Gang Migration of Virtual Machines using Cluster-wide Deduplication

Umesh Deshpande, Brandon Schlinker, Eitan Adler, and Kartik Gopalan
Computer Science, Binghamton University
{udeshpa1,kartik}@binghamton.edu

***Abstract*—Gang migration refers to the simultaneous live migration of multiple Virtual Machines (VMs) from one set of physical machines to another in response to events such as load spikes and imminent failures. Gang migration generates a large volume of network traffic and can overload the core network links and switches in a datacenter. In this paper, we present an approach to reduce the network overhead of gang migration using global deduplication (GMGD). GMGD identifies and eliminates the retransmission of duplicate memory pages among VMs running on multiple physical machines in the cluster. We describe the design, implementation and evaluation of a GMGD prototype using QEMU/KVM VMs. Evaluations on a 30-node Gigabit Ethernet cluster having 10GigE core links shows that GMGD can reduce the network traffic on core links by up to 65% and the total migration time of VMs by up to 42% when compared to the default migration technique in QEMU/KVM. Furthermore, GMGD has a smaller adverse performance impact on network-bound applications.**

I. Introduction

Live migration of a virtual machine (VM) refers to the transfer of a running VM over the network from one physical machine to another. Within a local area network (LAN), live VM migration mainly involves the transfer of the VM's CPU and memory state, assuming that the VM uses network attached storage, which does not require migration. Some of the key metrics to measure the performance of VM migration are as follows.

- *Total migration time* is the time from the start of migration at the source to its completion at the target.
- *Downtime* is the duration for which a VM's execution is suspended during migration.
- *Network traffic overhead* is the additional network traffic due to VM migration.
- *Application degradation* is the adverse performance impact of VM migration on applications running anywhere in the cluster.

We address gang migration [8], i.e. the simultaneous live migration of multiple VMs that run on multiple physical machines in a cluster. The cluster is assumed to have a high-bandwidth low-delay interconnect such has Gigabit Ethernet [10], 10GigE [9], or Infiniband [15]. Datacenter administrators may need to perform gang migration to handle resource re-allocation for peak workloads, imminent failures, cluster maintenance, or powering down of several physical machines to save energy.

This paper specifically focuses on reducing the network traffic overhead due to gang migration. Since gang migration can transfer hundreds of Gigabytes of data over the network, it can overload the core links and switches of the datacenter network. Gang migration can also adversely affect the performance at the network edges where the migration traffic competes with the bandwidth requirements of applications within the VMs. Reducing the network traffic overhead can also indirectly reduce the total time for migrating multiple VMs and the application degradation, depending upon how the traffic reduction is achieved.

Our approach to reduce the network traffic overhead uses the following observation. VMs within a cluster often have similar memory content, given that they may execute the same operating system, libraries, and applications. Hence, a significant number of their memory pages may be identical [26], [30]. One can reduce the network overhead of gang migration using *deduplication*, i.e. by avoiding the transmission of duplicate copies of identical pages. We present an approach called *gang migration using global (cluster-wide) deduplication* (GMGD). During normal execution, a duplicate tracking mechanism keeps track of identical pages across different VMs in the cluster. During gang migration, a distributed coordination mechanism suppresses the retransmission of identical pages over the core links. Specifically, only one copy of each identical page is transferred to a target rack (i.e. the rack where a recipient physical machine for a VM resides). Thereupon, the machines within each target rack coordinate the exchange of necessary pages. In contrast to GMGD, gang migration using *local deduplication* (GMLD) [8] suppresses the retransmission of identical pages from among VMs within a single host. Our main contributions are as follows:

- We present a technique to identify and track identical memory pages across VMs running on different physical machines in a cluster, including non-migrating VMs running on the target machines.
- We present a technique to deduplicate these identical pages during gang migration, while keeping the coordination overhead low.
- We describe a prototype implementation of GMGD on the QEMU/KVM [18] platform.
- We evaluate GMGD on a 30-node cluster testbed having three switches, 10GigE core links and 1Gbps edge links. We compare GMGD against two techniques – the QEMU/KVM's default live migration technique, which we call *online compression* (OC), and GMLD.

Prior efforts to reduce the data transmitted during VM migration have focused on live migration of a single VM [5], [20], [13], [16], live migration of multiple VMs running on the same physical machine (GMLD) [8], live migration of a virtual cluster across a wide-area network (WAN) [22], or non-live migration of multiple VM images across a WAN [17]. Compared to GMLD, GMGD faces the additional challenge of ensuring that the cost of global deduplication does not exceed the benefit of network traffic reduction during live migration. In contrast to migration over a WAN, which has high-bandwidth *high-delay* links, we focus on migration within a datacenter LAN, which has high-bandwidth *low-delay* links. This difference is important because hash computations, which are used to identify and deduplicate identical memory pages, are CPU-intensive operations. When migrating over a LAN, hash computations become a serious bottleneck if performed online during migration, whereas over a WAN, the large round-trip latency can mask the online hash computation overhead.

The rest of the paper is organized as follows. Section II and III describes the design and implementation of GMGD respectively. Section IV compares the performance of GMGD against OC and GMLD. Section V presents related work. Section VI concludes with a summary of contributions.

II. Architecture of GMGD

In this section, we describe the high-level architecture of GMGD. For simplicity of exposition, we first describe how GMGD operates when VMs are live migrated from one rack of machines to another rack, followed by a description of its operation in the general case. For each VM being migrated, the target physical machine is provided as an input to GMGD. Target mapping of VMs could be provided by another VM placement algorithm that maximizes some optimization criteria such as reducing inter-VM communication overhead [27] or maximizing the memory sharing potential [29]. GMGD does not address the VM placement problem nor does it assume the lack or presence of any inter-VM dependencies.

As shown in Figure 1, a typical cluster consists of multiple racks of physical machines. Machines within a rack are connected to a top-of-the-rack (TOR) switch. TOR switches are connected to one or more core switches using high-bandwidth links (typically 10Gbps or higher). GMGD does not preclude the use of other layouts where the core network could become overloaded.

Migrating VMs from one rack to another increases the network traffic overhead on the core links. To reduce this overhead, GMGD employs a cluster-wide deduplication mechanism to identify and track identical pages across VMs running on different machines. As illustrated in Figure 1, GMGD identifies the identical pages from VMs that are being migrated to the same target rack and transfers only one copy of each identical page to the target rack. At the target rack, the first machine to receive the identical page transfers the page to other machines in the rack that also require the page. This prevents duplicate transfers of an identical page over the core network to the same target rack.

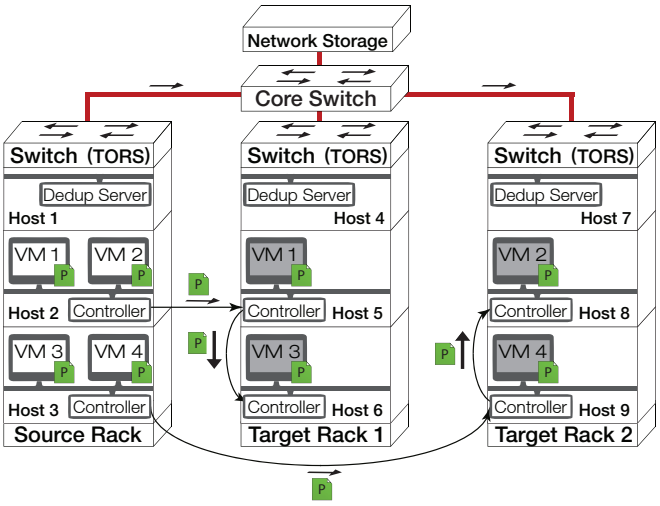

Fig. 1: Illustration of GMGD. Page P is identical among all four VMs at the source rack. VM1 and VM3 are being migrated to target rack 1. VM2 and VM4 are being migrated to target rack 2. One copy of P is sent to host 5 which forwards P to host 6 in target rack 1. Another copy of P is sent to host 9 which forwards P to host 8 in target rack 2. Thus identical pages headed for same target rack are sent only once per target rack over core network.

In our prototype, we implemented GMGD within the default pre-copy mechanism in QEMU/KVM. The pre-copy [5] VM migration technique transfers the memory of a running VM over the network by performing iterative passes over its memory. Each successive round transfers the pages that were dirtied by the VM in the previous iteration. Such iterations are carried out until very small number of dirty pages are left to be transferred. Given the throughput of the network, if the time required to transfer the remaining pages is smaller than a pre-determined threshold, the VM is paused and its CPU state and the remaining dirty pages are transferred. Upon completion of this final phase the VM is resumed at the target. For GMGD each VM is migrated independently with the pre-copy migration technique. Although our GMGD prototype is based on pre-copy VM migration, nothing in its architecture prevents GMGD from working with other live VM migration technique such as post-copy [13].

We next describe two phases of GMGD, namely *duplicate tracking* and *live migration*.

A. Duplicate Tracking Phase

This phase is carried out during the normal execution of VMs at the source machines before the migration begins. Its purpose is to identify all duplicate memory content (presently at the page-level) across all VMs residing on different machines. We use content hashing to detect identical pages. The pages having the same content yield the same hash value. When the hashing is performed using a standard 160-bit SHA1 hash [12], the probability of collision is less than the probability of an error in memory or a TCP connection [4].

In each machine, a *per-node controller* process coordinates the tracking of identical pages among all VMs in the machine. The per-node controller instructs a user-level QEMU/KVM process associated with each VM to scan the VM's memory

image, perform content based hashing and record identical pages. Since each VM is constantly executing, some of the identical pages may be modified (dirtied) by the VM, either during the hashing, or after its completion. To identify these dirtied pages, the controller uses the 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. The first write attempt to any read-only page results in a trap into the hypervisor which marks the faulted page as dirty in its dirty bitmap and allows the write access to proceed. The QEMU/KVM process uses a hypercall to extract the dirty bitmap from KVM to identify the modified pages.

The per-rack *deduplication servers* maintain a hash table, which is populated by carrying out a rack-wide content hashing of the 160-bit hash values pre-computed by per-node controllers. Each hash is also associated with a list of hosts in the rack containing the corresponding pages. Before migration, all deduplication servers exchange the hash values and host list with other deduplication servers.

B. Migration Phase

In this phase, all VMs are migrated in parallel to their destination machines. The pre-computed hashing information is used to perform the deduplication of the transferred pages at both the host and the rack levels. QEMU/KVM queries the deduplication server for its rack to acquire the status of each page. If the page has not been transferred already by another VM, then its status is changed to *sent* and it is transferred to the target QEMU/KVM. For subsequent instances of the same page from any other VM migrating to the same rack, QEMU/KVM transfers the page identifier. Deduplication servers also periodically exchange the information about the pages marked as *sent*, which allows the VMs in one rack to avoid retransmission of the pages that are already sent by the VMs from another rack.

C. Target-side VM deduplication

The racks used as targets for VM migration are often not empty. They may host VMs containing pages that are identical to the ones being migrated into the rack. Instead of transferring such pages from the source racks via the core links, they are forwarded within the target rack from the hosts running the VMs to the hosts receiving the migrating VMs. The deduplication server at the target rack monitors the pages within hosted VMs and synchronizes this information with other deduplication servers. Per-node controllers perform this forwarding of identical pages among hosts in the target rack.

D. Reliability

When a source host fails during migration, the reliability of GMGD is no worse than that of single-VM pre-copy in that only the VMs running on the failed source hosts will be lost, whereas other can continue migrating successfully. However, when a target host fails during migration, more VMs may suffer collateral damage using GMGD than using single-VM pre-copy. This is because the failed target host in GMGD may hold pages that are required by VMs migrating to other machines in the target rack. GMGD can be modified to solve this problem by resending copies of the lost pages from respective source hosts when a target host fails.

III. Implementation Details

We implemented a prototype of GMGD in the QEMU/KVM virtualization environment. Our implementation is completely transparent to the users of the VMs. With QEMU/KVM, each VM is spawned as a process on a host machine. A part of the virtual address space of the QEMU/KVM process is exported to the VM as its physical memory.

A. Per-node Controllers

Per-node controllers are responsible for managing the deduplication of outgoing and incoming VMs. We call the controller component managing the outgoing VMs as the *source side* and component managing the incoming VMs as the *target side*. The controller sets up a shared memory region that is accessible only by other QEMU/KVM processes. The shared memory contains a hash table which is used for tracking identical pages. Note that the shared memory poses no security vulnerabilities because it is outside the physical memory region of the VM in the QEMU/KVM process' address space and is not accessible by the VM itself.

The source side of the per-node controller coordinates the local deduplication of memory among co-located VMs. Each QEMU/KVM process scans its VM's memory and calculates a 160-bit SHA1 hash for each page. These hash values are stored in the hash table, where they are compared against each other. A match of two hash values indicates the existence of two identical pages. Scanning is performed by a low priority thread to minimize interference with the VMs' execution.

The target side of the per-node controller receives incoming identical pages from other controllers in the rack. It also forwards the identical pages received on behalf of other machines in the rack to their respective controllers. Upon reception of an identical page, the controller copies the page into the shared memory region, so that it becomes available to incoming VMs. The shared memory region is freed once the migration is complete.

B. Deduplication Server

Deduplication servers are to per-node controllers what per-node controllers are to VMs. Each rack contains a deduplication server that tracks the status of identical pages among VMs that are migrating to the same target rack and the VMs already at the target rack. Deduplication servers maintain a content hash table to store this information. Upon reception of a 160-bit hash value from the controllers, the last 32-bits of the 160-bit hash are used to find a bucket in the hash table. In the bucket, the 160-bit hash entry is compared against the other entries present. If no matching entry is found, a new entry is created.

Each deduplication server can currently process up to 200,000 queries per second over a 1Gbps link. This rate can

potentially handle simultaneous VM migrations from up to 180 physical hosts. For context, common 19-inch racks can hold 44 servers of 1U (1 rack unit) height [25]. We have also built a certain level of scalability into the deduplication server by using multiple threads for query processing, fine-grained reader/writer locks, and batching queries from VMs to reduce the frequency of communication with the deduplication server.

Finally, the deduplication server does not need to be a separate server per rack. It can potentially run as a background process within one of the machines in the rack that also runs VMs provided that a few spare CPU cores are available for processing during migration.

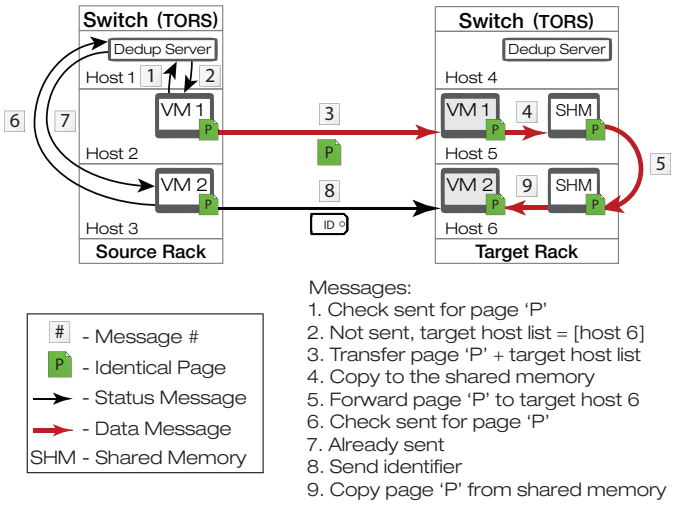

Fig. 2: Deduplication of identical pages during migration.

C. Operations at the Source Machine

Upon initiating simultaneous migration of VMs, the controllers instruct individual QEMU/KVM processes to begin the migration. From this point onward, the QEMU/KVM processes communicate directly with the deduplication servers, without any involvement from the controllers. After commencing the migration, each QEMU/KVM process starts transmitting every page of its respective VM. For each page it checks in the local hash table whether the page has already been transferred. Each migration process periodically queries its deduplication server for the status of next few pages it is about to transfer. The responses from the deduplication server are stored into the hash table, in order to be accessible to the other co-located VMs. If the QEMU/KVM process discovers that a page has not been transferred, then it transmits the entire page to its peer QEMU/KVM process at the target machine along with its unique identifier. QEMU/KVM at the source also retrieves from the deduplication server a list of other machines in the target rack that need an identical page. This list is also sent to the target machine's controller, which then retrieves the page and sends it to the machines in the list. Upon transfer the page is marked as *sent* in the source controller's hash table. The QEMU/KVM process periodically updates its deduplication server with the status of the sent pages. The deduplication server also periodically updates other deduplication servers with a list of identical pages marked as *sent*. Dirty pages and unique pages that have no match with other VMs are transferred in their entirety to the destination. Figure 2 shows the message exchange sequence between the deduplication servers and QEMU/KVM processes for an inter-host deduplication of page P.

D. Operations at the Target Machine

On the target machine each QEMU/KVM process allocates a memory region for its respective VM where incoming pages are copied. Upon reception of an identical page, the target QEMU/KVM process copies it into the VM's memory and inserts it into the target hash table according to its identifier. If only an identifier is received, a page corresponding to the identifier is retrieved from the target hash table, and copied into the VM's memory. Unique and dirty pages are directly copied into the VMs' memory space. They are not copied to the target shared memory.

E. Remote Pages

Remote pages are deduplicated pages that were transferred by hosts other than the source host. Identifiers of such pages are accompanied by a *remote* flag. Such pages become available to the waiting hosts in the target rack only after the carrying host forwards them. Therefore, instead of searching for such remote pages in 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 a *remote thread*, periodically traverses the list, and checks for each entry if the page corresponding to the 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 more recent dirtied 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. The identical pages already present at the target rack before the migration are also treated as the remote pages. The per-node controllers in the target rack forward such pages to the listed target hosts. This avoids their transmission over the core network links from the source racks. However, pages dirtied by VMs running in the target rack are not forwarded to other hosts and they are requested by the corresponding hosts from their respective source hosts.

F. Co-ordinated Downtime Start

A VM cannot be resumed at the target unless all of its pages have been received. Therefore initiating the VM's downtime before completing target-to-target transfers can increase its downtime. However, in the default QEMU/KVM migration technique, downtime is started at the source's discretion and the decision is made solely on the basis of the number of pages remaining to be transferred and the perceived link bandwidth at the source. Therefore, to avoid the overlap between the downtime and target-to-target transfers, we implement a co-ordination mechanism between the source and the

target of each QEMU/KVM process. We prevent the source QEMU/KVM process from starting the VM downtime and keep it in the live pre-copy iteration mode until all of its pages have been retrieved at the target and copied into memory. Thereon, the source is instructed by the target to initiate the downtime. This allows VMs to reduce their downtime, as only the remaining dirty pages at the source are transferred during the downtime. While the source side waits for a permission to initiate the downtime, the VM may dirty more pages. Hence, depending on its dirtying rate, the transfer of additional dirty pages may lead to an increase in the amount of data transferred and hence the total migration time.

G. Desynchronizing page transfers

We also implemented an optimization to improve the efficiency of deduplication. There is a small time lag between the transfer of an identical page by a VM and the status of the page being reflected at the deduplication server. This lag can result in duplicate transfer of some identical pages if two largely identical VMs start migration at the same time and transfer their respective memory pages in the same order of page offsets. To reduce the likelihood of such duplicate transfers, each VM transfers pages in different order depending upon their assigned VM number. With desynchronization, identical memory regions from different VMs are transferred at different times, allowing each QEMU/KVM process enough time to update the deduplication servers about the $sent$ pages before other VMs transfer the same pages.

IV. Evaluation

We evaluated GMGD in a 30-node cluster testbed having high-bandwidth low-delay Gigabit Ethernet. Each physical host has two quad-core 2GHz CPUs, 16GB of memory, and 1Gbps network card. Figure 3 shows the layout of the cluster testbed consisting of three racks, each connected to a different top of rack (TOR) Ethernet switch. The TOR switches are connected to each other by a 10GigE optical link, which acts as the core link. Although we had only the 30-node three-rack topology available for evaluation, GMGD can be used on larger topologies. Live migration of all VMs is initiated simultaneously and memory pages from the source hosts traverse the 10GigE optical link between the switches to reach the target hosts. For most of the experiments, each machine hosts four VMs and each VM has 2 virtual CPUs (VCPUs) and 1GB memory. We compare GMGD against the following VM migration techniques.

(1) Online Compression (OC): This is the default VM migration technique used by QEMU/KVM. Before transmission, it compresses pages that are filled with uniform content (primarily pages filled with zeros) by representing the entire page with just one byte. At the target, such pages are reconstructed by filling an entire page with the same byte. Other pages are transmitted in their entirety to the destination.

(2) Gang Migration with Local Deduplication (GMLD): [8] This technique uses content hashing to deduplicate the pages across VMs co-located on the same host. Only one copy of identical pages is transferred from the source host.

In our initial implementations of GMGD prototype, we considered the use of *online hashing*, in which hash computation and deduplication are performed *during migration* (as opposed to before migration). Hash computation is a CPU-intensive operation. In our evaluations, we found that the online hashing variant performed very poorly, in terms of total migration time, on high-bandwidth *low-delay* Gigabit Ethernet. For example, online hashing takes 7.3 seconds to migrate a 1GB VM and 18.9 seconds to migrate a 4GB VM, whereas offline hashing takes only 3.5 seconds and 4.5 seconds respectively. We found that CPU-heavy online hash computation became a performance bottleneck and, in fact, yielded worse total migration times than even the simple OC technique described above. Given that the total migration time of online hashing variant is considerably worse than offline hashing while achieving only comparable savings in network traffic, we omit the results for online hashing in the experiments below.

A. Network Load Reduction

1) Idle VMs: Here we migrate an equal number of VMs from each of the two source racks, i.e. for 12 x 4 configuration, 4 VMs are migrated from each of the 6 hosts on each source rack. Figure 4 shows the amount of data transferred over the core links for the three VM migration schemes with an increasing number of hosts, each host running four 1GB idle VMs. Since every host runs identical VMs, the addition of each host contributes a fixed number of unique and identical pages. Therefore for all three techniques we observe the linear trend. Among them, since OC only optimizes the transfer of uniform pages, a set that mainly consists of zero pages, it transfers the highest amount of data. GMLD also deduplicates zero pages in addition to the identical pages across the co-located VMs. As a result, it sends less data than OC. GMGD transfers the lowest amounts of data. For 12 hosts, GMGD transfers 65% and 33% less data through the core links than OC and GMLD respectively.

2) Busy VMs: To evaluate the effect of busy VMs on the amount of data transferred during their migration, we run Dbench [6], a filesystem benchmark, inside VMs. Dbench performs file I/O on a network attached storage. It provides an adversarial workload for GMGD because it uses the network interface for communication and DRAM as a buffer. We initiate the execution of Dbench after the deduplication phase of GMGD to ensure that the memory consumed by Dbench is not deduplicated. We migrate the VMs while execution of Dbench is in progress. Figure 5 shows that GMGD yields up to 59% reduction in the amount of data transferred over OC and up to 27% reduction over GMLD.

B. Total Migration Time

1) Idle VMs: To measure the total migration time of different migration techniques, we measure the end-to-end (E2E) total migration time, i.e. the time taken from the start of the migration of the first VM to the end of the migration of the last VM. Cluster administrators may be concerned with E2E

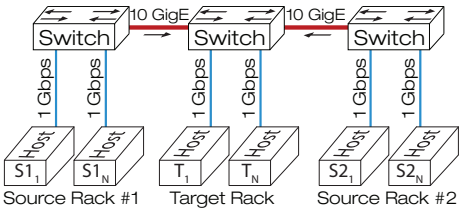

Fig. 3: Layout of the evaluation testbed.

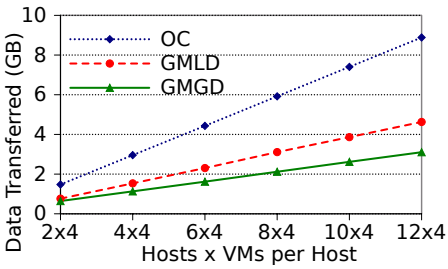

Fig. 4: Network traffic on core links when migrating idle VMs.

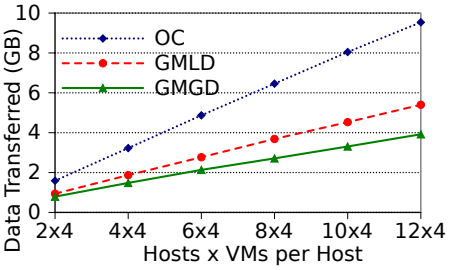

Fig. 5: Network traffic on core links when migrating busy VMs.

Hosts x VMs	Idle VMs			Busy VMs		
	OC	GMLD	GMGD	OC	GMLD	GMGD
2x4	7.28	3.79	3.88	8.6	5.17	4.93
4x4	7.36	3.89	4.08	8.74	5.10	5.06
6x4	7.39	3.92	4.17	8.69	5.15	5.01
8x4	7.11	4.12	4.16	8.77	5.13	4.90
10x4	7.38	4.08	4.27	8.75	5.18	4.91
12x4	7.40	4.05	4.27	8.53	5.06	4.98

TABLE I: Total migration time (in seconds)

total migration time of groups of VMs since it measures the time for which the migration traffic occupies the core links. The idle VM section of Table I shows the total migration time for each migration technique with an increasing number of hosts containing idle VMs. Note that even with the maximum number of hosts (i.e. 12 with 6 from each source rack), the core optical link remains unsaturated. Therefore, for each migration technique we observe nearly constant total migration time, irrespective of the number of hosts. Further, among all three techniques, OC has highest total migration time for any number of hosts, which is proportional to the amount of data it transfers. GMGD's total migration time is slightly higher than that of GMLD, approximately 5% higher for 12 hosts. The difference between the total migration time of GMGD and GMLD can be attributed to the overhead associated with GMGD for performing deduplication across the hosts. While the migration is in progress, it queries with the deduplication server to read, or update the status of deduplicated pages. Such requests need to be sent frequently for effective deduplication.

2) Busy VMs: Table I shows that Dbench increases the total migration time of all the VM migration techniques compared to their idle VM migration times. Since the Dbench traffic competes with the migration traffic for the source NIC, the total migration time of each technique is proportional to the amount of data it transfers. Therefore GMGD's total migration time is slightly lower than that of GMLD.

C. Downtime

Figure 6 shows that increasing the number of hosts does not have a significant impact on the downtimes for all three schemes. This is because each VM's downtime is initiated independently of other VMs. Downtime of all the techniques is in the range of 90ms to 120ms.

D. Background Traffic

In datacenters the switches along the migration path of VMs may experience network traffic other than the VM migration traffic. In overloaded switches, the VM migration traffic may impact the performance of applications running across the datacenter, and vice versa. We first compare the effect of background network traffic on different migration techniques. Conversely, we also compare the effect of different migration techniques on other network-bound applications in the cluster. For this experiment, we saturate the 10GigE core link between the switches with VM migration traffic and background network traffic. We transmit 8Gbps of background Netperf [2] UDP traffic between two source racks such that it competes with the VM migration traffic on the core link.

Figure 7 shows the comparison of total migration time with UDP background traffic for the aforementioned setup. With an increasing number of VMs and hosts, the network contention and packet loss on the two 10GigE core links also increases. We observe larger total migration time for all three techniques compared to the corresponding idle VM migration times listed in Table I. However, observe that GMGD has *lower* total migration time than both OC and GMLD, in contrast to Table I where GMGD had higher TMT compared to GMLD. This is because, in response to packet loss at the core link, all VM migration sessions (which are TCP flows) backoff. However, the backoff is proportional to the amount of data transmitted by each VM migration technique. Since GMGD transfers less data, it suffers less from TCP backoff due to network congestion and completes the migration faster. Figure 8 shows the converse effect, namely, the impact of VM migration on the performance of Netperf. With an increasing number of migrating VMs, Netperf UDP packet losses increase due to network contention. For 12 hosts, GMGD receives 15% more packets than OC and 7% more UDP packets than GMLD.

E. Application Degradation

Table II compares the degradation of applications running inside the VMs during migration using 12x4 configuration.

Sysbench: Here we evaluate the impact of migration on the performance of I/O operations from VMs in the above scenario. We host a Sysbench [24] database on a machine located outside the source racks and connected to the switch with a 1Gbps Ethernet link. Each VM performs transactions on the database over the network. We migrate the VMs while the

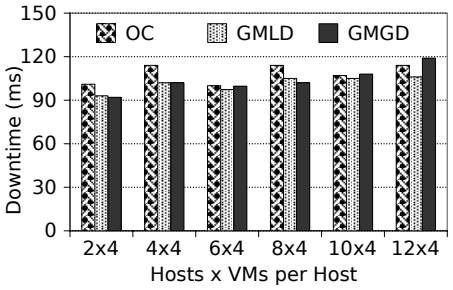

Fig. 6: Downtime comparison.

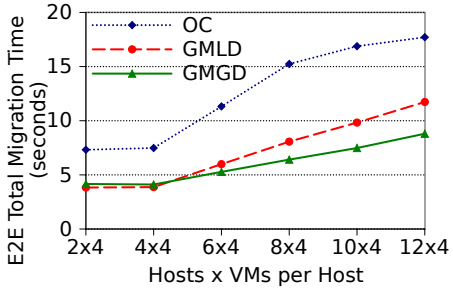

Fig. 7: Total migration time with background traffic.

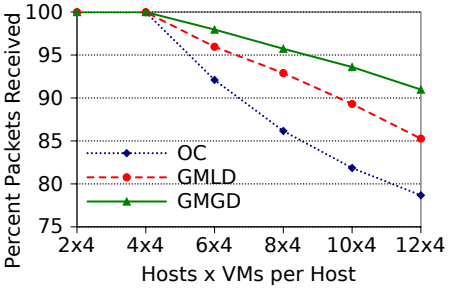

Fig. 8: Background traffic performance.

Benchmarks	W/o Migration	OC	GMLD	GMGD
Sysbench (trans/sec)	31.08	19.32	22.25	26.15
TCP-RR (trans/sec)	1271.6	515.7	742.7	888.33
Sum of Subsets (sec)	6.68	7.07	7.07	7.06

TABLE II: Application degradation in migrating 48 VMs.

benchmark is in progress to observe the effect of migration on the performance of the benchmark. Table II shows the average transaction rate per VM for Sysbench.

TCP_RR: We used the Netperf TCP_RR VM workload to analyze the effect of VM migration on the inter-VM communication. TCP_RR is a synchronous TCP request-response test. We use 24 VMs from 6 hosts as senders, and 24 VMs from the other 6 hosts as receivers. We migrate the VMs while the test is in progress and measure the performance of TCP_RR. Table II shows the average transaction rate per sender VM. Due to the lower amount of data transferred through the source NICs, GMGD keeps the NICs available for the inter-VM TCP_RR traffic. Consequently, it least affects the performance of TCP_RR and gives the highest number of transactions per second among the three.

Sum of Subsets: is a CPU-intensive workload that, given a set of integers and an integer k, finds a non-empty subset that sum to k. We run this program in the VMs during their migration to measure the average per-VM completion time of the program. Due to the CPU-intensive nature of the workload, the difference in the completion time of the application with the three migration techniques is insignificant.

F. Performance Overheads

Duplicate Tracking: Low priority threads perform hash computation and dirty-page logging in the background. With 4 VMs and 8 cores per machine, a CPU-intensive workload (sum of subsets) experienced 0.4% overhead and a write-intensive workload (random writes to memory) experienced 2% overhead. With 8 VMs per machine, the overheads were 6% and 4% respectively due to CPU contention.

Worst-case workload: To evaluate the VM migration techniques against a worst-case workload, we run a write-intensive workload inside VMs that reduces the likelihood of deduplication by modifying two times as much data as the size of each VM. We observe that GMGD does not introduce any additional overheads, compared against OC and GMLD.

Space overhead: At the source side, the shared memory region for local deduplication contains a 160-bit hash value for each VM page. In the worst case when all VM pages are unique, the source side space consumption is around 4% of the aggregate memory of VMs. At the target side, the worst-case space overhead in the shared memory could be 100% the aggregate memory of VMs when each page has exactly one identical counterpart on another host. However, target shared memory only contains identical pages. Unique pages are directly copied into VMs' memories, so they do not incur any space overhead. Further, both the source and the target shared memory areas are used only during the migration and are freed after the migration completes.

V. Related Work

Two lines of research are related to our work – content deduplication among VMs and optimization of VM migration.

Deduplication has been used to reduce the memory footprint of VMs in [3], [26], [19], [1], [29] and [11]. These techniques use deduplication to reduce memory consumption either within a single VM or between multiple co-located VMs. In contrast, we use cluster-wide deduplication across multiple physical machines to reduce the network traffic overhead when simultaneously migrating multiple VMs.

Non-live migration of a single VM can be speeded up by using content hashing to detect blocks within the VM image that are already present at the destination [23]. VM-Flock [17] speeds up the non-live migration of a group of VM images over a high-bandwidth high-delay wide-area network by deduplicating blocks across the VM images. In contrast, we focus on reducing the network performance impact of the *live and simultaneous* migration of the memories of multiple VMs within a high-bandwidth low-delay datacenter network. Cloudnet [28] optimizes the live migration of a single VM over wide-area network. It reduces the number of pre-copy iterations by starting the downtime based on page dirtying rate and page transfer rate. [31] and [28] further use page-level deduplication along with the transfer of differences between dirtied and original pages, eliminating the need to retransmit the entire dirtied page. [16] uses an adaptive page compression technique to optimize the live migration of a single VM. Post-copy [13] transfers every page to the destination only once, as opposed to the iterative pre-copy [20], [5], which

transfers dirtied pages multiple times. [14] employs low-overhead RDMA over Infiniband to speed up the transfer of a single VM. [21] excludes the memory pages of processes communicating over the network from being transferred during the initial rounds of migration, thus limiting the total migration time. [30] shows that there is an opportunity and feasibility for exploiting large amounts of content sharing when using certain benchmarks in high performance computing.

In the context of live migration of multiple VMs, our prior work on GMLD [8] deduplicates the transmission of identical memory content among VMs co-located within a single host. It also exploits sub-page level deduplication, page similarity, and delta difference for dirtied pages, all of which can be integrated in our GMGD prototype. Shrinker [22] migrates virtual clusters over high-delay links of WAN. It uses an *online hashing* mechanism in which hash computation for identifying duplicate pages (a CPU-intensive operation) is performed during the migration. The large round-trip latency of the WAN link masks the hash computation overhead during migration. We chose offline hashing, rather than online hashing, because we found online hashing to be impractical over *low-delay* links such as those in a Gigabit Ethernet LAN. In addition, issues such as desynchronizing page transfers, downtime synchronization, and target-to-target transfers need special consideration in a low-delay network. Further, when migrating a VM between datacenters over WAN, the internal topology of the datacenters may not be relevant. However, when migrating within a datacenter (as with GMGD), the datacenter switching topology and rack-level placement of nodes play important roles in reducing the traffic on core links.

Our preliminary results on this topic were published in a workshop paper [7] that focused upon the migration of multiple VMs between two-racks. This paper presents the comprehensive design, implementation, and evaluation of GMGD for a general cluster topology and also includes additional optimizations such as better downtime synchronization, improved target-to-target transfer, greater concurrency within the deduplication servers and per-node controllers, and more in-depth evaluations on a larger 30-node testbed.

VI. Conclusions

We presented gang migration with global deduplication (GMGD) – a solution to reduce the network load resulting from the simultaneous live migration of multiple VMs within a datacenter that has high-bandwidth low-delay interconnect. Our solution employs cluster-wide deduplication to identify, track, and avoid the retransmission of identical pages over core network links. Evaluations on a 30-node testbed show that GMGD reduces the amount of data transferred over the core links during migration by up to 65% and the total migration time by up to 42% compared to online compression.

Acknowledgement

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

References

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

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

[3] E. Bugnion, S. Devine, and M. Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. In *ACM Transactions on Computer Systems*, October 1997.

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

[5] 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.

[6] Dbench. http://samba.org/ftp/tridge/dbench.

[7] U. Deshpande, U. Kulkarni, and K. Gopalan. Inter-rack live migration of multiple virtual machines. In *Proc. of Virtualization Technologies in Distributed Computing*, June 2012.

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

[9] 10-Gigabit Ethernet. https://en.wikipedia.org/wiki/10_gigabit_ethernet.

[10] Gigabit Ethernet. https://en.wikipedia.org/wiki/gigabit_ethernet.

[11] 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.

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

[13] M. Hines, U. Deshpande, and K. Gopalan. Post-copy live migration of virtual machines. *In SIGOPS Operating Systems Review*, July 2009.

[14] W. Huang, Q. Gao, J. Liu, and D. K. Panda. High performance virtual machine migration with RDMA over modern interconnects. In *Proc. of IEEE International Conference on Cluster Computing*, September 2007.

[15] Infiniband. https://en.wikipedia.org/wiki/infiniband.

[16] 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.

[17] S. A. Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu. Vmflock: Virtual machine co-migration for the cloud. In *Proc. of High Performance Distributed Computing*, June 2011.

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

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

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

[21] A. Nocentino and P. M. Ruth. Toward dependency-aware live virtual machine migration. In *Proc. of Virtualization Technologies in Distributed Computing*, June 2009.

[22] 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 EURO-PAR*, September 2011.

[23] 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.

[24] Sysbench. http://sysbench.sourceforge.net/index.html.

[25] Rack Unit. http://en.wikipedia.org/wiki/rack_unit.

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

[27] J. Wang, K. L Wright, and K. Gopalan. XenLoop: a transparent high performance inter-vm network loopback. In *Proc. of High performance distributed computing*, June 2008.

[28] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. Van Der Merwe. Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines. In *Proc. of Virtual Execution Environments*, March 2011.

[29] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner. Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. In *Proc. of Virtual Execution Environments*, March 2009.

[30] L. Xia and P.A. Dinda. A case for tracking and exploiting inter-node and intra-node memory content sharing in virtualized large-scale parallel systems. In *Proc. of Virtualization Technologies in Distributed Computing*, June 2012.

[31] 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.