Fast Server Deprovisioning through Scatter-Gather Live Migration of Virtual Machines

Umesh Deshpande*, Yang You*, Danny Chan*, Nilton Bila†, Kartik Gopalan*
* Binghamton University, Binghamton, NY
{udeshpa1, yyou4, dchan20, kartik}@binghamton.edu
† IBM Research, Yorktown Heights, NY
nilton@us.ibm.com

***Abstract*—Traditional metrics for live migration of virtual machines (VM) include total migration time, downtime, network overhead, and application degradation. In this paper, we introduce a new metric, *eviction time*, defined as the time to evict the entire state of a VM from the source host. Eviction time determines how quickly the source host can be taken offline, or the freed resources re-purposed for other VMs. In traditional approaches for live VM migration, such as pre-copy and post-copy, eviction time is equal to the total migration time, because the source and destination hosts are coupled for the duration of the migration. Eviction time increases if the destination host is slow to receive the incoming VM, such as due to insufficient memory or network bandwidth, thus tying up the source host. We present a new approach, called *Scatter-Gather* live migration, which reduces the eviction time when the destination host is resource constrained. The key idea is to decouple the source and the destination hosts. The source scatters the VM's memory state quickly to multiple intermediaries (hosts or middleboxes) in the cluster. Concurrently, the destination gathers the VM's memory from the intermediaries using a variant of post-copy VM migration. We have implemented a prototype of Scatter-Gather in the KVM/QEMU platform. In our evaluations, Scatter-Gather reduces the VM eviction time by up to a factor of 6 while maintaining comparable total migration time against traditional pre-copy and post-copy for a resource constrained destination.**

***Keywords*-Virtual Machine, Live Migration, Deprovisioning**

I. Introduction

Live migration [4], [22], [13], [15] of Virtual Machines (VMs) is used in datacenters for consolidation, system maintenance, power savings, and load balancing. Traditional metrics that measure the performance of live VM migration include downtime, total migration time, network traffic overhead, and performance degradation of applications.

In this paper, we introduce a new metric, namely *eviction time*, which we define as the time taken to completely eliminate the state of a VM being migrated from the source host. Quickly evicting a VM from the source host is important in many situations. For example, administrators may want to opportunistically save power by turning off excess server capacity [2], [3], [26], [7], [25], quickly eliminate hotspots by scaling out physical resources for performance assurance [14], quickly evict a lower priority VM to accommodate other higher priority VMs, perform emergency system maintenance, or handle imminent failures.

In traditional live VM migration techniques [4], [22], [13], [15], the eviction time is equal to the *total migration time*, which is defined as the interval from the start of migration at the source to the time when all VM state has been transferred to the destination and the VM resumes execution. Since the source host directly transfers the VM's state to the destination host, typically over a TCP connection, a VM cannot be migrated faster than the slower of the two endpoints. Thus in traditional approaches, the source and the destination are *coupled* for the entire duration of VM migration.

A resource-constrained destination can throttle live migration even if the source can dedicate all of its resources to evict the VM and the network is not a bottleneck. For example, the eviction times can increase when the memory is fully committed at the destination; the destination may need to first free pages by ballooning [28], swapping, or purging its caches, before it can accommodate the new VM. Destination could also be running other network or CPU intensive VMs due to which it may be unable to dedicate sufficient reception bandwidth or processor cycles to receive an incoming VM. Concurrent VM migrations [11], triggered by host decommissioning or VM consolidation [7], [2] can also increase eviction times for individual VMs.

Long eviction times can defeat the key optimization goals of techniques that rely on full VM migration to quickly deprovision the source. For example, techniques that migrate the entire VM out of the source to save energy [3], [26], [7], [25] will be less effective if the eviction takes too long. Similarly, long eviction times can adversely affect the performance of other higher priority VMs that remain at the source.

In this paper, we present a new approach to rapidly evict a VM from the source when the destination host is resource constrained. The key idea is to temporally *decouple* the source and the destination hosts during migration. In other words, the source must be able to quickly unload the VM's state, preferably at its maximum transmission rate, whereas the destination should be able to retrieve and resume the VM at its own pace as enough local resources become available.

Our first contribution in this paper is the *Scatter-Gather*[1] live VM migration technique which reduces eviction time by using intermediate hosts for staging the VM's memory. The source host *scatters* (or evicts) the VM's memory to one

[1]Not to be confused with "Scatter-Gather I/O" [21], which refers to reading/writing data from/to multiple buffers which are separated in memory.

or more intermediate hosts or network middleboxes. Concurrently, the destination host *gathers* the VM's memory from intermediate hosts. Thus the source can evict the VM at its full speed even if the destination is slow. Since existing live migration approaches couple the source to the destination during migration, their eviction time equals total migration time. Our insight is that by decoupling the source and destination, and taking advantage of resources at intermediate hosts, Scatter-Gather migration is able to reduce eviction time, enabling faster deprovisioning of the source. The intermediaries could be peer hosts or network middleboxes such as network caches or memory devices. These need not be hosts that are dedicated for migration.

Our second contribution is to develop a variant of post-copy migration [15], [13] in Scatter-Gather, wherein the VM's CPU state is first resumed at the destination and then its memory pages are gathered from intermediate hosts through a combination of active pre-paging and demand paging. Employing a post-copy variant, as opposed to pre-copy [4], [22], allows the VM to quickly resume at the destination, even as its memory is fetched from intermediate hosts.

Our third contribution is the use of a Virtualized Memory Device (VMD) layer that aggregates the available memory across all intermediate hosts and exports this memory via a block device interface. The VM Migration Manager – a user-space migration process alongside each VM at both the source and destination – uses VMD to stage the VM's memory content, without having to juggle individual connections with multiple intermediate hosts. This increases the modularity and reduces the complexity of the overall system. To the best of our knowledge, this is the first work to demonstrate live VM migration using cluster-memory virtualization.

We expect that Scatter-Gather live migration will act as another useful tool in a datacenter administrator's toolbox that can be employed in situations where reducing VM eviction time is the primary concern. Our vision is that future live VM migration mechanisms will be adaptive in nature; they will dynamically choose from different strategies such as pre-copy, post-copy, Scatter-Gather, gang migration [11], or partial migration [2] that may be best suited for specific VM workloads and system optimization goals.

II. Background

Pre-copy Live Migration: In pre-copy [4], [22] method of live VM migration, contents of a VM's memory are copied over multiple iterations from source to destination, even as the VM is running at the source. The first iteration copies the entire memory of the VM whereas the subsequent iterations copy only the pages dirtied in the preceding iteration. Once the number of dirty pages are relatively small, or a maximum number of iterations are over, the VM is suspended and the CPU state and remaining dirty pages are transferred, after which the VM is resumed at the destination. This window of VM's inactivity during migration is known as *downtime*. If the VM's workload primarily performs memory reads and the writable working set (WWS) of the VM is small, then the downtime will be small. However, for write-intensive workloads, when the size of the WWS is sufficiently large, a significant number of dirty pages will be retransmitted in successive iterations. If the number of dirtied pages does not reduce sufficiently before the maximum threshold of iterations is reached then a large number of dirtied pages may be transferred during the downtime. This also lengthens total migration time, and by extension, eviction time, since the source and the destination are coupled throughout the migration.

Post-copy Live Migration: In post-copy [13], [15], [20] method of live VM migration, the VM is first suspended at the source and the CPU state is transferred to the destination, where the VM is resumed immediately. Subsequently, as the VM executes at the destination, its memory pages are actively pushed from the source – an operation known as pre-paging – with the expectation that most pages would be received by the destination before they are accessed by the VM. If the VM accesses a page that was not yet received by the destination, then the corresponding page is faulted in from the source over the network – an event called remote page fault. The fewer the number of remote page faults, the better the performance of post-copy. In contrast to pre-copy, post-copy sends each page over the network only once; this means that for write-intensive workloads post-copy yields lower network traffic overhead. The total migration time, and eviction time when using post-copy are comparable or lower than pre-copy. Technically, the downtime of post-copy is minimal since the VM's execution switches almost instantaneously to the destination. However, the performance degradation may be worse than pre-copy right after the execution switch because user-level applications in the VM may not become responsive till their working set is fetched from the source.

III. Demonstrating the Problem

This section motivates the need for the Scatter-Gather approach by experimentally demonstrating that eviction time suffers when the destination host is under resource pressure. All experiments in this section use dual quad core servers with 1.7GHz CPUs, 16GB DRAM, and 1Gbps Ethernet cards. All servers are connected to a Nortel 4526-GTX layer-2 Ethernet switch. Hosts run Linux kernel 2.6.32 and KVM/QEMU 1.6.50. All VMs run Linux kernel 3.2 as the guest OS. We use the standard implementation of pre-copy live migration that comes bundled with KVM/QEMU and the publicly available post-copy implementation from the Yabusame [15] project.

To avoid second-order effects in measurements, we presently disable an optimization to compress pages during transmission, since all migration techniques, including Scatter-Gather, can benefit from this optimization. The only reason that we disable compression is to normalize the comparison of different approaches in this paper; it is not meant to increase the baseline migration times.

We demonstrate here that memory pressure at a destination adversely affects VM eviction time using traditional pre-copy and post-copy approaches. We migrate an idle VM with 5GB memory size from the source to the destination. The source

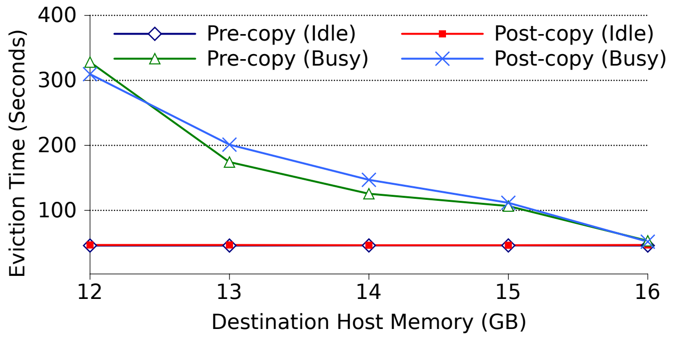

Fig. 1. Eviction time of a single idle VM. The destination host is either idle or runs two busy VMs running TunkRank (indicated in parentheses). The memory pressure at the destination is varied by setting the available DRAM in BIOS from 12GB (high memory pressure) to 16GB (low memory pressure).

host only performs migration of an idle VM and nothing else, whereas the destination host faces varying degrees of resource pressure. The destination host is either idle (denoted "Idle" in our results) or runs two VMs of 5GB memory size, each of which runs the TunkRank graph analytics benchmark from the CloudSuite [12] package. TunkRank is a memory and CPU-intensive benchmark which determines a Twitter user's influence based on the followers. TunkRank uses a 1.3GB Twitter database as input, which generates a runtime memory pressure of around 4GB per VM.

We increase the available DRAM at the destination host from 12GB to 16GB in 1GB increments (using BIOS options at boot time), thus gradually decreasing the memory pressure. Figure 1 plots the eviction time measured when using both pre-copy and post-copy. When the destination host is idle, both pre-copy and post-copy yield low eviction times. However, when the destination host is busy running the TunkRank workload in two VMs, the eviction time for both the migration techniques increases by a factor of 6.3 (from 52s to 328s) as the available DRAM is reduced from 16GB to 12GB. When the destination doesn't have enough free memory to store the incoming VM's pages, the host OS responds by swapping out the pages of the busy VMs running TunkRank. The time spent in reclaiming the memory pages to create free space for the incoming VM at the destination increases the eviction time at the source. Note that if, instead of being idle, the migrating VM was running a write-intensive workload, then the eviction time of pre-copy would be much worse than that of post-copy because the eviction time of pre-copy is impacted by the working set size of the VM.

IV. Architecture of Scatter-Gather Migration

In traditional live VM migration, as shown in Figure 2, the source would directly transfer the VM's state to the destination through a TCP connection, which carries both data (VM's memory and CPU state) and control information (handshakes, synchronization, etc.) This direct TCP connection would last until the destination receives the entire VM.

In the Scatter-Gather approach, as shown in Figure 3, the source and destination exchange bulk of VM's memory through intermediate hosts I_1 ... I_N. Only the VM's CPU

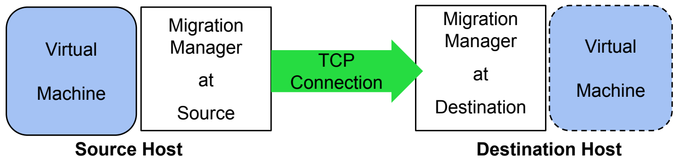

Fig. 2. Coupling in traditional pre-copy and post-copy: The VM's pages are transferred through a direct TCP connection between the Migration Managers at the source and destination hosts.

execution state, any demand-paged memory, and control information, are exchanged through a direct TCP connection between the source and destination. This connection lasts only until the source evicts the entire VM. The source and destination run *Migration Managers* for each VM being migrated. In KVM/QEMU virtualization platform, the Migration Manager is part of QEMU – a multi-threaded user-level process, one for each VM, that mediates between the VM and the hypervisor besides carrying out VM migration. A *Virtualized Memory Device* (VMD) layer, aggregates the free memory of all intermediate hosts and presents the collection as block device to the Migration Managers at the source and the destination.

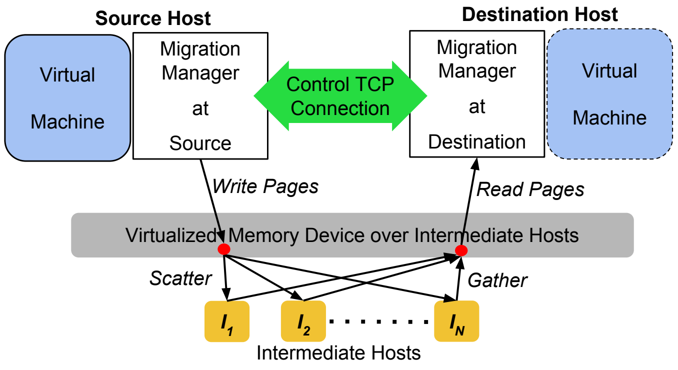

Fig. 3. Scatter-Gather migration: VM's pages are transferred through intermediate hosts using VMD. A direct TCP connection between the source and destination carries control and demand-paging information.

A. Scatter Phase

The goal of the scatter phase is to quickly evict the VM's memory and execution state from the source host. This phase is executed at the source host. First, a control TCP connection is established between the source and the destination. Next, the VM's CPU state is transferred to the destination where the VM is resumed immediately. Since the VM's memory still resides at the source host, the VM would start generating page-faults as it accesses its memory. The destination's Migration Manager sends all page-fault requests during the scatter phase to the source's Migration Manager over the control TCP connection, which then responds with the faulted page. This step is similar to the demand-paging component of traditional post-copy migration. However, simply relying on demand-paging would be terribly slow.

To speed up the eviction of VM's memory, the Migration Manager at the source also actively transfers the VM's pages out of the source host to intermediate hosts. The Migration Manager opens the block device exported by the VMD as a file and sequentially writes the VM's memory pages to this file. The VMD simply represents an aggregation of the memory of intermediate hosts. Thus the VMD layer at the source "scatters", or distributes, the pages written to the block device over the network to the intermediate hosts. For each page written to the VMD, corresponding control information is sent directly to the destination's Migration Manager over the control TCP connection. The control information consists of the address of each page that was scattered and its status, which may indicate if any content optimization, such as compression or deduplication, was applied to the page. This information is stored by the Migration Manager at the destination and used later to gather the VM's pages from the VMD. Once the VM's entire memory has been evicted to the VMD, the VM can be deprovisioned at the source.

B. Gather Phase

The gather phase retrieves the VM's memory pages from the intermediate hosts and the source. This phase runs concurrently with the scatter phase at the source. As soon as the destination receives the VM's execution state from the source, it starts executing the VM. Gather phase consists of two components: (a) pre-paging, or actively collecting, the VM's pages from the intermediate hosts and (b) demand-paging the faulted pages from the source. In pre-paging, the destination's Migration Manager opens a block device exported by the VMD to which the source host scatters the VM's memory and listens on the control TCP connection on which the source sends information about the scattered pages. The destination's Migration Manager uses the per-page control information received from the source to copy the received pages into the VM's memory. Thus the control TCP connection ensures that the destination reads each page from the VMD only after it has been written to the VMD by the source.

The demand-paging component proceeds as follows. The gather phase overlaps with the scatter phase till the time that the source completely evicts and deprovisions the VM. Hence, if the VM faults on any page during this overlap time, the destination's Migration Manager directly requests the source for the faulted pages. These demand-paging requests are sent over the control TCP connection to the source which then sends the requested page again over the TCP connection. To reduce the latency of servicing page faults, the source's Migration Manager gives a higher priority to transfer the faulted pages compared to the pages being scattered over VMD. If the VM at the destination faults for pages after the VM has been deprovisioned at the source, the faulted pages are read from the VMD. Thus the pre-paging and the demand-paging components of the gather phase proceed concurrently in independent threads with minimal mutual interference.

C. Selection of Intermediaries

The choice of intermediaries impacts the performance of Scatter-Gather migration. The selection should ensure that all intermediaries *collectively* have sufficient excess capacity (memory, CPU, and network bandwidth) to allow the source to evict the VM's memory at line speed. Intermediaries' selection also depends upon the context in which Scatter-Gather is used. For instance, when used to rapidly deprovision an entire rack of machines, the intermediaries should preferably be located at the destination rack so that the machines in the source rack do not participate in the gather phase and can be deprovisioned quickly. Also note that the destination host itself could be one of the "intermediaries" that participates in the VMD layer and receives a subset of the pages scattered by the source. Doing so would ensure that, in the worst case, if other intermediaries become slow for any reason, then Scatter-Gather migration is not slower than direct migration to the destination. Finally, the selection scheme may depend upon the consideration of fault-tolerance during migration wherein the VMD layer could replicate each page at multiple intermediaries for recovery. We leave to future work an in-depth investigation and evaluation of various intermediary selection algorithms.

D. Alternative Designs

The approach presented in this paper uses a variant of post-copy approach to resume the VM at the destination. Other design alternatives are also worth discussing. First alternative is to use a variant of pre-copy, instead of post-copy, to migrate the VM. Specifically, the source host could use iterative pre-copy rounds to save the VM's memory at the intermediate hosts, while the VM executes at the source. Concurrently, the destination can gather the pages from the intermediate hosts at its own pace. Thus the source and destination can be decoupled during the pre-copy phase. Once the set of dirty pages is small enough, the VM can be suspended, its execution state transferred to the destination, remaining pages retrieved from the source, and the VM resumed. The downtime is dependent on the number of pages that need to be retrieved by the destination from intermediate hosts in the last step; if the source completes much faster than the destination is able to pull pages, then downtime would be large.

To address this problem, a second alternative is to use a hybrid of pre-copy and post-copy approaches. Specifically, the last step of the above pre-copy-based approach can be altered so that the the VM resumes immediately at the destination after the CPU state is transferred. Any remaining pages at the intermediate hosts are gathered by the destination using post-copy (i.e. pre-paging plus demand-paging). While the downtime of this approach would be lower than the first alternative, eviction time would still be substantially long for write-intensive VM workloads because the pre-copy rounds will take a very long time to converge.

V. Implementation

We implement Scatter-Gather migration on KVM/QEMU platform. As described in Section IV, the Migration Managers at both the source and destination open the block device exported by the VMD layer to execute the scatter and gather phases. Below we describe implementation-specific details related to the VMD, the Migration Managers, and a rate limiting option at the destination during the gather phase.

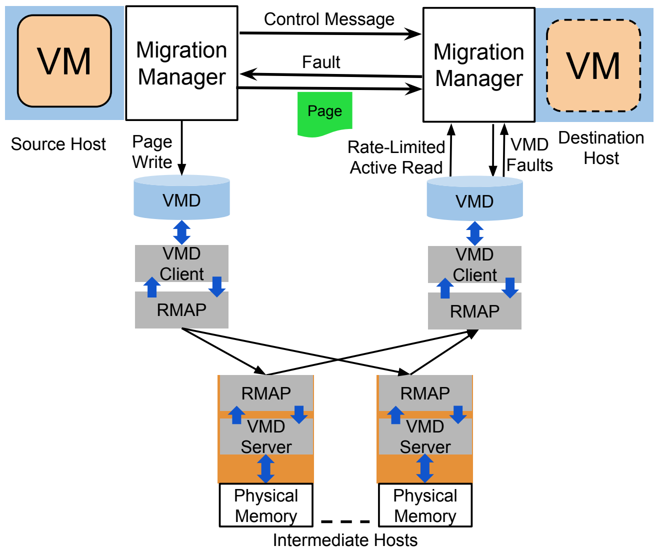

Fig. 4. Message exchange and data transfer between Migration Managers and VMD during Scatter-Gather migration.

A. *Virtualized Memory Device (VMD) Layer*

Our implementation of VMD is a successor to our prior work on the MemX system [10]. VMD is a distributed peer-to-peer memory sharing system among machines in an Ethernet network. VMD simplifies the implementation of Migration Managers by exporting the aggregate free memory of the intermediate hosts through a block device interface. Thus the Migration Managers can transfer pages to/from all intermediate hosts via a single interface without knowing the identity of the intermediaries or managing the location of each page.

VMD is implemented via two sets of Linux kernel modules – *client* VMD modules that run at the source and destination hosts, and *server* VMD modules at the intermediate hosts. The server module uses the free memory of the intermediate hosts to store and track pages. Figure 4 shows the interactions between the VMD clients and servers. VMD uses a custom-designed layer-2 protocol called RMAP, that is described in full in [10]. RMAP is a reliable message oriented protocol. RMAP includes features such as flow control and fragmentation-reassembly. Upon receiving a read request, the VMD server retrieves and forwards the corresponding page to the requesting client. The client module communicates with several intermediate servers and forwards the read or write request to the appropriate intermediate server. The placement of a VM's pages at the intermediaries is determined by the source's VMD client using a load-aware algorithm; the VMD server modules periodically report their excess memory capacity and reception bandwidth to the client; the source's VMD client sends pages in round-robin order to the set of VMD servers which last reported positive excess memory and reception bandwidth.

B. *Migration Manager*

We modify a publicly available post-copy implementation from the Yabusame project [15] to implement the Migration Managers. To recall, the Migration Manager at the source uses a control TCP connection with the destination to communicate control information about each page which includes the pseudo-physical address of the page in VM's memory and the page's location (block offset) in the VMD. During the scatter phase, demand-paging requests from the destination arrive at the source over the control TCP connection. The source prioritizes the transfer of the faulted pages by temporarily interrupting scatter operation so that the faulted pages do not face network queuing delays behind the pages being scattered.

The destination-side Migration Manager consists of a UMEM device (/dev/umem), its driver in the kernel-space, and a UMEMD process in the user-space. The UMEMD process coordinates with the source-side Migration Manager. UMEM device provides memory sharing between the QEMU process and the UMEMD process. Therefore, UMEMD can directly access the VM's memory to copy the received VM pages. UMEMD also opens a VMD device in read-only mode to read VM pages. For each page written to the VMD, the source side forwards a control message to the UMEMD, which it stores in an offset list. This information is later used to read faulted pages from the VMD.

When a running VM accesses a page that has not been received from the source it generates a fault. The UMEM driver in the kernel intercepts the fault and notifies the UMEMD user-space process. A dedicated user-space thread, created by the UMEMD process, handles the page faults. Upon receiving a fault from the UMEMD driver, the thread checks the state of the Scatter-Gather migration. If the scatter phase is in progress, the fault is redirected towards the source over the control connection. Otherwise the VMD offset of the faulted page is read from the offset list created earlier, the page is read from the VMD, and copied into the memory region shared with the QEMU process. Once the faulted page is in place, the UMEMD process notifies the VM through the UMEM device. UMEMD also creates a thread to actively gather the pages from the VMD. This thread traverses the offset list received from the source, sequentially reads the pages from the VMD and copies them into the VM's memory, unless they have been already serviced via a page-fault.

C. *Rate Limiting of Gather Phase*

Scatter-Gather provides the option to limit the rate at which the gather phase reads pages from the VMD. In the Section VI-B we demonstrate that rate-limiting the gather phase can reduce the performance impact of migration on co-located network-bound VMs at the destination while delivering low VM eviction time at the source. To implement rate-limiting, we allow users to input the rate at which a VM's pages can be read from the VMD. The destination Migration Manager divides the actively read pages into batches. It keeps track of the rate of the active reads for each batch of pages and instructs the thread reading the pages from the VMD to sleep for the duration needed to maintain the desired rate of reception.

VI. Evaluation

We now experimentally demonstrate that, compared to standard pre-copy and post-copy, Scatter-Gather live migration can reduce the VM eviction time at the source even when the destination is resource constrained. The experimental setup is the same as in Section III except that, for Scatter-Gather migration, we now use one intermediate node to stage the VM's memory. All nodes are connected to the same switch.

The reason we use only one intermediate node in our experiments is to show that Scatter-Gather can reduce eviction time even with just one intermediate node. Our current prototype can also run with multiple intermediate hosts. Our results, which we omit here due to space constraints, indicate that the eviction time and total migration time are independent of the number of intermediate hosts. This holds as long as the intermediate hosts are not bandwidth constrained and the sum of their free memory is greater than the migrating VM's size.

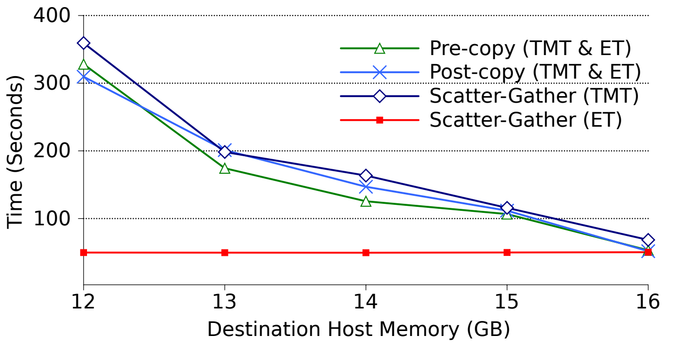

Fig. 5. Comparison of Eviction Time (ET) and Total Migration Time (TMT) for a single 5GB idle VM to a busy destination host. The destination already hosts two 5GB VMs running TunkRank.

A. Eviction Time with Memory Bottleneck

Recall that in Section III, we showed that memory pressure at a destination adversely affected the VM eviction time using traditional pre-copy and post-copy approaches. Here we show that Scatter-Gather migration can deliver consistently low eviction time even when the memory pressure at the destination increases. As in Figure 1, the memory pressure at the destination is controlled by changing the destination host's memory size from 12GB to 16GB in 1GB steps, indicating progressively less memory pressure. We migrate an idle 5GB VM to a destination hosting two 5GB VMs running TunkRank and measure the eviction time. Figure 5 shows that the eviction time for Scatter-Gather is around 6 times shorter than for pre-copy and post-copy. Furthermore, it remains fairly constant (at around 49 seconds) irrespective of the memory pressure at the destination. In contrast, eviction time for pre-copy and post-copy steadily increases with memory pressure.

At the same time, Figure 5 shows that the total migration time of Scatter-Gather is only slightly higher (by up to 10%) than pre-copy and post-copy. This modest overhead is due to two reasons. First, the memory pages are transferred over two hops to the destination, as opposed to just one for pre-copy and post-copy. Secondly, our implementation of layer-2 RMAP protocol in VMD presently delivers around 750 to 800Mbps throughput on a 1Gbps Ethernet when the intermediate nodes simultaneously handle reads and writes, whereas direct TCP connection between source and destination can achieve close to 900Mbps throughput. The second factor is merely an implementation artifact and we plan to resolve it soon. Note that, even with a lower transmission throughput, Scatter-Gather can still deliver a low VM eviction time; higher throughput will only help reduce it further.

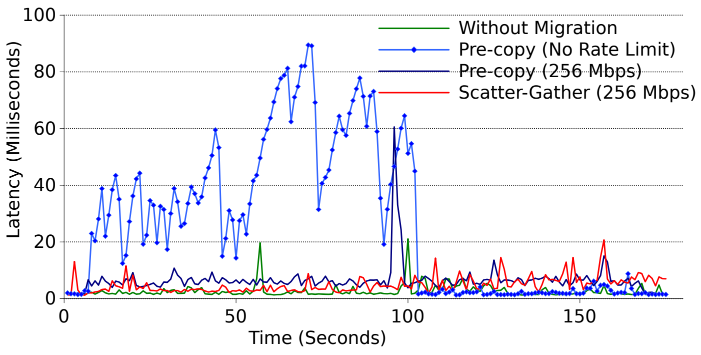

Fig. 6. Memcached request latency. The destination runs memcached servers in two 5GB VMs while an idle 5GB VM is migrated from the source.

B. Bandwidth Pressure at the Destination

We now consider the impact of bandwidth pressure at the destination. Our focus here is not just VM eviction time by itself, but *the tradeoff between the VM eviction time and the performance of network-bound VMs running at the destination*. It is well known [5] that during live VM migration, performance of other network-bound applications at the source and destination can suffer because of bandwidth contention with VM migration traffic. Here we consider the performance of co-located applications only at the destination; we assume that the source can dedicate its entire transmission bandwidth to evict the VM quickly.

To avoid performance impact on other network-bound applications, a commonly prescribed solution is to rate-limit (i.e. limit the bandwidth used by) the VM migration. While this does improve the network bandwidth available to co-located applications, it also has the unfortunate side-effect of prolonging the VM's eviction. We show that, when using Scatter-Gather, this trade-off between eviction time at the source and the application performance at the destination is unnecessary, i.e. we can lower VM eviction time at the source and simultaneously rate-limit the gather phase to maintain application performance at the destination.

We run two VMs at the destination, each running a memcached server. Each VM caches a 3GB Twitter dataset in its memory and responds to query and update requests from an external client. We simultaneously migrate a 5GB idle VM from the source to the destination. The quality of service (QoS) guarantee for the memcached benchmark in CloudSuite [12] is specified as 95% of the requests being executed within 10ms. During migration, the incoming migration traffic competes with the memcached request-response traffic for the link bandwidth. Figure 6 shows that, without any rate limiting for the migration, all of the memcached requests sent during the

	Eviction Time (Seconds)		
	Pre-copy	Post-copy	Scatter-Gather
Rate Limit (256 Mbps)	160.8	164.3	49.8
No Rate Limit	98.6	106.3	49.5

TABLE I
EVICTION TIME COMPARISON WHEN THE MIGRATING 5GB IDLE VM WITH AND WITHOUT RATE LIMITING.

	Network Overhead (GB)		
	Pre-copy	Post-copy	Scatter-Gather
Idle VM	5.01	5.00	10.03
Busy VM	15.48	5.00	10.24

TABLE II
AMOUNT OF DATA TRANSFERRED FOR THE MIGRATION OF A SINGLE 5GB IDLE OR A BUSY VM. BUSY VM RUNS TUNKRANK.

migration take more than 10ms to complete. When we rate-limit the migration at 256Mbps, memcached performance improves with the QoS specifications for all migration schemes. However, Table I shows that, for pre-copy and post-copy, rate limiting the VM migration increases the VM eviction time by almost 1.5 times or more. In contrast, rate-limiting the gather phase of Scatter-Gather does not significantly increase the eviction time.

This experiment considered a bandwidth constrained destination. If on the other hand, the source is bandwidth constrained then the eviction time is lower-bounded by the bandwidth that the source can dedicate to evict the VM. Scatter-Gather is primarily useful when the destination is more resource constrained and slower than the source.

C. Network Overhead

In our current implementation, lower eviction time of Scatter-Gather comes with a tradeoff, namely a higher network overhead. In this section we quantify this overhead and its implications. Table II shows the amount of VM migration traffic when an idle or a busy 5GB VM is migrated. All busy VMs run TunkRank while the migration is in progress. Note that since post-copy transfers each page only once over the network it has lowest network overhead with an idle or a busy VM among the three VM migration techniques. With pre-copy, the network overhead for a busy VM increases three fold as compared to its network overhead with an idle VM. Since TunkRank is a write-intensive application it continuously dirties VM pages during its migration. Since pre-copy retransmits dirtied pages, its network overhead increases. In Scatter-Gather, each page traverses the network twice: once during the scatter phase and again during either the gather or demand-paging phase. For migrating an idle VM, Scatter-Gather has the highest network overhead as a result. However, for migrating a busy VM, pre-copy has the highest network overhead since dirtied pages are retransmitted multiple times.

For this paper, we did not use any content-based optimizations in Scatter-Gather, such as compression or deduplication. Prior work [9], [11], [23], [18] has shown that deduplication can significantly reduce the network overhead of VM migration. We expect Scatter-Gather VM migration to benefit the most from redundancy elimination by network optimizers because the same (unmodified) pages are transferred twice and can be easily cached by middleboxes in the network.

VII. Related Work

To our best knowledge, Scatter-Gather live migration is the first approach that aims to reduce VM eviction time when the destination is resource constrained. Numerous live VM migration approaches exist in literature; here we review the ones related to lowering the total migration time, checkpoint/restart, and applications of post-copy.

Lowering Total Migration Time: Traditional post-copy [13], [15] VM migration provides a lower total migration time and network traffic overhead for write-intensive applications compared to traditional pre-copy [4], [22]. Optimizations such as ballooning [28], [13], dropping the guest cache [17], deduplication [11], [9], [8], [18], [23], [29], compression [16], [11], and dynamic VM synthesis [24], can lower the amount of migration traffic and consequently the total migration time. These optimizations are orthogonal to our contributions. We are presently implementing optimizations such as cluster-wide deduplication [9] in VMD.

Relationship to Checkpoint/Restore: All virtualization platforms [27], [1], [19] include a checkpoint/restore functionality in which one can take a snapshot of a VM's memory and execution state and restore the VM later from the snapshot. If eviction time were the only metric of interest, then checkpoint/restore via the memory of intermediate nodes would yield the lowest eviction time. Traditionally, restoration is performed only after the checkpoint operation is complete, resulting in a large downtime. Scatter-Gather live migration approach can be viewed as a combination of live post-copy migration and checkpoint/restore via intermediate nodes; the checkpointing (scatter) phase proceeds concurrently with the restoration (gather) phase, yielding lower downtime while matching the eviction time of traditional checkpoint/restore. Remus [6] provides high-availability for Xen VMs by capturing high-frequency VM snapshots at a destination backup site using a variant of Xen's pre-copy VM migration. While the VM can be quickly restored in the case the source fails, high-frequency snapshots can impose significant overhead during runtime, especially for write-intensive workloads.

Post-copy and its applications: Post-copy VM migration was first proposed in [13] for the Xen platform and subsequently also implemented in [15] for the KVM/QEMU platform. The primary advantage of post-copy is that it quickly offloads the VM's CPU execution state to another machine and the memory follows later. This property of post-copy has been exploited in a number of scenarios. SnowFlock [20] uses post-copy to swiftly clone VMs and execute them simultaneously on multiple hosts to run high performance parallel applications; these cloned VMs disappear when the computation ends. Jettison [2] proposes partial VM migration in which only the working set of an idle VM is migrated; this can be used to save power by consolidating idle VMs from multiple desktops at a central server so that the desktops can enter sleep mode. Post-copy has also been used for performance assurance [14] by quickly eliminating hotspots. We propose a variant of post-copy in Scatter-Gather where pre-paging at the destination actively fetches pages from intermediate hosts and demand-paging fetches faulted pages from the source.

VIII. Future Work

We plan to extend this work as follows. First, we will investigate eviction time when migrating multiple VMs such as when deprovisioning an entire rack of machines. Secondly, we plan to investigate ways to reduce network overhead in Scatter-Gather by incorporating VMD-level optimizations such as distributed deduplication and compression, which may be particularly effective when simultaneously migrating multiple VMs. Thirdly, we plan to investigate alternative designs of Scatter-Gather VM migration discussed in Section IV-D. Finally, we will investigate the inclusion of the destination host as part of the VMD layer to ensure that Scatter-Gather is never slower than direct migration to the destination.

IX. Conclusions

Eviction time has not been considered as a metric in previous live VM migration approaches. When the VM's destination host is resource constrained and slow in receiving the VM's state, the source host is unable to quickly repurpose the resources previously allocated for the VM. We presented a new approach called *Scatter-Gather* live migration with the specific objective of reducing VM eviction time. Our approach decouples the source and the destination hosts during migration; the source scatters the VM's memory pages to multiple intermediate hosts from where the destination gathers these pages. We described a prototype implementation of Scatter-Gather live migration in KVM/QEMU platform. Our implementation uses a variant of post-copy in the gather phase of migration and uses a distributed memory virtualization layer to simplify the core Migration Manager. In our evaluations, Scatter-Gather migration reduces the VM eviction time by up to a factor of 6 while maintaining comparable total migration time against traditional pre-copy and post-copy for a resource-constrained destination. We expect that Scatter-Gather migration will act as another useful tool in a datacenter administrator's toolbox that can be used when low VM eviction time is important.

Acknowledgement

This work is supported in part by the National Science Foundation through grants CNS-1320689, CNS-0845832, CNS-0855204, CCF-1005153, and CNS-1040666.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[2] N. Bila, E. de Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan. Jettison: Efficient idle desktop consolidation with partial VM migration. In *Eurosys*, April 2012.

[3] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing SLA violations. In *Integrated Network Management*, May 2007.

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

[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 *NSDI*, 2005.

[6] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Network System Design and Implementation*, April 2008.

[7] T. Das, P. Padala, V. Padmanabhan, R. Ramjee, and K. G. Shin. LiteGreen: Saving energy in networked desktops using virtualization. In *USENIX Annual Technical Conference*, 2010.

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

[9] U. Deshpande, B. Schlinker, E. Adler, and K. Gopalan. Gang migration of virtual machines using cluster-wide deduplication. In *International Symposium on Cluster, Cloud and Grid Computing*, May 2013.

[10] U. Deshpande, B. Wang, S. Haque, M. Hines, and K. Gopalan. MemX: Virtualization of cluster-wide memory. In *International Conference on Parallel Processing*, September 2010.

[11] U. Deshpande, X. Wang, and K. Gopalan. Live gang migration of virtual machines. In *High Performance Distributed Computing*, June 2011.

[12] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A.D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *ASPLOS*, 2012.

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

[14] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi. Reactive consolidation of virtual machines enabled by postcopy live migration. In *Virtualization Technologies in Distributed Computing*, June 2011.

[15] T. Hirofuchi and I. Yamahata. Yabusame: Postcopy Live Migration for Qemu/KVM. In *KVM Forum 2011, Vancouver, Canada*, August 2011.

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

[17] C. Jo, E. Gustafsson, J. Son, and B. Egger. Efficient live migration of virtual machines using shared storage. In *Virtual Execution Environments*, March 2013.

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

[19] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the linux virtual machine monitor. In *Linux Symposium*, June 2007.

[20] H.A. Lagar-Cavilla, J.A. Whitney, A.M. Scannell, P. Patchin, S.M. Rumble, E de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *EuroSys*, 2009.

[21] S. Loosemore, R.M. Stallman, R. McGrath, A. Oram, and U. Drepper. *The GNU C library reference manual*. Free software foundation, 2001.

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

[23] 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 *EURO-PAR*, September 2011.

[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, October 2009.

[25] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering energy proportionality with non energy-proportional systems-optimizing the ensemble. *HotPower*, 2008.

[26] A. Verma, P. Ahuja, and A. Neogi. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware'08*.

[27] VMWare Inc. Architecture of VMWare ESXi, http://www.vmware.com/files/pdf/esxi_architecture.pdf.

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

[29] 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 *Virtual Execution Environments*, March 2011.