physical hosts. The scale of live migration can range from migrating VMs across only a few physical machines to entire racks of physical machines. Live migration is a network intensive activity that involves the transfer of tens to hundreds of gigabytes of memory over the network (assuming the use of network storage, which does not require migration). This traffic can overload the core network links and switches within the datacenter Ethernet and degrade the performance of other network-bound applications whose packets traverse the core links. Live migration traffic also consumes the bandwidth at the source and target network interfaces and competes with the bandwidth requirements of applications running within the VMs. On one hand, an administrator may want to maintain application-level quality of service of VMs that are not being migrated by rate-limiting the VM migration traffic. On the other hand, the administrator may also wish to minimize the total migration time of VMs by increasing the bandwidth available to VM migration. *Both objectives can be aided if we can reduce the total amount of data transmitted during live VM migration.*

Existing techniques to reduce the data transferred during live migration either focus on optimizing the migration of a single VM in isolation [6, 16, 11, 13] or groups of VMs running on the same physical machine [8]. In this paper, we address the problem of optimizing *inter-rack live migration* (IRLM) of multiple VMs, i.e., simultaneously live migrating VMs from one rack of machines to another rack. IRLM may be performed for reasons such as load-balancing, system maintenance, or powering down an entire rack to save energy. Generally, VMs communicating with each other are placed within the same rack to reduce communication overhead (by traversing fewer switches) and also to minimize the performance impact on other VMs in the datacenter [12, 20]. VM proximity can be maintained by choosing the same destination rack, when the entire rack is migrated.

Reducing the data transferred by IRLM is important because inter-rack network traffic is more likely to traverse the core links of the datacenter Ethernet where network overload is more likely to produce an adverse cluster-wide impact (as opposed to congestion at the edge links).

The main insight to reducing VM migration traffic in general is that VMs within a cluster (or even within a single machine) often execute the same operating system, libraries, and sometimes even applications. Thus, their memory images are likely to contain significant number of pages having identical content. This observation could be used to avoid transmitting duplicate pages across the network – a concept generally known as *deduplication*. An existing technique called live gang migration [8] (our prior work) exploits deduplication among VMs within a single physical machine. However it does not deduplicate pages when the VMs to be migrated are executing on different physical machines.

In this paper, we present our ongoing work on a distributed system for IRLM. Our initial prototype exploits page-level *distributed deduplication* across VMs running on different physical machines in the rack in order to reduce VM migration traffic on the core network links. This reduces the degradation experienced by network-bound applications in the entire cluster. We describe the prototype implementation of IRLM on the QEMU/KVM [14] platform and compare it against two techniques – the QEMU/KVM's default live migration technique, which we call *online compression*, and gang migration [8], which performs deduplication among VMs within a single host. We also present a preliminary evaluation of our IRLM prototype on a high-bandwidth low-latency Gigabit Ethernet having 10GigE core links and 1Gbps edge links. When compared against online compression, IRLM reduces the network traffic overhead on the core links by up to 44% and the total migration time by up to 26% for a configuration of 6 machines per rack and 4 VMs per machine. When compared against gang migration for the same configuration, IRLM reduces the network traffic on the core links by 17% and increases the total migration time by 7% (as a result of synchronization costs). Our ongoing work focuses on further reducing the data transferred and the total migration time through a number of optimizations.

The rest of the paper is organized as follows. Section 2 describes the design and implementation of IRLM. Section 3 presents a preliminary evaluation comparing IRLM against online compression and gang migration. Section 4 compares IRLM with related work. Section 5 concludes with a summary of contributions.

2. DESIGN AND IMPLEMENTATION OF IRLM

In this section, we describe the high-level architecture and low-level implementation details in the operation of IRLM.

2.1 Typical Cluster Layout

Figure 1 shows a typical layout of racks, machines, and switches in a datacenter cluster. A rack consists of a stack of tens of physical machines, each connected to an edge switch, known as the *top-of-the-rack* switch. One or more core switches, connect the top-of-the-rack switches via high-

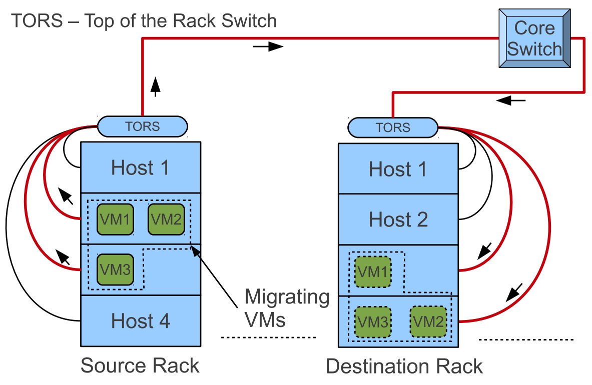

Figure 1: Layout of racks in the datacenters.

bandwidth low-latency links (typically 10Gbps or higher) to form a tree-like hierarchy. When the VMs are migrated from one rack to another, their memory pages traverse the core links before reaching the destination rack. The path is shown in the Figure 1 with a thick red line. This produces substantial network traffic overhead on the core links, which can adversely affect the performance of other network-bound applications whose traffic shares these links. Although other cluster layouts, such as a fat-tree architecture, are not explicitly considered in this paper, they can also potentially benefit from the use of IRLM.

2.2 Architecture of IRLM

Memory images of multiple migrating VMs often contain duplicate instances of applications, libraries, or operating system pages, which we refer to as *identical pages*. Eliminating the transfer of these duplicate pages can yield significant reduction in network traffic overhead over the core links. The *key challenge* then is to design an effective mechanism that can identify, track, and avoid the transmission of these identical pages which are spread across physical machines.

Figure 2 shows the architecture of the IRLM. IRLM incorporates a distributed deduplication mechanism to identify and track identical pages across VMs in a rack. During migration, only one copy of each identical page is transferred from the source rack to the target rack. At the target rack, identical pages are exchanged among the destination hosts, allowing VMs on the receiving host to reproduce their own copies of identical pages. Considering the aforementioned typical layout of datacenters, deduplication can not only relieve the core links of unnecessary traffic but also speed up the overall live migration process.

Goal of IRLM is to reduce the load on core network links by optimizing the transfer of group of VMs across the racks. We accomplish this by combining pre-copy VM migration technique with host-level and rack-level deduplication of VM memory contents.

2.3 QEMU/KVM-specific Aspects

We implemented a prototype of IRLM in the QEMU/KVM virtualization environment. Our implementation is completely transparent to the users of the

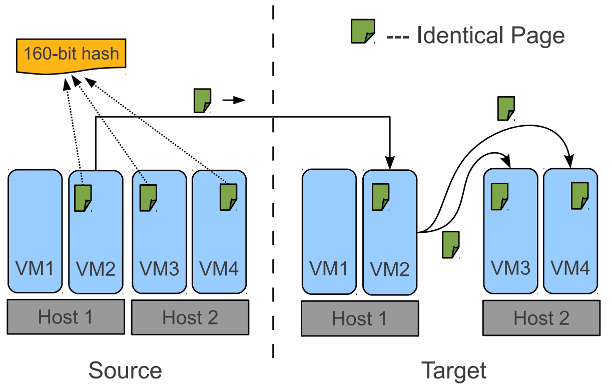

Figure 2: Architecture of inter-rack live migration.

VMs. In QEMU/KVM environment, each VM is spawned as a process on a host machine. According to the specified memory size, a part of the virtual address space of the QEMU/KVM process is exported to the VM as its physical memory. We also allocated a shared memory region at the source and the target hosts, which is accessible to each QEMU/KVM process on the same host. The shared memory contains a hash table, which is used for performing local deduplication of VM memory. Since the shared memory region is outside the physical memory region of the VMs in the QEMU/KVM process's address space, it is not accessible from within the VMs and hence poses no security vulnerabilities.

IRLM consists of two phases, namely the preparation phase and the migration phase.

2.4 Preparation Phase

This phase is carried out even before initiating the migration of VMs. The purpose of this phase is to identify the duplicate content, at the granularity of a page-level, across the VMs residing on the source rack. We use content hashing to detect identical pages and deduplicate them. The pages having the same content have the same hash value. When the hashing is performed using standard 160-bit SHA1 hash [10], the probability collision is less than the probability of a memory error, or an error in a TCP connection. [5].

To identify identical pages among VMs within each physical machine, a controller process co-ordinates the hashing (and later migration) of co-located VMs. It instructs the QEMU/KVM process associated with each co-located VM to scan the VM memory image to perform content based hashing. This allows the controller to maintain a record of identical pages within the host. To identify the identical pages that were dirtied by the VM, either during the hashing, or after its completion, we enabled dirty logging mode of QEMU/KVM. In this mode, all VM pages are marked as read-only in the shadow page table maintained by the hypervisor, and any write attempt to a VM page results into a trap into the hypervisor. The hypervisor, KVM in our case, marks the faulted page as dirty into its dirty bitmap and allows the write access to the page. Control is returned back to the writing process, and execution proceeds normally henceforth. QEMU/KVM, a user level process, uses an ioctl to extract dirty bitmap from the KVM, which we use during the migration to identify the modified pages.

To identify the identical pages among VMs across physical machines, we run an index server within the source rack. Index server is analogous to the host-level controller except at the rack level. It maintains a hash table, which is populated by carrying out a rack-wide comparison of 160-bit hash values pre-computed by the controllers at the host level. Each hash value is associated with a list of hosts containing pages of that hash value. After inserting a hash into the hash table, the index server replies to the controller with an index value that indicates the position of that 160-bit hash value in the hash bucket, i.e. $(bucketid, index)$. This identifier is used during the migration to identify an entry created for a unique page-size content.

2.5 Migration Phase

When the VMs are ready for migration, all member VMs are migrated in parallel to the destination site. The pre-computed hashing information is used to carry out the deduplication of the pages transferred at both the host and the rack levels.

Source End.

At the source side, before transferring a page QEMU/KVM process queries the index server to acquire its status. If it hasn't already been transferred by any other VM, it is transferred to the destination along with its $(bucketid, index)$ identifier. At the destination, the page is copied into the target hash table contained inside a shared memory region at the $(bucketid, index)$ location. The source side also acquires a list of target hosts that need a page of identical content, so that at the target rack, the page can be forwarded to the other target hosts. Upon transfer the page is marked as *sent* into the local hash table. QEMU/KVM process periodically updates the index server with the sent pages. For subsequent instances of the same page from any host, only a $(bucketid, index)$ identifier indicating its position in the target physical machine is transferred. Unique pages and dirty pages that have no match among the co-located VMs and even across the hosts, are transferred in their entirety to the destination.

Target End.

The target end consists of a target controller and per-VM QEMU/KVM processes. The target side controller is responsible for inter-target transfer of identical pages. After initiating the migration, each QEMU/KVM process allocates a memory region for its respective VM where incoming pages are copied. In addition to VMs' memory regions, a shared memory region is created to contain a hash table, which is shared with the other QEMU/KVM processes on the same host and the target side controller. Upon reception of an identical page, its identifier, and corresponding target host list, target end QEMU/KVM process copies the received page into the VM's memory and also inserts it into the target hash table at the $(bucketid, index)$ location received as its identifier. The target controller reads the identical page from the hash table and forwards it to the other target hosts on the list. If only a $(bucketid, index)$ identifier is received, a page corresponding to the received identifier is found from the target hash table and copied into the VM's memory. Unique and dirty pages are directly copied into the VMs' memory space.

The $(bucketid, index)$ identifiers of the pages that were transferred by other hosts are accompanied by a *remote* flag. Such pages become available to the waiting hosts only after they are forwarded by the carrying host. Therefore, instead of searching for such pages into the target hash table immediately upon reception of an identifier, the identifier, and the address of the page are inserted into a per-host waiting list. A per QEMU/KVM process thread, called as *remote thread* periodically traverses the waiting list and checks for each entry if the page corresponding to $(bucketid, index)$ identifier has been added into the target shared memory. The received pages are copied into the memory of the respective VMs after removing the entry from the list. Upon reception of a recent copy of the page whose entry happens to be on the waiting list, the corresponding entry is removed from the list to prevent the thread from over-writing the page with its stale copy.

3. EVALUATION

In this section we present the evaluation of IRLM. We compare IRLM against following VM migration techniques.

- **Online Compression (OC):** This is the default VM migration technique used by QEMU/KVM. It compresses the VM pages filled with uniform content during their migration. This mainly includes zero pages. Only one byte representing an entire page is transferred to reduce the amount of data transferred. At the target, such pages are uncompressed by filling an entire page with the same byte. Other pages are transferred normally to the destination.

- **Gang Migration (GM):** This technique deduplicates pages across the memories of VMs located on the same host using content-based hashing. Only one instance of pages having identical content is transferred from the source host. At the target host each VM reproduces its instance of the identical pages from this single copy.

We have implemented an IRLM prototype on the QEMU/KVM virtualization platform. We perform the evaluation of IRLM in a cluster testbed having high bandwidth low latency Gigabit Ethernet interconnect. The cluster consists of 13 physical hosts, each equipped with two Quad core 2GHz CPUs, 16GB of memory, and 1Gbps Network card. 6 hosts are used as sources, other 6 as targets, and 1 machine as an index server. Figure 3 shows the layout of the cluster testbed used for VM migration. Source and target machines are connected to two different top-of-the-rack Ethernet switches. The top-of-the-rack switches are connected to each other by a 10GigE optical link, which acts as the core link. To carry out the experiment, we initiate the migration of all the VMs on the source hosts at the same time. VM pages transmitted from the source side traverse the 10GigE optical link between the two switches to reach the target

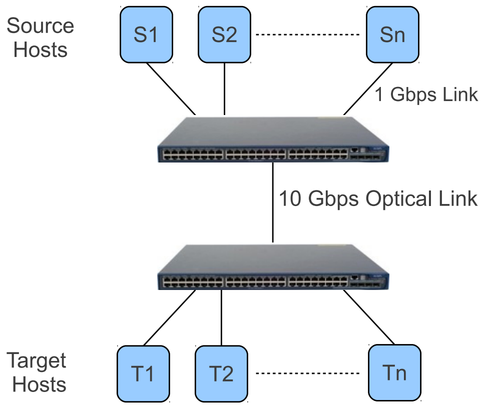

Figure 3: Layout of testbed used for evaluation.

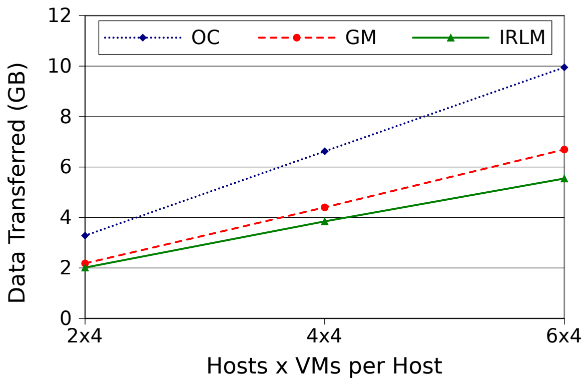

Figure 4: Amount of data transferred for idle VMs.

hosts. For all the following experiments, we fix the number of VMs per host to four, and configure each VM with 2 virtual CPUs (VCPUs) and 1GB of memory. In the following sections, we evaluate various aspects of IRLM and demonstrate that it yields significant performance improvements over both online compression and gang migration.

3.1 Amount of Data Transferred

Figure 4 shows the amount of data transferred over the core link for the three VM migration schemes with increasing number of hosts, each host running four 1GB idle VMs. Since online compression only optimizes the transfer of uniform pages, a set that mainly consists of zero pages, it transfers the highest amount of data. Gang migration deduplicates both zero and non-zero pages across co-located VMs. As a result, it sends less data than online compression. IRLM yields the best results for the amount of data transferred. Also, the difference between gang migration and IRLM increases with number of hosts due to deduplication of pages across hosts. This indicates that the performance gap between IRLM and others is likely widen as the number of physical machines per rack increases. For 6 hosts, IRLM shows more than 44% and 17% decrease in the data transferred through the core link over online compression and gang migration respectively.

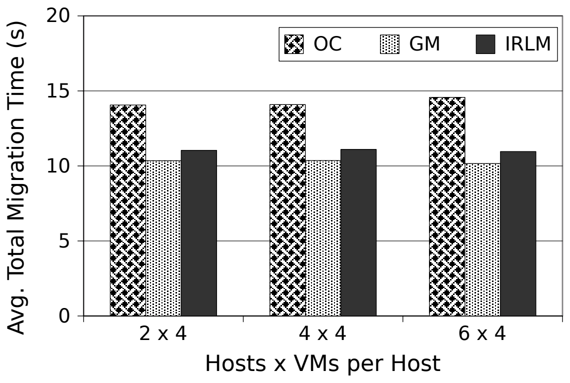

Figure 5: Total migration time for idle VMs.

3.2 Total Migration Time

To measure the total migration time of different migration techniques, we calculate an average of individual total migration times of VMs. From Figure 5, online compression has highest total migration time for any number of hosts, which is proportional to the amount of data it transfers. IRLM's total migration time is slightly higher than that of gang migration, approximately 7% higher for 6 hosts. The difference between the total migration time of IRLM and gang migration can be attributed to the overhead associated with IRLM for performing deduplication across hosts. While the migration is in progress, it queries with the index server to read, or update the status of deduplicated pages. Such requests need to be sent frequently to perform effective deduplication.

3.3 Application Degradation

This section evaluates the degradation of applications running inside the VMs. We initiate the applications inside VMs and perform migration to observe its effect on the performance of the applications. Table 1 shows the comparative evaluation of the applications with various techniques of migration. For the following experiments, we use 6 x 4 configuration of VMs.

NFS I/O Benchmark: We host a NFS server on a machine located outside the source rack and connected to the switch with 1Gbps Ethernet link. Each VM mounts a partition from the NFS server and runs a 100MB sequential file write benchmark. We carry out the migration of VMs while the benchmark is in progress, and observe the effect of migration on the performance of the benchmark. Since, at the source network interface, the NFS traffic interferes with the migration traffic, the benchmark shows degradation proportional to the amount of data the migration technique transfers. Table 1 shows the NFS write bandwidth per VM. IRLM yields the smallest reduction in observed bandwidth among the three.

TCP_RR: TCP_RR is a synchronous TCP request-response test. For Netperf TCP_RR test, we use 12 VMs from 3 hosts as senders, while other 12 VMs from other 3 hosts as receivers. We migrate the VMs while the test is in progress and measure the performance of TCP_RR. Figures in Table 1 show the average of transaction rate per sender

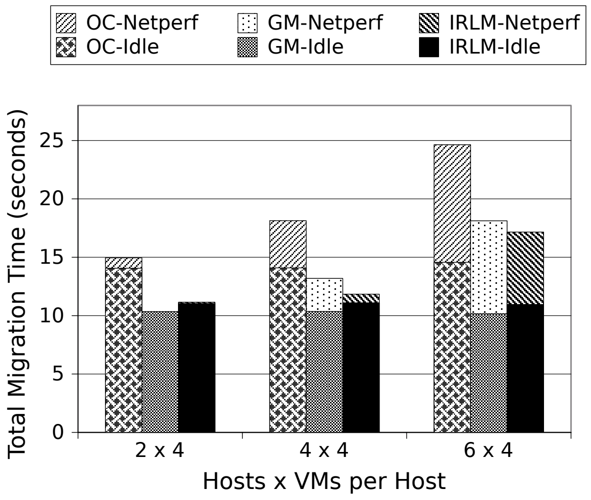

Figure 6: Total migration time with Netperf UDP_STREAM background traffic.

VM. It can be seen that IRLM least affects the performance of TCP_RR and gives the highest number of transactions per second among the three.

3.4 Degradation of Background Applications

Here we evaluate the effect of the migration of group of VMs on the network-bound applications running on the source rack. By saturating the 10Gbps optic fiber link between the switches with VM migration traffic and application's network traffic, we observe its effect on the performance of VM migration and the application. For this experiment, we use 14 more machines, 7 on the source rack, and remaining 7 on the target rack. Netperf [3] is run on each of the 7 machines on the source rack to send TCP/UDP packets to their corresponding target machines on the other rack. We migrate varying number of VMs while the 10GigE optical link is carrying 7Gbps of Netperf traffic in the background.

Netperf UDP_STREAM background traffic: Figure 6 shows the comparison of total migration time with and without UDP background traffic (idle VMs) for the aforementioned setup. We migrate 8 VMs from 2 hosts (2 x 4), in addition to 7Gbps of traffic from the UDP flows in the background. Since the link remains unsaturated with less than 10Gbps of traffic, all flows can transfer the data at full strength, and no variation from its idle mode is observed. With higher number of VMs, i.e. with 4 x 4, and 6 x 4 configurations, source rack generates more than 10Gbps of inter-link traffic. Congestion at the outgoing port of optical link results in loss of packets. In presence of UDP flows, due to loss of their packets, TCP flows, i.e. VM migration sessions, backoff. The backoff is proportional to the amount of data each VM migration technique transfers. Consequently, IRLM performs better than gang migration and online compression in terms of total migration time.

Figure 7 shows the effect of VM migration on the performance of Netperf. As mentioned above, 2 x 4 configu-

Benchmarks	W/o Migration	OC	GM	IRLM
NFS (Mbps per VM)	40.93	32.16	33.92	39.36
TCP-RR (trans/sec)	1270	307.85	399.29	454.65

Table 1: Application degradation during the VM migration

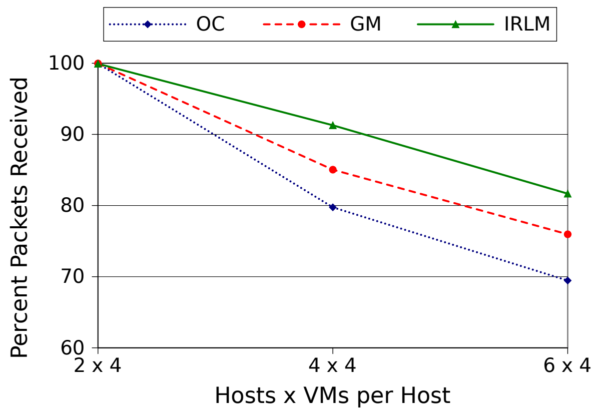

Figure 7: Percent packets received at the target with Netperf UDP_STREAM.

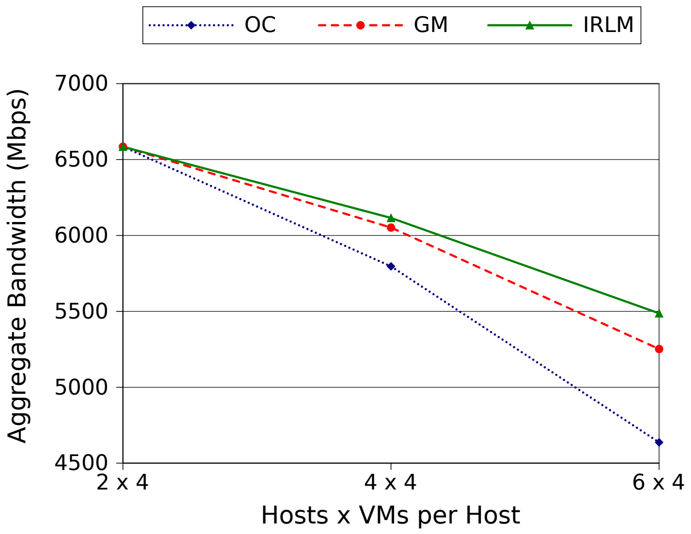

Figure 8: Aggregate bandwidth of 7 Netperf TCP_STREAM background flows.

ration doesn't saturate the optical link, therefore in an uncongested network all UDP packets from the source reach the target. With increasing number of migrating VMs, UDP packet drop increases. For 6 hosts, IRLM receives 5.7% more UDP packets than gang migration, and 11% more packets than online compression.

Netperf TCP_STREAM background traffic: From Figure 8, with the migration of VMs from more than 4 hosts, and Netperf TCP background traffic, we notice the similar performance impact on the Netperf flows that we noticed with UDP background traffic. However, due to congestion control algorithm of TCP, Netperf flows backoff by reducing their transmission rates. This reduces the packet drop at the switch, hence the congestion. Therefore, VM migration doesn't suffer as much as it suffers for UDP flows. From Figure 9, it can be seen that the total migration time with TCP Netperf traffic is almost same as that of with no background traffic. However, the backoff of TCP flows is lesser for IRLM than other two techniques. For 6 hosts, with IRLM, background TCP flows transfer with 4% and 18% higher bandwidth than gang migration and online compression respectively.

3.5 Overhead of the Preparation Phase

Gang migration and IRLM carry out the intra-host deduplication by calculating 160-bit SHA1 hash for each VM page before the migration. Dirty logging is also enabled to track any modification to the VM pages. In this section we evaluate the CPU overhead of the hashing and dirty logging on the applications running inside the VMs. In order to measure the impact of hashing, in each VM, we run an instance of the sum of subsets program (a CPU bound NP-complete algorithm) alongside the hashing process. To measure the impact of dirty logging, we run a per-VM memory write-intensive program which allocates a definite amount of memory, and writes randomly into it. In both experiments, we record the average completion time for the applications, and compare it to their completion times in absence of hashing and dirty logging disabled.

VMs	Percent Overhead	
	CPU Intensive	Write Intensive
4	0.34	1.99
8	5.85	3.93

Table 2: Overhead of the preparation phase of IRLM with varying number of VMs with CPU and memory write-intensive VM workloads.

Table 2 shows the percent difference of the hashing overhead and dirty logging overhead with varying number of VMs. Since the host has 8 CPUs, with 4 VMs each configured with 1 VCPU, the hashing thread, and the VCPUs of VMs are scheduled on different physical CPUs. Therefore minimal overhead is observed on the CPU-bound, or memory write-intensive applications. With 8 VMs the overhead increases, since the VCPUs, and the hashing contend for the physical CPU. However, we have assigned lower priority to the hashing thread, therefore it only uses idle CPU cycles, allowing the VCPUs of VMs to run with higher priority.

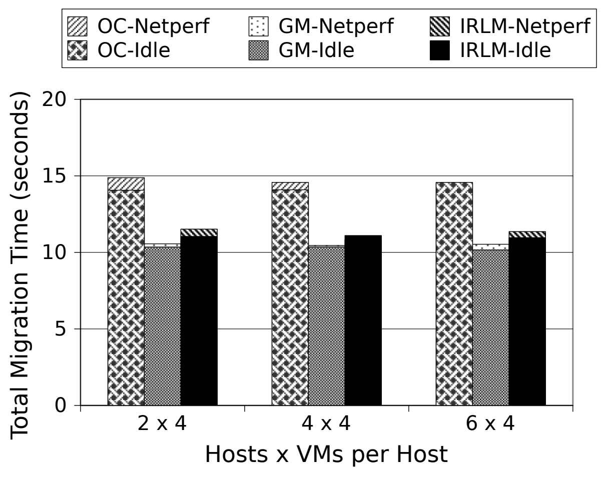

Figure 9: Total migration time of VMs with Netperf TCP_STREAM background traffic.

4. RELATED WORK

In this section we review related research work under the categories of content-based page sharing and optimized VM migration techniques.

Content-based Page Sharing: Content-based page sharing was originally studied in the context of reducing the memory footprint in Disco [4] and later in [21, 15, 2, 23, 9]. Satori [15] exploits reading data from a block device to detect and share pages in para-virtualized guests in the Xen environment. Kernel Samepage Merging (KSM) [2] detects and shares the identical memory regions of two already running processes in the Linux kernel. QEMU/KVM can be configured with KSM so that co-located VMs (running as user-level processes) could share identical memory regions. Difference Engine [9] proposes memory savings through subpage-level sharing and in-memory compression across VMs residing on the same physical host. All the above techniques reduce memory consumption either within a single VM or between multiple co-located VMs. In contrast, our work reduces bottleneck link traffic due to VM migration through page-level deduplication across a group of VMs on different hosts in a rack.

Optimized VM migration techniques: [19] optimizes the non-live migration of a single VM by using content hashing to detect blocks within the VM image that are already present at the destination. By not transmitting such blocks over the low-bandwidth network helps to reduce migration time. VMFlock [1] optimizes the non-live migration of a group of VMs over WAN through accelerated VM booting on the destination in addition to deduplicating blocks across the images of the group as well those already present at the destination. While the above approaches focus on optimizing the transfer of single or multiple VM images in the context of non-live migration, we focus on optimizing the live transfer of memories of a group of VMs.

In the context of live migrating a single VM, [22] optimizes the live migration of a single VM over wide-area network through a smart stop-and-copy approach which reduces the number of memory copying iterations. [24] and [22] further use of page level deduplication in conjunction with the transfer of differences between modified and original pages (eliminating the need to transfer the entire dirtied page). MECOM [13] uses an adaptive page compression technique to optimize live migration of a single VM.

In the context of live migrating multiple VMs, live gang migration [8] improves the migration of multiple VMs co-located on a single host by combining sub-page level deduplication, page similarity, and delta for dirtied pages. Shrinker [18] proposes a technique for migrating virtual clusters over the high-bandwidth high-delay links of wide-area network (WAN). It uses an online deduplication mechanism, meaning that it performs hash computation for identifying duplicate pages (a CPU-intensive operation) during the migration. Since the WAN link is a high-delay link, the large round-trip latency masks the hash computation overhead during migration. Hence when used in low-latency high-bandwidth datacenter networks, online deduplication yields worse total migration times than the default online compression technique of QEMU/KVM. Therefore IRLM employs offline, rather than online, page-level deduplication to optimize inter-rack migration within a datacenter Ethernet.

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented a system for inter-rack live migration (IRLM) of a group of VMs within a datacenter network having high-bandwidth low-latency Ethernet. IRLM exploits rack-wide deduplication of memory pages to reduce the amount of data transferred over the core network links during migration. Our preliminary evaluations show that, for a configuration of 6 hosts per rack and 4 VMs per host, IRLM can reduce the amount of data transferred over the core links during migration by up to 44% compared to online compression (the default live migration technique in QEMU/KVM) and by up to 17% compared to gang migration. In addition, total migration time of the VM cohort reduces by up to 26% compared to online compression, while it increases by 7% over gang migration. For larger configurations having more machines per rack, we expect the savings in core link bandwidth and migration time to be much greater. We are currently optimizing IRLM to further reduce the amount of data transferred over core links and performing a thorough evaluation on a larger testbed. We are also investigating further reductions in the total migration time of IRLM using the post-copy approach in which target-to-target page transfers can be delayed until after the resumption of VMs at the target rack. Deduplicated pages can either be demand-paged from other targets, or can be prefetched before they are faulted upon. We are also investigating the use of Ethernet multicast to eliminate target-to-target transfers and more aggressive sub-page-level deduplication.

Acknowledgement

This work is supported in part by the National Science Foundation through grants CNS-0845832 and CNS-0855204.

6. REFERENCES

[1] Samer Al-Kiswany, Dinesh Subhraveti, Prasenjit Sarkar, and Matei Ripean. Vmflock: Virtual machine co-migration for the cloud. In *Proc. of High Performance Distributed Computing*, June 2011.

[2] A. Arcangeli, I. Eidus, and C. Wright. Increasing memory density by using ksm. In *Proc. of Linux Symposium*, July 2009.

[3] Netperf: Network Performance Benchmark. http://www.netperf.org/netperf.

[4] Edouard Bugnion, Scott Devine, and Mendel Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. In *ACM Transactions on Computer Systems*, October 1997.

[5] F. Chabaud and A. Joux. Differential collisions in sha-0. In *Proc. of Annual International Cryptology Conference*, August 1998.

[6] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. of Network System Design and Implementation*, May 2005.

[7] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2.

[8] U. Deshpande, X. Wang, and K. Gopalan. Live gang migration of virtual machines. In *Proc. of High Performance Distributed Computing*, June 2010.

[9] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C Snoeren, G. Varghese, G. M Voelker, and A. Vahdat. Difference engine: Harnessing memory redundancy in virtual machines. In *Proc. of Operating Systems Design and Implementation*, December 2010.

[10] OpenSSL SHA1 hash. http://www.openssl.org/docs/crypto/sha.html.

[11] M. Hines, U. Deshpande, and K. Gopalan. Post-copy live migration of virtual machines. *SIGOPS Operating Syst. Review*, 43(3):14–26, July 2009.

[12] W. Huang, M. Koop, Q. Gao, , and D.K. Panda. Virtual machine aware communication libraries for high performance computing. In *Proc. of SuperComputing*, November 2007.

[13] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan. Live virtual machine migration with adaptive, memory compression. In *Proc. of Cluster Computing and Workshops*, August 2009.

[14] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. Kvm: The linux virtual machine monitor. In *Proc. of Linux Symposium*, June 2007.

[15] G. Milos, D.G. Murray, S. Hand, and M.A. Fetterman. Satori: Enlightened page sharing. In *Proc. of USENIX Annual Technical Conference*, June 2009.

[16] M. Nelson, B. H Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *Proc. of USENIX Annual Technical Conference*, April 2005.

[17] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proc. of Cluster, Cloud, and Grid Computing*, May 2009.

[18] P. Riteau, C. Morin, and T. Priol. Shrinker: Improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing. In *Proc. of EUROPAR*, September 2011.

[19] C. P Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *Proc. of Operating Systems Design and Implementation*, December 2002.

[20] J. Sonnek, J. Greensky, R. Reutiman, and A. Chandra. Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. In *Proc. of International Conference on Parallel Processing*, September 2010.

[21] C. A. Waldspurger. Memory resource management in VMware ESX server. In *Proc. of Operating Systems Design and Implementation*, December 2002.

[22] T. Wood, K. Ramakrishnan, J. van der Merwe, and P. Shenoy. CloudNet: a platform for optimized WAN migration of virtual machines. *University of Massachusetts Technical Report TR-2010*, 2, 2010.

[23] Timothy Wood, Gabriel Tarasuk-Levin, Prashant Shenoy, Peter Desnoyers, Emmanuel Cecchet, and Mark D. Corner. Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, March 2009.

[24] X. Zhang, Z. Huo, J. Ma, and D. Meng. Exploiting data deduplication to accelerate live virtual machine migration. In *Proc. of International Conference on Cluster Computing*, September 2010.