Scatter-Gather Live Migration of Virtual Machines

Umesh Deshpande, Danny Chan, Steven Chan, Kartik Gopalan, and Nilton Bila

Abstract—We introduce a new metric for live migration of virtual machines (VM) called *eviction time* defined as the time to evict the state of one or more VMs from the source host. Eviction time determines how quickly the source can be taken offline or its resources repurposed for other VMs. In traditional live migration, such as pre-copy and post-copy, eviction time equals the total migration time because the source is tied up until the destination receives the entire VM. We present *Scatter-Gather* live migration which decouples the source and destination during migration to reduce eviction time when the destination is slow. The source scatters the memory of VMs to multiple nodes, including the destination and one or more intermediaries. Concurrently, the destination gathers the VMs' memory from the intermediaries and the source. Thus eviction from the source is no longer bottlenecked by the reception speed of the destination. We support simultaneous live eviction of multiple VMs and exploit deduplication to reduce network overhead. Our Scatter-Gather implementation in the KVM/QEMU platform reduces the eviction time by up to a factor of 6 against traditional pre-copy and post-copy while maintaining comparable total migration time when the destination is slower than the source.

Index Terms—Virtual machine, live migration, eviction time, deprovisioning

1 Introduction

Live migration [1], [2], [3], [4] 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 evict the state of one or more VMs being migrated from the source host. Fast eviction of VMs from the source is important in many situations. For example, administrators may want to opportunistically save power by turning off excess server capacity [5], [6], [7], [8], [9], [10], quickly eliminate hotspots by scaling out physical resources for performance assurance [11], quickly evict lower priority VMs to accommodate other higher priority ones, perform emergency maintenance [12], or handle imminent failures.

In traditional live VM migration techniques [1], [2], [3], 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 the entire VM state has been transferred to the destination and the VM resumes execution. When 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. The source is *coupled* to the destination for the entire duration of VM migration and cannot be quickly deprovisioned.

There are various reasons why a destination may receive a VM slower than a source can evict it. The destination host may be under a transient resource pressure which the VM placement algorithm could not predict. The network path from the source to the destination may be congested. The destination may be a consolidation server that is concurrently receiving VMs from multiple sources, resulting in reduced reception speed for individual VMs. Finally, the key priority at a given moment may be to evict an idle or a less important VM from the source to free up resources for more important VMs, irrespective of the reception speed of the destination.

When the source and destination are coupled during migration, slower reception at the destination increases the VM's eviction time. Long eviction times can defeat key system optimizations 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 [6], [7], [8], [9] 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.

We present a new approach to rapidly evict one or more VMs from the source when the destination is slow in receiving the VMs. The key idea is to temporally *decouple* the source and the destination hosts during migration. The source should quickly unload the state of migrating VMs, preferably at its maximum transmission rate, whereas the destination should retrieve and resume the VMs at its own pace, as local resources become available. Our key contributions are as follows:

- We introduce ***Scatter-Gather***[1] **live VM migration** which reduces eviction time for migrating one or more VMs by using intermediaries to stage the bulk

[1] Not to be confused with "Scatter-Gather I/O" [13], which refers to reading/writing data from/to multiple buffers in memory.

- U. Deshpande is with the IBM Research Labs - Almaden, San Jose, CA. E-mail: udeshpa@us.ibm.com.
- D. Chan and K. Gopalan are with the Binghamton University, Binghamton, NY. E-mail: {dchan20, kartik}@binghamton.edu.
- S. Chan is with the University of California, Irvine, CA. E-mail: chans3@uci.edu.
- N. Bila is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY. E-mail: nilton@us.ibm.com.

*Manuscript received 29 Dec. 2014; revised 1 Aug. 2015; accepted 4 Sept. 2014. Date of publication 23 Sept. 2015; date of current version 7 Mar. 2018.
Recommended for acceptance by B. Schulze.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2015.2481424*

of VMs' memory. The source host *scatters* (or evicts) the memory to multiple nodes, including the destination and one or more intermediaries. The intermediaries could be other hosts in the cluster or network middleboxes (e.g., network caches or memory devices) that together have sufficient memory and bandwidth to receive the VM at full eviction speed from the source. Concurrently, the destination *gathers* the memory from both the source and the intermediaries. Thus, by temporally decoupling the source and destination, the source can evict VMs at its full speed even if the destination is slower in receiving VMs. Worst case eviction time with Scatter-Gather is no worse than direct migration using pre-copy and post-copy because Scatter-Gather first saturates the direct connection between the source and destination before routing excess traffic through intermediaries.

- We develop **a variant of post-copy migration [3]** in Scatter-Gather, wherein the VM's CPU state is first resumed at the destination while its memory is still scattered across the intermediaries and the source. The VM's memory pages are gathered from intermediaries through a combination of active pre-paging and demand paging. Using a post-copy variant, as opposed to pre-copy [1], [2], allows the VM to resume at the destination even as its memory is fetched from intermediaries.
- We introduce a novel use of **cluster-wide memory virtualization** [14] for live VM migration. A Virtualized Memory Device (VMD) layer aggregates free memory across all intermediaries and exports it in the form of a block device. Due to over-provisioning of resources in datacenters, machines often have spare memory available [15] and such machines can be used as intermediaries in the VMD. The VMD is used by the *VM Migration Manager*—a user-space migration process alongside each VM—at both the source and destination 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 our system.
- We exploit **deduplication of VMs' memory** to reduce the network overhead when Scatter-Gather migration is used to simultaneously migrate multiple VMs. The source and intermediaries identify pages that have identical memory content on different migrating VMs; the source transfers only one copy of such duplicate pages to the intermediaries. Deduplication is implemented completely in the VMD layer and is fully transparent to the migrating VMs.

1.1 When to Use Scatter-Gather Migration

No migration technique is appropriate for every situation. 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 the VM eviction time is the primary concern. While the Scatter-Gather eviction time is never worse than pre-copy and post-copy, it may increase the total migration time (as we show in Section 6). Our vision is that algorithms will be used to automate the selection of different migration techniques such as pre-copy [1], post-copy [3], Scatter-Gather, gang migration [16], or partial migration [5], [17], that are best suited for specific optimization goals, configurations, and workloads.

2 Background

We begin with a brief background of pre-copy and post-copy and their relation to VM eviction time.

2.1 Pre-Copy Live Migration

In the pre-copy [1], [2] method, a VM's memory contents are copied from source to destination over multiple iterations 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 is relatively small or the maximum number of iterations has completed, the VM is suspended and its CPU state and remaining dirty pages are transferred to the destination where it is resumed. This period of VM's inactivity during migration is known as *downtime*. If the VM's workload primarily performs memory reads and the writable working set of the VM is small, then the downtime will be small. However, for write-intensive workloads, when the size of the writable working set 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 a maximum threshold of iterations is reached, then a large number of dirtied pages may be transferred during the downtime. Thus, write-intensive workloads lengthen the downtime, total migration time and, by extension, eviction time since the source and destination are coupled throughout the migration.

2.2 Post-Copy Live Migration

In the post-copy [3], [4], [18] method, 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 and total migration time, and hence eviction time. 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 until their working set is fetched from the source.

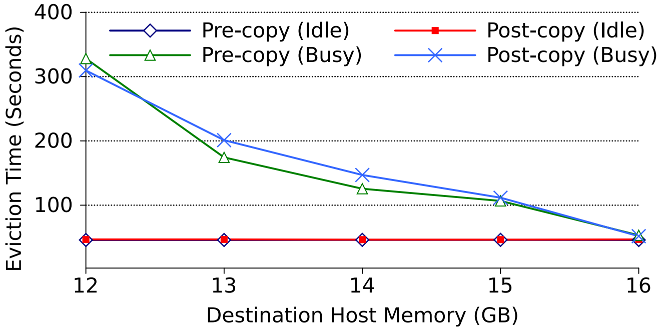


Fig. 1. Eviction time of a single idle VM. The *destination host* is either idle or runs two busy VMs running TunkRank.

3 Demonstrating the Problem

Here we experimentally demonstrate that eviction time suffers with traditional migration techniques when the destination host is under resource pressure. All experiments in this section use dual quad core servers with 2.3 GHz CPUs, 26 GB DRAM, and 1 Gbps Ethernet cards. All servers are connected to a Nortel 4526-GTX Ethernet switch. Hosts run Linux kernel 3.2.0.4-amd64 and KVM/QEMU 1.6.50. All VMs run Ubuntu 14.04.2 as the guest OS, use Linux kernel 3.2, have 2 virtual CPUs (vCPUs), and have Virtio enabled for both the hard disk and network adapter. 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 [4] project.

Fig. 1 demonstrates 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 5 GB memory size from the source to the destination. The source host only performs migration of an idle VM and nothing else, whereas the destination host faces varying degrees of memory pressure. The destination host is either idle (denoted "Idle" in our results) or runs two VMs of 5 GB memory size both running TunkRank. TunkRank is a memory and CPU-intensive graph analytics benchmark from the CloudSuite [19] package which determines a Twitter user's influence based on followers. We have configured TunkRank to use a 1.3 GB Twitter database which generates a memory pressure of around 4 GB per VM during execution.

We increase the available DRAM at the destination host from 12 GB to 16 GB in 1 GB increments (using BIOS options at boot time), thus gradually decreasing the memory pressure. Fig. 1 plots the eviction time for a single VM 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 migration techniques increases by a factor of 6.3—from 52 s for 16 GB DRAM to 328s for 12 GB DRAM. The hypervisor at the destination must swap out resident pages to accommodate the incoming VM, which slows down the reception rate and increases the eviction time at the source.

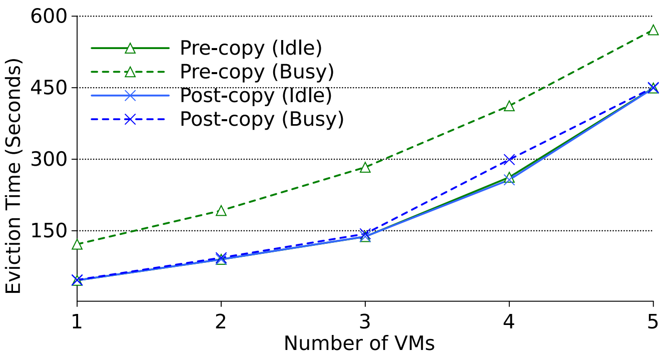


Fig. 2. Eviction time of multiple 5 GB VMs. The *migrating VMs* are either idle or busy running TunkRank.

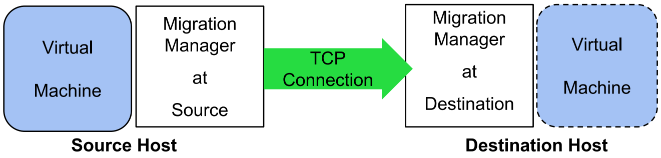


Fig. 3. Coupling in traditional Pre-copy and Post-copy.

Fig. 2 shows the severity of this problem when multiple 5 GB VMs are migrated simultaneously. The destination has 26 GB memory and hosts two 5 GB busy VMs running TunkRank. The migrating VMs are either idle or busy executing TunkRank. The eviction time increases by about 400 s for all cases as the number of VMs being migrated increases from 1 to 5. When migrating busy VMs, pre-copy eviction time is higher because TunkRank dirties a large number of pages which must be retransmitted. In contrast, post-copy sends each page only once, so the eviction time remains similar when migrating both idle and busy VMs.

4 Scatter-Gather Migration Design

In traditional live VM migration, shown in Fig. 3, the source directly transfers the VM's state to the destination. The source and destination run *Migration Managers* for each VM being migrated. A TCP connection between the Migration Managers carries both data (VM's memory and CPU state) and control information (handshakes, synchronization, etc). This connection is torn down only after the destination receives the entire VM.

The Scatter-Gather approach is shown in Fig. 4. The source not only sends the VM's memory directly to the destination, to the extent allowed by the destination's reception bandwidth, but also scatters any remaining memory to the

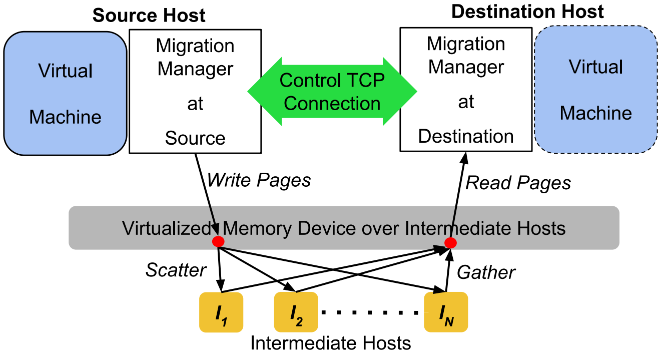


Fig. 4. Scatter-Gather migration: The VM's state is transferred through intermediaries. A direct connection between the source and destination carries control information, faulted pages, and some actively pushed pages.

intermediaries $I_1 ... I_N$. The destination concurrently gathers the scattered content from the intermediaries at its own pace. The direct TCP connection between the source and destination is also used to exchange the VM's CPU execution state, any demand-paged memory, and control information. Unlike traditional migration, the direct connection lasts only until the source evicts the entire VM.

For simplicity and without loss of generality, the discussion below treats the destination and intermediaries as distinct entities. We assume that the destination host is selected either by an administrator or an automated VM placement algorithm. Scatter-Gather can be used with any VM Placement algorithm.

4.1 Virtualized Memory Device

We begin by introducing the *Virtualized Memory Device* (VMD) layer, through which the source transfers the bulk of the VM's memory to the destination. Although Scatter-Gather can be implemented without it, the VMD layer simplifies and modularized the design.

The VMD layer aggregates the available free memory of all intermediaries and presents the collection as a block device, one per VM being migrated. The Migration Manager at the source writes (scatters) to the block device the part of the VM's memory that is not sent directly to the destination. The Migration Manager at the destination concurrently reads (gathers) the VM's memory from the block device. No physical memory is reserved in advance at the intermediaries; instead, the VMD layer at the source uses the memory availability information at the intermediaries to dynamically decide where to scatter the memory pages (details in Section 5).

4.2 Scatter Phase

The goal of the scatter phase is to quickly evict the VM's memory and execution state from the source host. First, a control TCP connection is established between the Migration Managers at the source and the destination. Next, the VM's CPU state is transferred to the destination where the VM is resumed immediately (as in post-copy migration). Since the VM's memory still resides at the source host, the VM generates 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, relying on demand-paging alone would be very slow.

To speed up the eviction of VM's memory, the Migration Manager at the source also actively scatters the VM's pages out of the source to intermediaries and the destination. To send pages to the intermediaries, the Migration Manager writes the VM's memory pages to the block device which was exported by the VMD. Each page written to the VMD is sent to one of the intermediaries depending on its offset in the VM's memory. To improve the fault-tolerance of migration, each page could also be replicated to multiple intermediaries. For each page written to the VMD, the source sends the corresponding control information directly to the destination's Migration Manager over the TCP connection. The control information consists of the address of each page that was scattered and its status, which may indicate whether any content optimization, such as compression or deduplication, was applied to the page. This information is used later by the Migration Manager at the destination to gather the VM's pages from the VMD. Once the VM's entire memory has been evicted, the VM can be deprovisioned at the source and its memory reused for other VMs.

4.3 Gather Phase

The gather phase retrieves the VM's memory pages from the intermediaries 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. The gather phase consists of two components: a) pre-paging, or actively collecting the VM's pages from the intermediaries and the source 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. In addition, it 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 from the VMD 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. Pages actively pushed from the source are copied into the VM's memory just like in post-copy migration.

The demand-paging component proceeds as follows. The gather phase overlaps with the scatter phase until 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 service the faulted pages compared to the pages being actively scattered. After the source has deprovisioned the VM, any pages subsequently faulted upon by the destination are read from the VMD. Pre-paging and demand-paging in the gather phase proceed concurrently in independent threads with minimal mutual interference.

4.4 Deduplication when Evicting Multiple VMs

Multiple VMs may be evicted simultaneously when deprovisioning a server or racks of servers. Simultaneous migration of VMs generates a large volume of network traffic, which increases the individual eviction time of each VM. Prior work [16], [20], [21], [22], [23], [24] has shown that VMs that execute similar operating systems, applications or libraries may have significant number of identical pages. While the use of deduplication during VM migration is not new, our goal here is to demonstrate that Scatter-Gather migration can also exploit any memory content redundancy

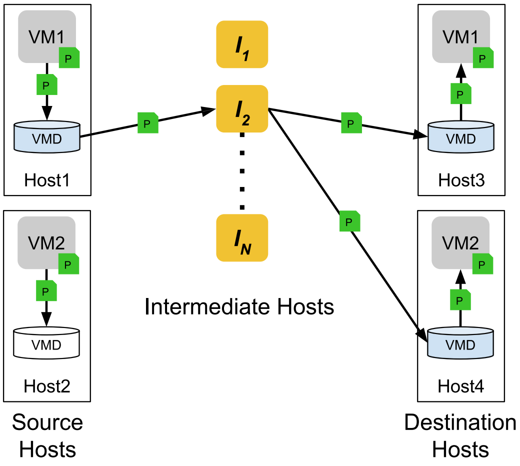


Fig. 5. Deduplication of page ‘P’ during the Scatter phase.

across multiple migrating VMs to reduce the amount of data transferred during the scatter phase. A side effect of using deduplication is that it also reduces the memory requirement at the intermediaries to store the VM pages.

For modularity, we include deduplication as part of the VMD layer, which has a global view of the memory pages of various VMs. As the pages are written from the source, the VMD layer identifies memory pages having duplicate memory content (by comparing their 160-bit SHA1 [25] hash value) and eliminates the retransmission of identical pages across all migrating VMs.

Fig. 5 shows the deduplication of an identical page ’P’ during the scatter phase of migration. $Host_1$ and $host_2$ run VM_1 and VM_2 respectively, containing an identical page ’P’. Both the VMs are migrated simultaneously to $host_3$ and $host_4$. In the scatter phase of migration, the VMD client module in $host_1$ transfers the page ‘P’ from VM_1 to the intermediate node I_2. Therefore the VMD client module in $host_2$ eliminates the duplicate transfer of the same page from the VM_2. In the gather phase, the identical page ‘P’ is forwarded to each destination host separately.

4.5 Selection of Intermediaries

The goal of intermediary selection should be to ensure at the minimum that all intermediaries *collectively* have sufficient excess capacity (memory, CPU, and network bandwidth) to receive the VM being evicted from the source at line speed. Eviction should occur as fast as the source is capable and no single intermediary should be overloaded. Beyond these requirements, the specific algorithms for selecting intermediaries is orthogonal to the core Scatter-Gather technique. Various requirements for intermediary selection can be formulated as a constraint optimization problem and solved using previously developed techniques [26]. We briefly discuss some factors that may affect intermediary selection.

Intermediary selection should be such as to prevent a a snowballing effect where one Scatter-Gather migration might slow down an intermediary enough to trigger other Scatter-Gather migrations. While the VMD currently does not reserve any memory at the intermediaries, one could place upper limits on the memory and bandwidth used at the intermediaries by transiting VMs and give higher priority to any locally incoming VMs.

Another factor affecting selection is that the collective network bandwidth between the intermediaries and the destination should ideally match, if not exceed, the available bandwidth between the source and the destination. Although eviction time reduction is our primary goal, we do not want the total migration time with Scatter-Gather to significantly increase.

Intermediaries’ selection also depends upon the context in which Scatter-Gather migration 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.

4.6 Alternative Designs

The goal of Scatter-Gather migration is to reduce the eviction time. Therefore the approach presented in this paper uses a variant of post-copy, which transfers each page only once as opposed to pre-copy, which performs iterative memory copying. Two other design alternatives are also worth a brief discussion.

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 intermediaries 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 intermediaries, and the VM resumed.

A second alternative is to use a hybrid of pre-copy and post-copy approaches. The above pre-copy-based approach can be altered so that (a) the source limits the number of iterative pre-copy rounds to a fixed value (say one or two), (b) the VM resumes immediately at the destination after the CPU state is transferred, and (c) any remaining pages at the intermediaries are gathered by the destination using post-copy (i.e., pre-paging plus demand-paging).

While the performance degradation due to the above alternatives may be lower than Scatter-Gather, the eviction time would be longer because of pre-copy rounds. Downtime depends on the number of pages retrieved by the destination from the intermediaries in the last step; if the source completes much faster than the destination is able to pull pages, then the downtime would be large.

5 Implementation

We have implemented Scatter-Gather migration on the KVM/QEMU virtualization platform. Below we describe the implementation details of the VMD, the Migration Managers, deduplication, and destination rate limiting.

5.1 Virtualized Memory Device (VMD) Layer

The VMD is a distributed peer-to-peer memory sharing system among machines in an Ethernet network. Our

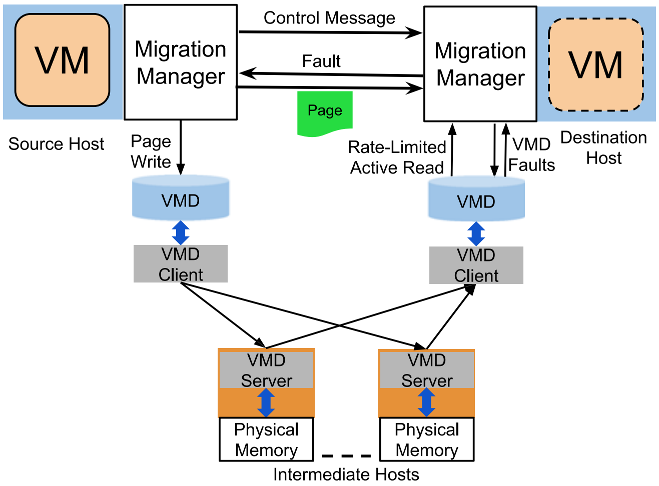


Fig. 6. Interaction between migration managers and VMD.

implementation of the VMD is based on the MemX system [14]. MemX is a single client system where a pool of memory can be accessed only from a single host. VMD extends MemX by making this pool simultaneously accessible from multiple VMD clients. This allows the Migration Managers to simultaneously write and read the VM memory state from the source and the destination hosts. The VMD is implemented as two sets of Linux kernel modules—VMD *client* modules that run at the source and destination, and VMD *server* modules at the intermediaries. The VMD clients are stateless, whereas the VMD servers store the page location and content hashes.

For each VM being migrated, the VMD client modules at the source and destination export a block device to the Migration Managers. The block device is a logical representation of the aggregated memory; no physical memory is reserved in advance at the intermediaries. Instead, the VMD servers periodically broadcast resource announcement messages to advertise their identity and memory availability to VMD clients. The source VMD client uses this information to decide where subsequent writes are sent. The block device allows the Migration Managers to transfer pages via a single interface without knowing the identity of the intermediaries or managing the location of each page.

Every VMD server acts as both an *index server* and a *content server*. The index server stores the location of a page whereas the content server stores the page content. Each index server is responsible for a range of page offsets and maintains a mapping of page offsets to content hash values for each stored page within its offset range. This mapping allows it to locate the content server for a given page offset. The content hash determines the content server on which a page is stored; distinct range of content hash values are handled by each content server. The separation of page index and content information between the index and content server allows for deduplication, which means that identical pages can be stored on the same content server but indexed from different index servers.

Fig. 6 shows the interactions between the Migration Managers, the VMD clients, and VMD servers. The VMD clients communicate with the VMD servers using the TCP protocol. At the start of migration, intermediaries are chosen according to their memory and bandwidth availability. Intermediaries are iteratively added to the list until they collectively provide sufficient bandwidth to saturate the outgoing link at the source and sufficient memory to accommodate the VM's pages. During a migration, the list of intermediaries in the VMD remains static. Once the VM has been completely migrated, the destination's Migration Manager dissolves the VMD and releases its resources.

5.2 Migration Manager

The Migration Manager executes as 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. The Migration Managers at both the source and destination open the block device exported by the VMD layer to carry out the scatter and gather phases. We modify a publicly available post-copy implementation from the Yabusame project [4] to implement the Migration Managers. During the migration, the Migration Manager at the source uses a TCP connection with the destination to communicate control information about each page, which includes the pseudo-physical address of the page in the VM's memory and its block offset in the VMD.

During the scatter phase, demand-paging requests from the destination arrive at the source over the 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 Migration Manager at the destination uses a special device (called `/dev/umem`) to share memory between the QEMU process and a user-level UMEMD process. The latter communicates with the source-side Migration Manager to coordinate the transfer of VM's memory. Thus the UMEMD process directly copies the received pages into the VM's memory. UMEMD also opens the VMD block device in read-only mode to retrieve pages from intermediaries. For each page written to the VMD, the source forwards a control message to the UMEMD providing the page offset, which the UMEMD uses to read the page from the VMD.

When a migrating VM begins executing at the destination, it can generate page faults on pages that have not yet been retrieved by the UMEMD. A page fault implies that either the source Migration Manager hasn't written the page to the VMD or the destination Migration Manager hasn't read the page from the VMD. The UMEM driver in the kernel intercepts the page fault and notifies a dedicated user-space thread in the UMEMD process. The thread then checks the state of the Scatter-Gather migration. If the scatter phase is in progress, then the faulted pages is requested from the source over the TCP connection. If the scatter phase has completed then all pages have already been transferred to the VMD. Hence the faulted page is retrieved from the VMD using the page offset that was earlier received from the source and the page is copied into the VM's memory shared with the QEMU process. Once the faulted page is in its place, the VM is allowed to continue. UMEMD also creates a thread to actively gather the pages from the VMD.

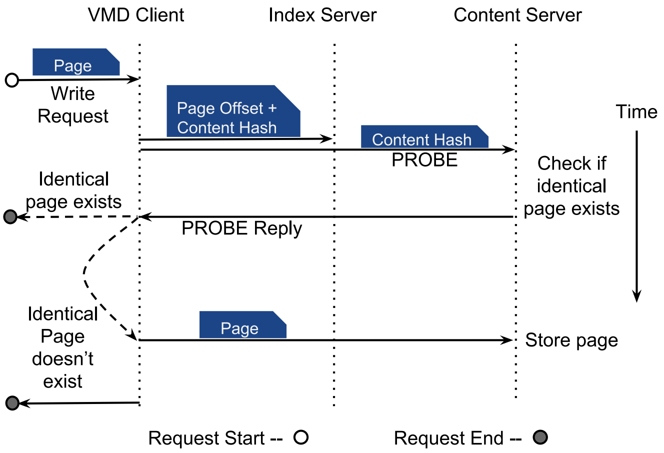


Fig. 7. Message sequence for a page written to the VMD.

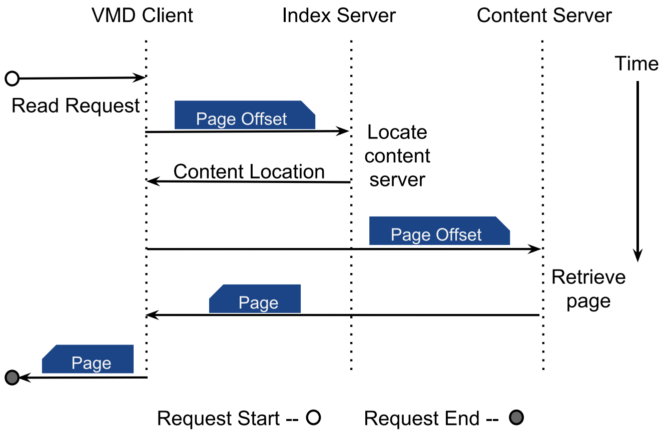


Fig. 8. Message sequence to read a deduplicated page.

This thread traverses the page offsets received from the source, sequentially reads the pages from the VMD and copies them into the VM's memory, unless they have already been serviced via page-faults.

5.3 Deduplication in VMD

The VMD transparently deduplicates identical pages across multiple migrating VMs and eliminates their retransmission during migration. The Migration Managers at the source and the destination do not require any modifications to use the content deduplication in VMD. Identical pages are identified by comparing their 160-bit SHA1 [25] hash, the probability of a hash collision being less than the probability of an error in memory or a TCP connection [27].

5.3.1 Write Operation with Deduplication

Fig. 7 shows the message sequence between a VMD client and VMD servers for the deduplication of a page written to the VMD. The VMD client receives a write request from the Migration Manager at the source host during the scatter phase. The client avoids the transfer of any page whose identical copy already exists on the VMD. To confirm this, upon receiving a write request, the client module calculates a 160-bit SHA1 hash value for the page. Then it forwards a PROBE message to the content server responsible for that hash value. The content server searches the local database of hash values to check if a page with same hash value exists. Depending upon the reply of the content server (PROBE Reply), the client can either forward the page if no identical page is found, or it can eliminate the transfer of additional copy if the identical page already exists at the server. In either case, the index server is notified of the page-offset-to-content-hash mapping for the written page.

5.3.2 Read Operation

Fig. 8 shows the message sequence between a VMD client and VMD servers for reading a page from the VMD. To read a page, the Migration Manager at the destination requests its VMD client to retrieve the VM's pages. The client module forwards the read requests to an index server that is responsible for the offset of the requested page. The index server finds out the content hash for the given page and replies to the VMD client. From the content hash, the client determines the content sever and submits the read request. The content server searches for the page using its content hash and forwards the page to the VMD client.

5.4 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 Section 6.3, 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 bounds the rate at which pages are read from the VMD by appropriately pausing the thread performing the read I/O operations.

6 Evaluation

In this section, we evaluate the performance of Scatter-Gather migration compared to standard pre-copy and post-copy migration. We evaluate eviction times when migrating single and multiple VMs, performance degradation on both co-located and migrating VMs, network overhead reduction using deduplication, and the impact of using multiple intermediaries. For all the experiments, each data-point shows an average performance over six iterations. The experimental setup is the same as in Section 3 except that, for Scatter-Gather migration, we now use one intermediary, or more in Section 6.6, to stage the VM's memory. Each intermediary runs Linux kernel 3.2.0.4-amd64 and has dual quad core 2.3 GHz CPUs, 70 GB DRAM, and a 1 Gbps Ethernet card. All nodes are connected to the same Ethernet switch.

6.1 Eviction Time with Memory Bottleneck

Recall that in Section 3, 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

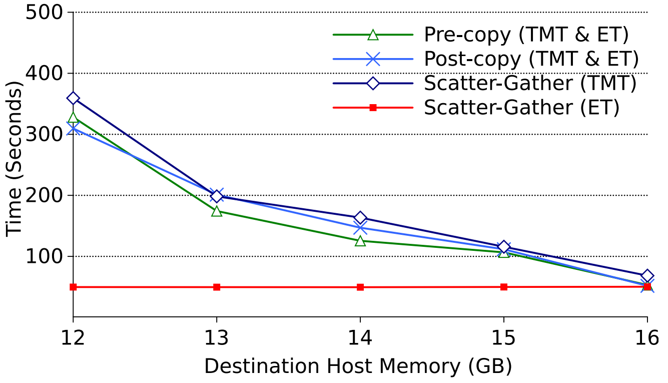


Fig. 9. Eviction Time (ET) and Total Migration Time (TMT) for migrating a 5 GB idle VM to a busy destination.

pressure at the destination increases. We control the memory pressure at the destination by changing the destination's memory size in the BIOS from 12 GB to 16 GB in 1 GB steps, indicating progressively less memory pressure. We migrate an idle 5 GB VM to a destination that already runs two 5 GB VMs running TunkRank and measure the eviction time. Fig. 9 shows that for a host under memory pressure (12 GB), the eviction time for Scatter-Gather is around six times shorter than for pre-copy and post-copy. Furthermore, since the source and destination are decoupled, a memory constrained destination does not throttle the migration from the source, hence the eviction time remains fairly constant (at around 49 seconds). In contrast, eviction times for pre-copy and post-copy increase with memory pressure.

At the same time, Fig. 9 shows that the total migration time of Scatter-Gather is only slightly higher (by up to 10 percent) 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. Second, our implementation of the VMD presently delivers around 750 to 800 Mbps throughput on a 1 Gbps Ethernet when the intermediate nodes simultaneously handle reads and writes, whereas direct TCP connection between source and destination can achieve close to 900 Mbps throughput. 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.

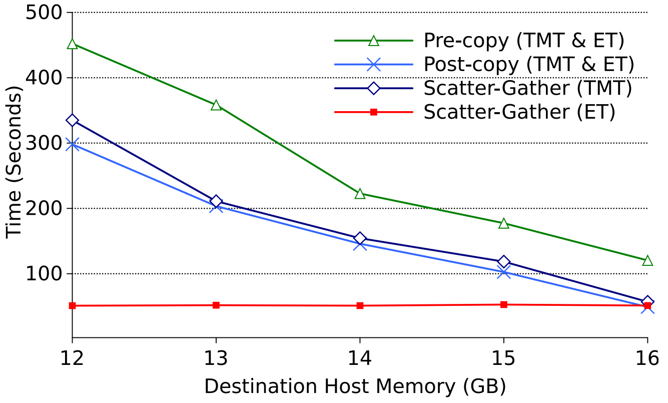


Fig. 10. Eviction Time (ET) and Total Migration Time (TMT) for migrating a 5 GB busy VM to a busy destination.

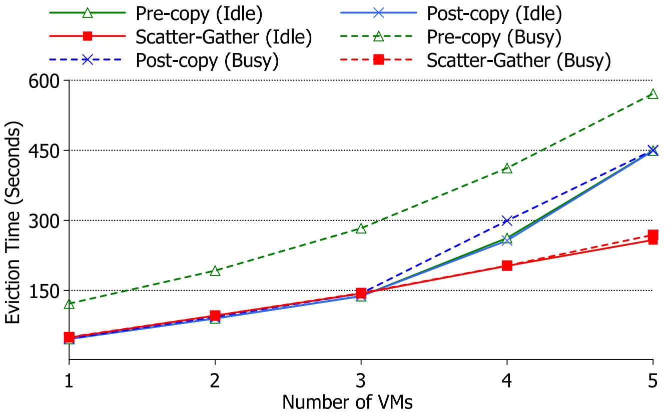


Fig. 11. Eviction Time (ET) for increasing number of VMs.

Fig. 10 shows the eviction times and the total migration times for the migration of a 5 GB busy VM running TunkRank. Since TunkRank is a write-intensive application, using pre-copy results in a significant increase in the number of retransmitted pages. Both post-copy and Scatter-Gather transmit each page at most once with eviction times almost identical to those for idle VMs.

6.2 Migration of Multiple VMs

In this section, we show that Scatter-Gather can deliver low eviction times when simultaneously migrating multiple VMs to a memory constrained destination. The eviction time for multiple VMs is the time from the beginning of the migration of the first VM to the end of the migration of the last VM. The source and the destination have 26 GB of memory. The destination runs two 5 GB VMs executing TunkRank. We migrate an increasing number of 5 GB idle and busy VMs from the source and measure the eviction time. Busy VMs run TunkRank with the same 5 GB working set. With 26 GB of memory, the destination can accommodate five VMs in its memory; that is three more VMs in addition to the two VMs already running at the destination. Therefore, the memory pressure increases at the destination as we increase the number of migrating VMs.

Fig. 11 shows the eviction time for the migration of multiple VMs running TunkRank. For the migration of up to three VMs, the destination does not experience memory pressure, therefore all techniques, except pre-copy migration of a busy VM, perform equally well. For the migration of four and five VMs, Scatter-Gather delivers a lower eviction time than pre-copy and post-copy because the link between the source and VMD remains non-congested even though the destination is under memory pressure. Busy VMs have a high page dirtying rate, therefore pre-copy migration of a busy VM experiences a higher eviction time than others. For idle VMs, few pages are modified by the VM during the migration, therefore pre-copy yields the same eviction time as post-copy. Scatter-Gather achieves roughly identical eviction times with both idle and busy VMs as it does not retransmit dirty pages.

6.3 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*

TABLE 1
Performance of Two 5 GB VMs Running Memcached at the Destination When a 5 GB Idle VM is Migrated

	Eviction Time (s), Latency (ms)	
	Rate Limit 256 Mbps	No Rate Limit
No Migration	N/A, 2.71	
Pre-copy	160.8, 6.00	109.7, 18.74
Post-copy	164.3, 6.92	109.3, 18.94
SG	49.8, 5.47	49.2, 18.66

performance of network-bound VMs running at the destination. It is well known [1] 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 assuming that the source and the intermediaries have sufficient network bandwidth to evict the VM quickly.

To avoid an adverse performance impact on other network-bound applications, a common 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 tradeoff 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 [28] server. Each VM caches a 3 GB Twitter dataset in its memory and responds to query and update requests from an external client. We simultaneously migrate a 5 GB idle VM from the source to the destination. The quality of service (QoS) guarantee for the Memcached benchmark in CloudSuite [19] is specified as 95 percent of the requests being executed within 10 ms. During migration, the incoming migration traffic competes with the Memcached request-response traffic for the link bandwidth. Table 1 shows that, without any rate limiting for the migration, the average response times for Memcached requests received during the migration are longer than 10 ms under all three migration approaches. When we rate-limit the migration at 256 Mbps, Memcached performance improves with the QoS specifications for all migration schemes. However, for pre-copy and post-copy, rate limiting the VM migration increases the VM eviction time by almost 1.5 times. In contrast, rate-limiting the gather phase of Scatter-Gather does not increase the eviction time. As the scatter phase is not rate limited, the source can evict the VM's memory to intermediaries at the maximum available bandwidth.

6.4 Eviction Time versus Application Degradation

In Fig. 12, we show the tradeoff between the eviction time and workload performance when a busy VM is migrated to a memory-constrained destination. The migration starts at 100 seconds and is compared with pre-copy, post-copy and Scatter-Gather migration. On the source, we run a 5 GB VM hosting a Redis [29], [30] database server, which contains a 4.5 GB dataset. The dataset is queried from a Yahoo Cloud Servicing Benchmark (YCSB) [31] client running on an external host. We measure the performance of YCSB in terms of operations per second. The destination has 12 GB of memory and it executes two 5 GB TunkRank VMs to create memory pressure. We chose YCSB/Redis applications for this experiment because YCSB displays the performance of the Redis server periodically, which allows us to continuously monitor the performance of the VM during the migration.

With post-copy and Scatter-Gather migration, the performance of YCSB remains low throughout the migration, at around 300 ops/s. The severe degradation is caused by a large number of page faults and also because the destination is under memory pressure. YCSB is particularly an adversarial workload for post-copy and Scatter-Gather. It queries the entire 4.5 GB of Redis dataset and competes with the migration traffic at the destination's network interface, thereby delaying the retrieval of the faulted pages. With pre-copy migration, the performance of YCSB also degrades, but not as much as post-copy and Scatter-Gather, and remains fairly stable. This is because the source host is not under memory pressure, unlike the destination, and the application is not slowed by network-bound page faults. Although the VM eviction traffic competes with the YCSB request/response traffic at the source's network interface, any resulting performance degradation is limited compared to post-copy and Scatter-Gather.

However, when looking at eviction time, it can be observed that pre-copy has the longest eviction time due to retransmission of dirtied pages, which is worsened by the active YCSB workload. In contrast, eviction time with Scatter-Gather is almost 6.6 times lower, at 57 seconds. Thus, even though the performance degradation using Scatter-Gather is higher than with pre-copy, the eviction time is significantly lower than both pre-copy and post-copy. This tradeoff between eviction speed and application performance should be a major consideration in selecting the migration technique to use in any situation.

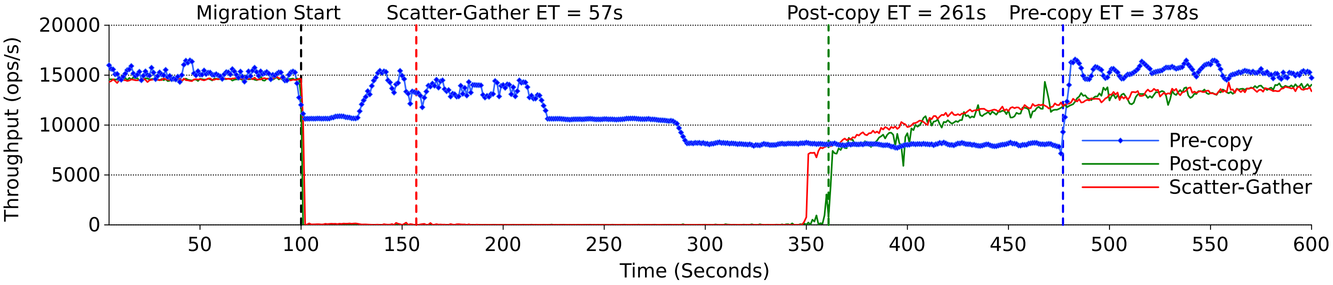


Fig. 12. Comparison of YCSB throughput. YCSB queries a 4.5 GB Redis dataset inside the 5 GB migrating VM.

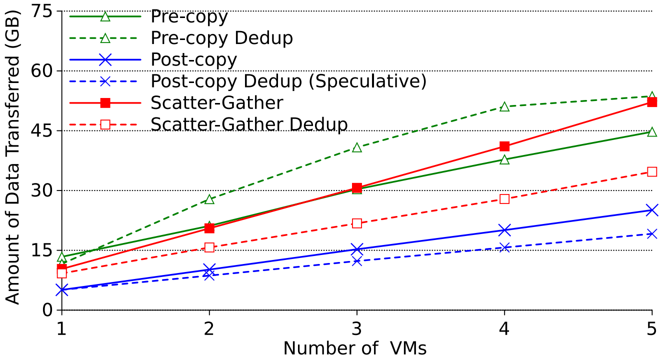


Fig. 13. The amount of data transferred for the migration of multiple 5 GB busy VMs with and without deduplication.

6.5 Deduplication Effectiveness

Scatter-Gather has an implicit tradeoff between lower eviction time and higher network overhead. Every scattered page is transmitted twice over the network, once to the intermediary and then to the destination. In Sections 4.4 and 5.3, we introduced deduplication as a technique to reduce the network overhead when migrating multiple VMs.

In this section, we compare the amount of data transferred and the eviction time with all three techniques with and without deduplication. For Scatter-Gather migration, the VMD client at the source deduplicates the identical pages during the migration, whereas for pre-copy, the QEMU at the source performs deduplication. Using deduplication with post-copy requires a significant implementation effort which is not the focus of this paper. Hence, we estimate the amount of data that would be transferred if post-copy were to use deduplication. Our estimates for post-copy are derived by measuring the number of identical pages present in the VM's memory at the start of the migration. The corresponding eviction time cannot be reliably estimated for post-copy with deduplication because it depends on multiple factors such as the computational overhead of deduplication, rate of pre-paging, the number of remote page faults, fault servicing latency, etc; these cannot be accurately determined except during runtime.

For the following experiments, the source and destination have 26 GB of memory. The destination runs two VMs executing TunkRank. We migrate an increasing number of 5 GB busy VMs from the source and observe the eviction time and the amount of data transferred. The migrating VMs also run the same TunkRank application.

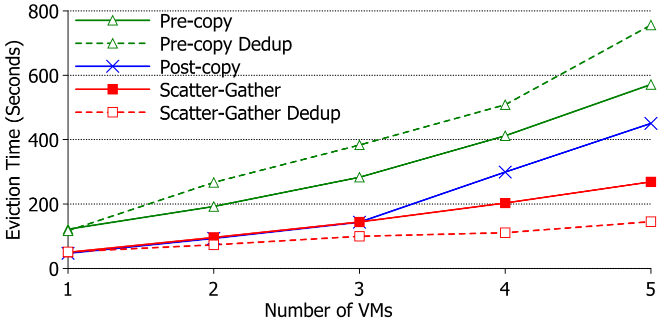


Fig. 14. Eviction time for the migration of multiple 5 GB busy VMs with and without deduplication.

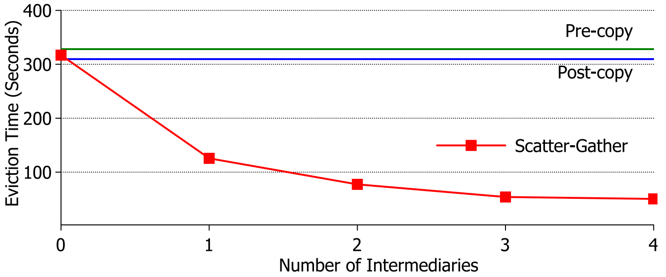


Fig. 15. Reduction in eviction time when using multiple intermediaries, each adding 200 Mbps of bandwidth.

Fig. 13 shows the amount of data transferred with all three techniques and their variants that use deduplication. Note that for pre-copy, contrary to intuition, deduplication during the migration increases the amount of data transferred. Deduplication requires identifying the identical pages so that migration can avoid their duplicate transfer. However, this is a time consuming process and it slows down the migration, thus allowing VMs to modify even more pages. These pages are then retransmitted in pre-copy, increasing its network overhead. Post-copy transfers the lowest amount of data, followed by Scatter-Gather. Deduplication further reduces the amount of data transferred for both the techniques.

Fig. 14 shows the corresponding eviction times (except post-copy with deduplication). Pre-copy has the highest eviction time due to large amount of data transferred during the migration. Deduplication further increases its eviction time due to additional time spent in identifying identical pages. Scatter-Gather yields the lowest eviction time among all three techniques while deduplication further reduces its eviction time.

6.6 Eviction Time with Multiple Intermediaries

Fig. 15 shows the migration of one idle 5 GB VM to a memory constrained destination while increasing the number of intermediaries. The destination has 12 GB of memory and hosts two 5 GB VMs executing TunkRank. During migration, the destination swaps out pages to accommodate the incoming VM. Each intermediary is bandwidth constrained and can only provide 200 Mbps of bandwidth during the scatter phase.

When migrating without the use of intermediaries, Scatter-Gather functions just like post-copy. Therefore, both post-copy and Scatter-Gather migration result in high eviction time when migrating to a resource constrained destination. Since the VM is idle, performance of pre-copy is also comparable to that of post-copy and Scatter-Gather.

When migrating with the use of intermediaries, Scatter-Gather migration uses the maximum available reception bandwidth at the destination for direct transfer of the VM memory state and any available reception bandwidth at the intermediaries to further speed up the eviction. Therefore, each additional intermediary allows the source to offload the VM's memory state faster until its transmission bandwidth limit is reached. Pre-copy and post-copy do not use intermediaries and hence their eviction time remains constant.

7 Related Work

To the best of our knowledge, Scatter-Gather live migration is the first approach[2] that aims to reduce VM eviction time when the destination is resource constrained. Here we review the literature related to lowering the total migration time, checkpoint/restart, and applications of post-copy.

7.1 Lowering Total Migration Time

Post-copy [3], [4] migration provides a lower total migration time and network overhead for write-intensive applications compared to pre-copy [1], [2]. Work in [20], [33], and [34] optimize the live migration of multiple VMs using techniques such as memory compression, memory deduplication, and controlling the sequence of VMs being migrated. XvMotion [35] integrates a number of optimizations for memory and storage migration over the local and wide area networks. Optimizations such as ballooning [3], [36], dropping the guest cache [37], [38], deduplication [16], [20], [21], [22], [23], [24], and compression [16], [39], can lower the network traffic and total migration time. The above optimizations are orthogonal to our contributions.

7.2 Relationship to Checkpoint/Restore

All virtualization platforms [41], [42], [43] include a checkpoint/restore functionality. Traditionally, restoration is performed only after the checkpoint operation is complete, resulting in a large downtime. Scatter-Gather live migration can be viewed as a combination of live post-copy migration plus checkpoint/restore via intermediaries; the checkpointing (scatter) phase proceeds concurrently with the restoration (gather) phase, yielding lower downtime while matching the eviction time of traditional checkpoint/restore. Remus [44] provides high-availability by capturing high-frequency VM snapshots at a backup site using a variant of pre-copy. While the VM can be quickly restored in case of failure, high-frequency snapshots can impose high overheads for write-intensive workloads.

7.3 Post-Copy and Its Applications

Post-copy was first proposed in [3] for the Xen platform and subsequently also implemented in [4] for the KVM/QEMU platform. SnowFlock [18] uses post-copy to clone VMs and execute them simultaneously on multiple hosts to run HPC applications. Jettison [5], [17] 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 [11] by quickly eliminating hotspots. Traffic-sensitive migration [45] monitors the traffic at the source and the destination hosts to select either pre-copy or post-copy for co-migrating VMs. Guide-Copy [46] runs the migrating VM's context at the source to guide which pages are pro-actively sent to the destination. We propose a variant of post-copy in Scatter-Gather where pre-paging actively fetches pages from intermediaries and demand-paging fetches faulted pages from source.

8 Conclusions

Eviction time of VMs has not been considered as an explicit performance metric in traditional live VM migration approaches, especially, when the destination host is slow in receiving the state of migrating VMs. 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 destination hosts during migration; the source scatters the memory pages of migrating VMs to multiple intermediaries from where the destination gathers these pages. Our approach introduces a variant of post-copy migration, supports the simultaneous live eviction of multiple VMs, uses a distributed memory virtualization layer to stage the VMs' memory, and employs cluster-wide deduplication. In evaluations, Scatter-Gather migration reduces VM eviction time by up to a factor of 6 while maintaining comparable total migration time against traditional pre-copy and post-copy.

Acknowledgments

Work supported in part by National Science Foundation (CNS-1320689, CNS-0845832, CNS-0855204, CCF-1005153, CNS-1040666) and ED GAANN Fellowship. Research conducted at Binghamton University, Binghamton, NY.

References

[1] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. Netw. Syst. Des. and Implementation*, 2005, pp. 273–286.

[2] M. Nelson, B. H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proc. USENIX Ann. Tech. Conf.*, 2005, p. 25.

[3] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *SIGOPS Operating Syst. Rev.*, vol. 43, no. 3, pp. 14–26, 2009.

[4] T. Hirofuchi and I. Yamahata, "Yabusame: Postcopy live migration for Qemu/KVM," presented at KVM Forum, 2011.

[5] N. Bila, E. J. Wright, E. D. Lara, K. Joshi, H. A. Lagar-Cavilla, E. Park, A. Goel, M. Hiltunen, and M. Satyanarayanan, "Energy-Oriented partial desktop virtual machine migration," *ACM Trans. Comput. Syst.*, vol. 33, no. 1, pp. 2:1–2:51, Mar. 2015.

[6] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. IFIP/IEEE Symp. 10th Int. Symp. Integr. Netw. Manag.*, May 2007, pp. 119–128.

[7] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," in *Proc. 9th ACM/IFIP/USENIX Int. Conf. Middleware*, 2008, pp. 243–264.

[8] T. Das, P. Padala, V. Padmanabhan, R. Ramjee, and K. G. Shin, "LiteGreen: Saving energy in networked desktops using virtualization," presented at USENIX Ann. Tech. Conf., 2010.

[9] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu, "Delivering energy proportionality with non Energy-Proportional Systems - Optimizing the ensemble," in *Proc. Conf. Power Aware Comput. and Syst.*, 2008, p. 2.

[10] A. Jaikar, D. Huang, G.-R. Kim, and S.-Y. Noh, "Power efficient virtual machine migration in a scientific federated cloud," *Cluster Comput.*, vol. 18, no. 2, pp. 609–618, 2015.

[11] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, "Reactive consolidation of virtual machines enabled by postcopy live migration," in *Proc. Fifth Int. Workshop Virtualization Technol. in Distrib. Comput.*, Jun. 2011, pp. 11–18.

[12] S. Setty and G. Tarasuk-Levin, "vMotion in VMware vSphere 5.0: Architecture, performance and best practices," in *Proc. VMworld 2011,*, Las Vegas, Nevada, USA, 2011, p. 24.

[13] S. Loosemore, R. Stallman, R. McGrath, A. Oram, and U. Drepper, *The GNU C Library Reference Manual*, 2001.

[2] Shorter version [32] of this work appears in IEEE Cloud 2014.

[14] U. Deshpande, B. Wang, S. Haque, M. Hines, and K. Gopalan, "MemX: Virtualization of Cluster-Wide memory," in *Proc. 39th Int. Conf. Parallel Processing*, Sep. 2010, pp. 663–672.

[15] J. Hwang, A. Uppal, T. Wood, and H. H. Huang, "Mortar: Filling the gaps in data center memory," presented at *10th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ. (VEE)*, 2014.

[16] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines," in *Proc. 20th Int. Symp. High Performance Distrib. Comput.*, Jun. 2011, pp. 135–146.

[17] 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 *Proc. Eurosys: Seventh ACM Eur. Conf. Comput. Syst.*, Apr. 2012, pp. 211–224.

[18] H. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan, "SnowFlock: Rapid virtual machine cloning for cloud computing," in *Proc. EuroSys: Fourth ACM Eur. Conf. Comput. Syst.*, 2009, pp. 1–12.

[19] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *Proc. ASPLOS: 17th Int. Conf. Architectural Support for Program. Languages and Operating Syst.*, 2012, pp. 37–48.

[20] U. Deshpande, B. Schlinker, E. Adler, and K. Gopalan, "Gang migration of virtual machines using Cluster-wide deduplication," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing*, May 2013, pp. 394–401.

[21] U. Deshpande, U. Kulkarni, and K. Gopalan, "Inter-rack live migration of multiple virtual machines," in *Proc. Virtualization Technol. in Distrib. Comput.*, Jun. 2012, pp. 19–26.

[22] S. A. Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "VMFlock: Virtual machine Co-Migration for the cloud," in *Proc. Sixth Int. Workshop High Perform. Distrib. Comput.*, Jun. 2011, pp. 159–170.

[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 *Proc. EURO-PAR: 17th Int. Conf. Parallel Process.*, Sep. 2011, pp. 431–442.

[24] 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. Seventh ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, Mar. 2011, pp. 121–132.

[25] OpenSSL SHA1 [Online]. Available: http://www.openssl.org/docs/crypto/sha.html, 2015.

[26] D. P. Bertsekas, "Constrained optimization and lagrange multiplier methods," *Computer Science and Applied Mathematics*, vol. 1, Boston: Academic Press, 1982.

[27] F. Chabaud and A. Joux, "Differential collisions in SHA-0," in *Proc. 18th Ann. Int. Cryptology Conf.*, Aug. 1998, pp. 56–71.

[28] D. Interactive, *Memcached: Distributed Memory Object Caching* [Online]. Available: http://www.danga.com/memcached/, 2015.

[29] *Introduction to Redis* [Online]. Available: http://redis.io/topics/introduction, 2015.

[30] J. L. Carlson, *Redis in Action*. Greenwich, CT, USA: Manning Publications Co., 2013.

[31] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. First ACM Symp. Cloud Comput.*, 2010, pp. 143–154.

[32] U. Deshpande, Y. You, D. Chan, N. Bila, and K. Gopalan, "Fast server deprovisioning through Scatter-Gather live migration of virtual machines," in *Proc. IEEE Seventh Int. Conf. Cloud Computing*, Anchorage, AK, USA, 2014, pp. 376–383.

[33] R. K. Hui Lu, Cong Xu and D. Xu, "vHaul: Towards optimal scheduling of live Multi-VM migration for Multi-tier applications," in *Proc. IEEE Eighth Int. Conf. Cloud Comput.*, New York, NY, USA, Jun. 2015, pp. 453–460.

[34] H. Liu and B. He, "VMbuddies: Coordinating live migration of Multi-Tier applications in cloud environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 1192–1205, Apr. 2015.

[35] A. J. Mashtizadeh, M. Cai, G. Tarasuk-Levin, R. Koller, T. Garfinkel, and S. Setty, "XvMotion: Unified virtual machine migration over long distance," in *Proc. USENIX Ann. Tech. Conf.*, 2014, pp. 97–108.

[36] C. A. Waldspurger, "Memory resource management in VMware ESX server," in *Proc. Fifth Symp. Operating Syst. Des. and Implementation*, Dec. 2002, pp. 181–194.

[37] C. Jo, E. Gustafsson, J. Son, and B. Egger, "Efficient live migration of virtual machines using shared storage," in *Proc. Ninth ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, Mar. 2013, pp. 41–50.

[38] S. Akiyama, T. Hirofuchi, R. Takano, and S. Honiden, "Fast wide area live migration with a low overhead through page cache teleportation," in *Proc. Int. Symp. Cluster, Cloud and Grid Comput. (CCGrid)*, 2013, pp. 78–82.

[39] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive memory compression," in *Proc. IEEE Int. Conf. Cluster Computing and Workshops*, Aug. 2009, pp. 1–10.

[40] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[41] VMWare Inc, Architecture of VMWare ESXi [Online]. Available: http://www.vmware.com/files/pdf/esxi_architecture.pdf, 2007.

[42] 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 Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.

[43] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: The linux virtual machine monitor," in *Proc. Linux Symp.*, Jun. 2007, pp. 225–230.

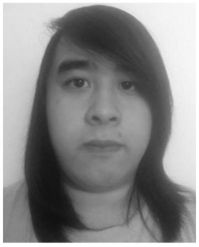
[44] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: High availability via asynchronous virtual machine replication," in *Proc. Fifth USENIX Symp. Netw. Syst. Des. and Implementation*, Apr. 2008, pp. 161–174.

[45] U. Deshpande and K. Keahey, "Traffic-Sensitive live migration of virtual machines," in *Proc. IEEE/ACM 15th Int. Symp. Cluster Comput. and the Grid Environ.*, May 2015, pp. 51–60.

[46] J. Kim, D. Chae, J. Kim, and J. Kim, "Guide-Copy: Fast and silent migration of virtual machine for datacenters," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage and Anal. (Supercomputing)*, 2013, pp. 1–12.

Umesh Deshpande received the BE degree in computer science from Pune University, India, the MS and PhD degrees in computer science from Binghamton University. He is a research staff member at IBM Research—Almaden. Earlier, he worked at Calsoft Inc. (India) in storage virtualization. His research interests include VM migration, memory deduplication, and memory virtualization.

Danny Chan received the BS degree in computer science from California State University, Fullerton, he is currently working toward the master's degree in computer science at Binghamton University. His research interests include virtualization and operating systems. During summer 2013, he was part of the National Science Foundation's Research Experience for Undergraduates (REU) program at Binghamton University where he worked on VM migration. He is currently a GAANN fellow at Binghamton University.



Steven Chan is currently working toward the graduation degree in computer science at University of California, Irvine. His research interests include virtualization, operating systems, and networking. During summer 2014, he was part of the National Science Foundation's Research Experience for Undergraduates (REU) program at Binghamton University where he worked on VM migration.

Kartik Gopalan received the BE degree from Delhi Institute of Technology, MS degree from Indian Institute of Technology, Chennai, and the PhD degree from Stony Brook University. He is an associate professor in computer science at Binghamton University. His research interests include virtualization, operating systems, security, and networks. He is a recipient of the National Science Foundation CAREER Award. He has been a faculty in computer science at Florida State University, Lead Architect and Developer at Rether Networks Inc., and senior software engineer at Wipro Global R&D.

Nilton Bila received the BSc degree from Trent University, the PhD and MSc degrees from the Department of Computer Science at the University of Toronto. He is a research staff member at IBM Thomas J. Watson Research Center. His research interests include systems dependability, operational analytics, virtualization, and power management. He has been a visiting scholar at the School of Computer Science at Carnegie Mellon University and spent a summer at Bell Labs.