Tackling Memory Footprint Expansion During Live Migration of Virtual Machines

Roja Eswaran, Mingjie Yan, Kartik Gopalan
{reswara1,myan28,kartik}@binghamton.edu
Computer Science, Binghamton University
Binghamton, NY, USA

***Abstract*—Live migration is widely used in cloud platforms to transfer Virtual Machines (VMs) from one physical machine to another. Live migration is useful for workload consolidation, load balancing, failure management, and energy savings. Copy-on-write (COW) page sharing allows identical pages to be shared both within a VM and across co-located VMs to reduce their collective memory footprint. Current live migration techniques are not aware of such page sharing; thus they do not preserve pre-existing page sharing when migrating VMs to a common destination machine. Consequently, each shared page is replicated at the destination multiple times as if they were separate pages. This expansion of the memory footprint of VMs during migration can lead to problems such as migration failure, increased network traffic, and longer migration times. We propose Sharing-aware Live Migration (SLM), which preserves pre-existing COW page sharing within and across VMs that are migrated to a common destination machine. The key idea is to identify guest pages that are mapped to the same physical page at the source machine and to map them to the same physical page at the destination. We present SLM technique for both pre-copy and post-copy live migration of multiple VMs and describe its implementation on the KVM/QEMU virtualization platform. Our evaluations show that SLM successfully preserves pre-existing COW page sharing during migration, eliminates the risk of migration failure due to memory expansion, and reduces total migration time and network traffic overhead.**

***Index Terms*—Cloud Computing, Live Migration, Operating System, Deduplication, Copy-on-Write Page Sharing.**

I. Introduction

Virtual machines (VMs) boost the resource usage efficiency of data centers by allowing the co-location of multiple VMs on the same physical machine while preserving their functional isolation. In this paper, we focus on the intersection of two essential techniques for managing co-located VMs: live migration and copy-on-write (COW) page sharing. Live migration [5], [17], [15] is a key technology in data centers that transfers running VMs from one physical machine to another. It is widely used for a variety of purposes, such as load balancing [2], [18], [39], meeting service level agreements [33], energy savings [41], and seamless maintenance of physical servers.

Co-located VMs may often need to be migrated to the same destination machine for various reasons. For example, co-located VMs that run different components of a multi-tier application [19] may need to be migrated to the same destination machine to maintain low inter-VM communication latency [43], [25], [38], [21], [27] or meet other QoS targets [46]. Additionally, physical server availability, hardware availability, and multi-tenancy limitations may necessitate the migration of co-located VMs to the same destination machine.

COW page sharing (both within a VM and across co-located VMs) is often used by the hypervisor's memory management system to reduce the collective memory footprint by sharing identical pages, whenever doing so is feasible and safe. For example, Kernel Samepage Merging (KSM) [1] is a Linux kernel feature that identifies identical memory pages among VMs that run the same guest OS or similar applications and maps them to the same physical page through COW page sharing. As KSM continuously identifies and merges identical pages across different VMs, the memory footprint of co-located VMs progressively decreases. Another example of inter-VM page sharing is when a common VM template image is used to quickly boot up multiple lightweight VMs; the base template image is mapped COW into each VM's memory [42], [24], [36], [44]. As VMs execute and write to (dirty) different pages of their memory, those pages are copied for the respective VM, and their memory footprints diverge over time.

Unfortunately, current live VM migration techniques are unaware of pre-existing COW page sharing across VMs that are being migrated to the same destination. As a result, shared pages are transferred and replicated multiple times at the destination, as if they were separate pages, resulting in an expanded memory footprint at the destination. This expansion can lead to migration failures when the destination lacks sufficient memory to accommodate the additional footprint of the VMs that were comfortably co-located at the source. The duplication of previously shared pages also results in longer migration times and increased network traffic, potentially affecting the performance of other network-bound workloads in the cluster. Previous approaches [8], [7], [47], [20], [4], [12] either use content hashing to detect and avoid the retransmission of identical pages during migration, but do not detect pre-existing shared pages, or work only for the limited case of pre-copy migration of templated VM instances.

In this paper, we address the general problem of preserving all pre-existing page sharings among multiple co-located VMs as they are live migrated together to a common destination machine. Our goal is to prevent the expansion of VMs' memory footprint at the destination for both pre-copy and post-copy live migration, for all types of VMs, irrespective of the

underlying page sharing mechanisms. The contributions of this work are as follows:

1) We identify and demonstrate the problem of memory footprint expansion caused by traditional live migration techniques, specifically both pre-copy [5] and post-copy [17]. This expansion occurs because these techniques lack awareness of pre-existing COW page sharing among co-located VMs, both within and across VMs.
2) We then present a *Sharing-aware Live Migration* (SLM), which identifies and preserves all types of pre-existing page sharing resulting from techniques such as KSM, VM templating, or others. SLM adds COW-awareness to both pre-copy and post-copy live migration.
3) We implement and evaluate SLM for both pre-copy and post-copy migration in the KVM/QEMU [22] virtualization platform using several workloads and microbenchmarks. Besides preserving all pre-existing page sharings at the destination machine, SLM reduces the total migration time by up to 59% and network traffic by up to 62%.

While not the focus of this paper, we note that side-channels [16], [26], [6], [45] might exploit memory sharing among mutually untrusting VMs and solutions exist to mitigate these risks [35], [23]. This paper assumes that appropriate mitigation strategies are deployed when page sharing is used, such as by sharing pages only among mutually trusting VMs [35]. Further, memory being a bottleneck resource, safe page sharing among mutually trusting VMs is important to retain consolidation and multiplexing benefits of virtualization. Finally, while we use the KVM/QEMU platform to demonstrate our techniques in this paper, the core conceptual ideas of our solution are applicable to other hypervisors as well.

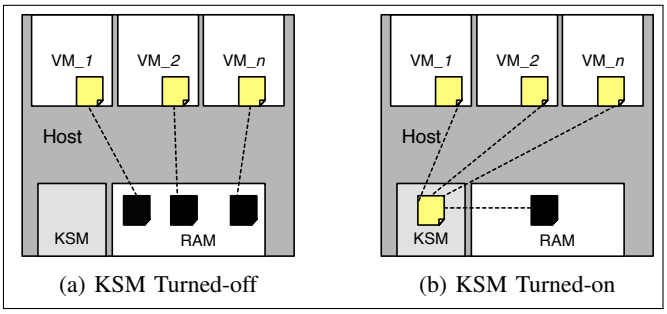
In the rest of this paper, we first present the background on pre-copy and post-copy live migration techniques and demonstrate the problem of memory footprint expansion during live migration. Next we present the design and implementation SLM followed by its evaluation. The paper concludes with a discussion of related work and summary of contributions.

II. Background and Problem

In this section, we first provide background on pre-copy and post-copy live migration and inter-VM memory sharing. Then we motivate the problem of sharing-aware live migration.

A. Pre-copy and Post-copy Live Migration

Pre-copy live migration [5], [34] is the most common technique to migrate VMs from a source machine to a destination machine. It works by first transferring the VM's memory pages to the destination, even as the VM continues running at the source, and then transfers the CPU execution state at the end; hence the name *pre*-copy which means to transfer memory before CPU state. However, as the VM's memory is being transferred, its virtual CPUs (VCPUs) may write to previously transferred pages, thus *dirtying* them again and requiring their retransmission.



(a) KSM Turned-off (b) KSM Turned-on

Fig. 1: (a) Without KSM, each virtual page has its own physical page in RAM. (b) With KSM, duplicated pages are merged into single virtual page, COW-mapped to single physical page in RAM.

For the traditional pre-copy technique, the VM's memory is transferred over multiple rounds. The first pre-copy round is the longest since it transfers all pages of the VM. The second round transfers only the pages dirtied in the first round; the third stage transfers only pages dirtied from the second round, and so forth. When the number of remaining dirtied pages is small enough, the migration process switches to the final *downtime* stage in which all the VCPUs of the VM are paused at the source, and the remaining dirtied pages, VCPU states, and I/O device states are sent to the destination. Finally, the VCPUs are resumed at the destination, and the VM starts execution where it left off at the source.

Post-copy live migration [17] is another technique that first transfers a VM's VCPUs to the destination, resumes them there, and then transfers the VM's memory pages from the source. The VM's pages are transferred by two concurrent mechanisms: (a) active-push of the pages from the source to the destination, with preference for pages in the VM's working set, and (b) remote demand paging by the destination from the source when a VM's VCPU faults on a page that has not yet been transferred from the source. Post-copy aims to reduce the number of remote page faults by pushing pages before the VM accesses them at the destination.

B. Inter-VM Memory Sharing

Many *virtual* pages, both within a single VM and across multiple co-located VMs, may be mapped to a common physical page due to memory reduction optimizations implemented by either the host OS or the hypervisor. For instance, upon a fork operation, the host OS may map the child process' pages COW with its parent. Newly allocated pages may be mapped to a common *zero* page until they are first written to. Or the hypervisor may use deduplication techniques to perform COW sharing of identical pages across VMs.

Kernel Samepage Merging (KSM) [10], [37] is a technique which performs memory deduplication among co-located VMs to fit more VMs into physical memory. Many duplicated pages exist when the same guest OS and applications run in different co-located VMs. Without KSM, as shown in Figure 1(a), multiple identical virtual pages of VMs are mapped to their

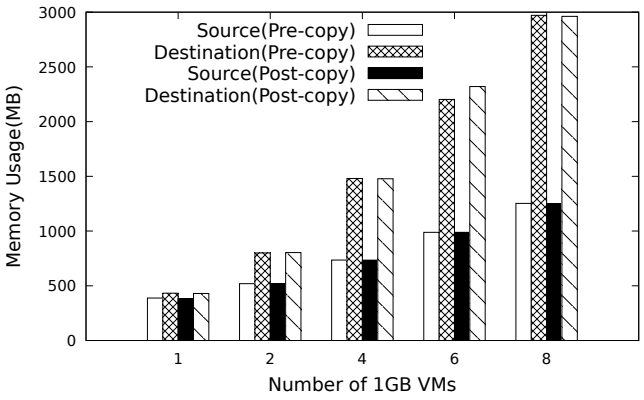


Fig. 2: Memory footprint of VMs expands at destination after both pre-copy and post-copy live migration, because pages shared among VMs at the source are replicated for each VM at the destination.

own physical pages resulting in increased memory usage. In contrast, KSM regularly scans the memory of all VMs, identifies identical pages, and replaces them with a single COW-shared page, as shown in Figure 1(b). In addition, COW page sharing between VMs can also arise if they are started from a common template image which is mapped COW into each VM's memory [42], [36], [44].

C. Problem: Lack of Sharing Awareness in Live Migration

Traditional pre-copy and post-copy live migration techniques are unaware of underlying COW page sharing among co-located VMs. Hence, they transfer each shared physical page multiple times to the destination, storing multiple copies of each previously shared page. Thus the memory footprint of the migrated VMs ends up being larger at the destination than it was at the source, also resulting in more network traffic and longer total migration time.

To experimentally demonstrate this problem, we measured the memory footprint of multiple concurrent 1GB VMs at both the source and destination machines after migrating the VMs using traditional pre-copy and post-copy techniques. KSM is used at the source machine to deduplicate the memory of co-located VMs before migration begins, thus establishing pre-existing COW-shared pages across VMs. The actual memory usage of each VM in this experiment is smaller than their maximum 1GB permitted since the VMs are not yet using their full allocation. Figure 2 shows that both pre-copy and post-copy, which are unaware of existing COW-shared pages among VMs, result in a larger memory footprint at the destination than at the source after live migration completes.

While one can deduplicate again (say, using KSM) at the destination machine to reestablish page sharing and reduce the overall memory usage, this can take several minutes to converge depending on how aggressively KSM is configured to scan pages [37]. There is also a more severe possibility that, when migrating multiple VMs to the same destination, some VM migrations might fail due to a temporary lack of memory at the destination. However, sufficient memory exists if pre-existing page sharings from the source were faithfully reproduced at the destination during migration. Even if the destination has enough memory and the migration succeeds, it will cause higher memory pressure at the destination, more network traffic, and longer total migration time.

III. Sharing-aware Live Migration (SLM)

We now present the design of a more general SLM technique, which preserves all pre-existing COW page sharings among co-located VMs being migrated concurrently. SLM is designed to operate effectively with both pre-copy and post-copy algorithms. The key insight behind SLM is that irrespective of the underlying page-sharing mechanism (such as KSM, VM templating, fork, or others), multiple COW-mapped guest pages will map to the same page in the physical memory. SLM examines the physical address of each guest page being transferred, identifies COW-mapped shared pages at the source node, and avoids transmitting them multiple times to the destination. Instead, such shared pages at the source node before migration are mapped to the same physical page at the destination node.

The traditional pre-copy migration transfers the memory pages of a VM over several rounds, where the initial round transfers the entire memory of the VM, while the subsequent rounds only transfer pages that the VM has dirtied (i.e., written to) in the previous rounds. This dirtying operation during live migration may break pre-existing COW mappings at the source. SLM for pre-copy is designed to detect when such COW mappings break at the source across multiple pre-copy rounds and to disassociate the corresponding COW-mapped pages at the destination. On the other hand, post-copy migration transfers each page only once and, since the VM executes at the destination, there are no dirtied pages at the source to retransmit.

As shown in Algorithm 1, 2 and Figure 3, SLM operates on both the source and the destination nodes. At the source node, SLM classifies the type of each page (Unique, Duplicate, or Dirty) and transfers them to the destination according to their type. At the destination, SLM receives each page's information and maps it accordingly into the VMs' memory. We describe these steps at the source and destination in more detail below.

A. Identifying Page Type at Source

As illustrated in Algorithm 1, SLM follows a two-step process for each page transfer. In the first step, SLM determines the page's physical frame number (PFN) and the virtual page number (VPN). This information is stored in a hash table for efficient lookup during subsequent transfers. In the second step, SLM categorizes pages into one of the three types, as outlined in Table I, based on the presence or absence of the PFN and VPN in the hash table.

1) *Unique Page:* Pages that have not been transferred yet are considered Unique. In this scenario, the corresponding PFN and VPN are not present in the hash table.

Algorithm 1 SLM: Source

Input:

- N is the total number of pages in a VM.
- $page[N]$ is the array of all pages.
- $vpn\_list[N]$ is the array of Virtual Page Numbers (VPN).
- $pfn\_list[N]$ is the array of Page Frame Numbers (PFN).

```
1: function MIGRATE(VM)                                   ▷ Source
2:     for i ← 1 to N do
3:         Find VPN and PFN of page[i]
4:         if vpn not in vpn_list then
5:             Append vpn to vpn_list
6:             if pfn not present in pfn_list then
7:                 Append pfn to pfn_list                 ▷ Unique page
8:                 Send pfn of page[i]
9:                 Send page[i]
10:            else                                       ▷ Duplicate page
11:                Send pfn of page[i]
12:            end if
13:        else                                           ▷ Dirty page
14:            Send page[i]
15:        end if
16:    end for
17: end function
```

Algorithm 2 SLM: Destination

Input:

- N is the total number of pages in a VM.
- $page[N]$ is the array of all pages.
- $identical[N]$ is the array of identifiers for pages.
- $offset\_list[N]$ is the array of mmap_offsets from a memory-backend-file.

```
1: function RECEIVE(VM)                                   ▷ Destination
2:     for i ← 1 to N do
3:         mmap_offset ← 0
4:         Receive identifier for page[i]
5:         if identical[i] = 0 then                       ▷ Unique page
6:             Append mmap_offset to offset_list
7:             Mmap page[i] with mmap_offset
8:             Receive page[i] from the network
9:             Increment mmap_offset
10:            COW-protect page[i]
11:        else if identical[i] = 1 then                  ▷ Duplicate page
12:            Retrieve mmap_offset from offset_list
13:            Mmap page[i] with mmap_offset
14:            COW-protect page[i]
15:        else                                           ▷ Dirty page
16:            Receive page[i] from the network
17:        end if
18:    end for
19: end function
```

PFN	VPN	Page Type
✗	✗	Unique
✓	✗	Duplicate
✓/✗	✓	Dirty

TABLE I: Determining page type using PFN and VPN

Thus, both the PFN and VPN are inserted into the hash table before sending the page content.

2) *Duplicate Page:* Pages that have already been transferred are considered Duplicate. In this case, the PFN is present in the hash table, but the VPN is not. Therefore, only the VPN is inserted into the hash table before sending the PFN.
3) *Dirty Page:* Pages that require retransmission due to being dirty in the previous pre-copy round are referred to as Dirty pages. The PFN may or may not be present in the hash table, but the VPN is present. In this case, only the PFN is inserted into the hash table before sending the page content. Since post-copy only transfers pages once, this type of page doesn't exist for SLM post-copy.

For Unique and Dirty pages, SLM transfers the entire page, including the page type and its PFN at the source, as a unique identifier. However, for Duplicate pages, SLM does not send the page content but only the page type and the PFN.

B. Preserving COW Sharing at Destination

SLM at the destination node works as shown in Algorithm 2. At its core, the algorithm operates in two ways depending on whether a full page or only a source PFN is received from the source node. If a full page is received, it is copied into the corresponding VM's memory, and the source PFN is recorded for future reference. On the other hand, if only the source PFN is received, the corresponding virtual page in the VM is mapped to the previously received physical page with the same source PFN.

To facilitate COW sharing at the destination, we set up an in-memory backend-file into which each received page is memory mapped using the `mmap` system call [31]. There are two flags of importance, `MAP_SHARED` and `MAP_PRIVATE`. The `MAP_SHARED` flag causes any writes to a mapped virtual address to be written back to the backend-file. On the other hand, `MAP_PRIVATE` results in COW mapping, meaning that any writes to the mapped virtual address result in the allocation of a new private page to the process before the write is committed, ensuring that the write is not transmitted to the backend-file.

For SLM pre-copy, when a Unique page is received, the entire backend-file is initially configured with the `MAP_SHARED` flag. The received page is then written to the backend-file, and the `mmap` configuration for that page is changed to `MAP_PRIVATE` to enable COW mapping for any future duplicates of the same page. The received PFN and its corresponding mmap offset are also recorded in a hashtable. When a Duplicate page is received, SLM retrieves the corresponding mmap offset from the hashtable using the received

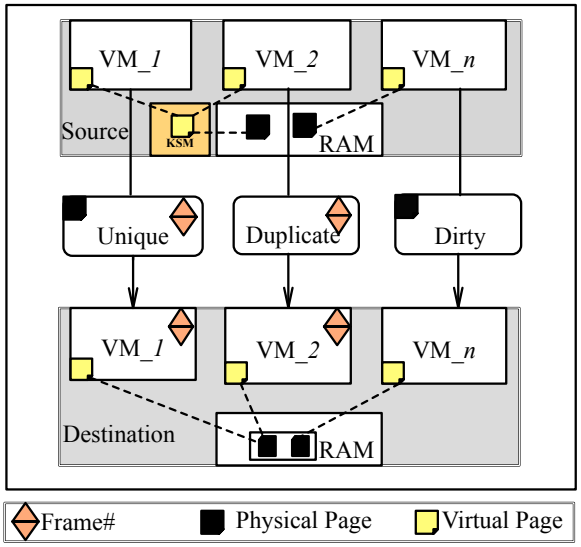


Fig. 3: SLM classifies pages of VMs at the source as Unique, Duplicate, or Dirty. Duplicate pages are not re-transmitted; instead, the destination COW maps them into the same physical page of the memory-backend-file located within RAM.

PFN and maps the virtual address to the backend-file using the `MAP_PRIVATE` flag. If a Dirty page is received, SLM skips any `mmap` operations and copies the entire page content directly from the network into the VM's address space.

In SLM post-copy, the migration thread is tasked with copying page content from the network socket, whether received through active-pushing or demand-paging. This algorithm operates in three stages: (1) For a Unique page type, the migration thread directly copies the temporary page to the backend-file and records the received PFN and its corresponding mmap offset in the hash table. For Duplicate page types, the migration thread retrieves the mmap offset from the hash table using the received PFN. (2) The migration thread maps the virtual address to the backend-file using the `mmap` system call and configures the page as `MAP_PRIVATE`. (3) Finally, if a VCPU accessing this page was suspended due to a page fault, the migration thread wakes it up. At the destination, if the VMs introduce any new duplicated pages in the future, KSM continues to deduplicate them.

C. Retrieval and Tracking of PFN

All virtual pages mapped to a shared physical page must have the same PFN, irrespective of which sharing mechanism generates such mapping. QEMU is a user-level management process whose address space has specific regions reserved for guest memory [22]. To get the guest's physical address of a page (VPN), we could directly access the addresses that are part of the reserved region.

The Linux kernel exposes page table information to userspace using `/proc/pid/pagemap` [40]. With this file, a userspace process can find the PFN for a specific VPN. Each entry in the pagemap contains 64-bit information indexed by the VPN, with the first 56 bits indicating the PFN. SLM takes advantage of the pagemap in the pseudo file system to retrieve accurate PFNs for each VPN, enabling the determination of the page type. SLM uses hashtable at the source for page type and at destination for $PFN \rightarrow mmap\_offset$ mappings. A new mmap offset entry is inserted into the table using PFN as a key every time a Unique page arrives since they are guaranteed to have new page content. Whenever a Duplicate page arrives, SLM looks up the hashtable to retrieve the corresponding mmap offset using its PFN as the key and maps the VPN to the respective mmap offset with COW protection. Finally, SLM skips the lookup during the arrival of Dirty pages.

D. Synchronization Across Multiple VMs

One synchronization challenge we encountered relates to the order of arrival of Unique and Duplicate page information. In an ideal scenario, for a given PFN, a Unique page (comprising both its page content and PFN) should arrive at the destination before any Duplicate page (containing only the PFN). But, during the migration of multiple VMs, there are instances where the PFN for a Duplicate page arrives for a VM (e.g., VMx) before the Unique page content for the corresponding PFN arrives for another VM (e.g., VMy). However, the Duplicate page cannot be COW-mapped until the Unique page is mapped and its content is written into the backend-file.

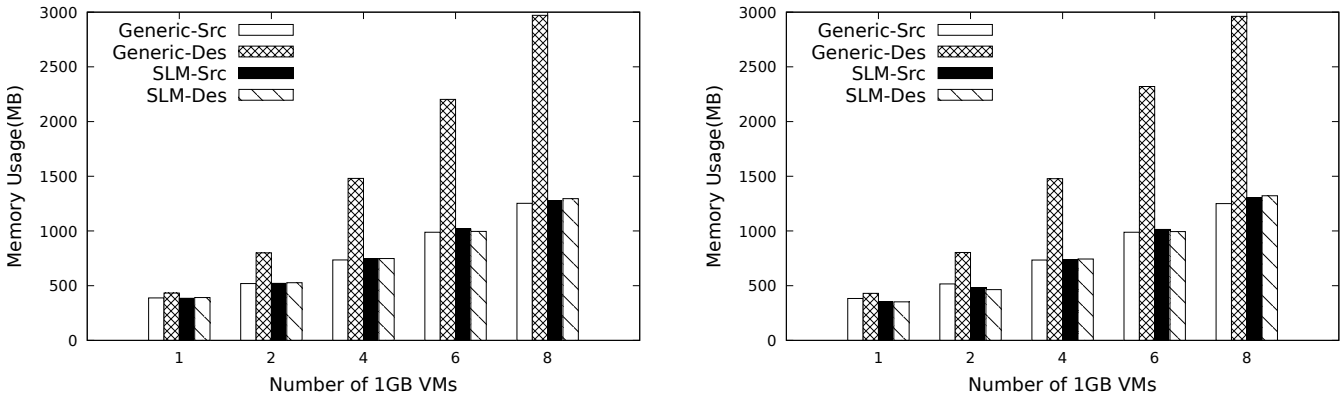
For SLM post-copy, when a Duplicate page information arrives before its Unique page, QEMU for VMx busy waits, anticipating the arrival of the Unique page with the expectation that the waiting time will be short. When the Unique page with page content for VMy arrives, QEMU for VMx proceeds to COW map the Duplicate page.

For SLM pre-copy, VMy's page content may change in subsequent pre-copy rounds. We update all pending Duplicate pages in VMx that depend on VMy at the end of each pre-copy round through busy waiting to prevent stale mapping entries. Busy waiting at the end of the last pre-copy round, just before downtime, can extend VM downtime. To mitigate this latency, SLM includes an additional live pre-copy round without busy waiting before the downtime phase. This issue doesn't arise with SLM post-copy since pages are sent only once in post-copy.

IV. Evaluation

We now evaluate the performance of SLM against traditional pre-copy and post-copy live migration methods. Our experimental setup consists of three machines, each equipped with two Intel Xeon E5-2620 v2 processors and 128GB of DRAM, running Ubuntu. We implemented SLM versions of pre-copy and post-copy in the KVM/QEMU [22] virtualization platform on Linux. We modified QEMU's default pre-copy and post-copy algorithms with no changes to the guest operating system in the VMs. We used VMs of varying sizes, ranging from 1GB to 32GB, as needed. Each experiment was repeated at least five times to calculate average values. Our key performance metrics for evaluation are as follows:

- **Memory Usage:** The collective memory footprint of the VMs at the source (before migration) and destination



(a) Pre-copy

(b) Post-copy

Fig. 4: Comparison of memory footprint at source vs. destination when multiple VMs are migrated concurrently using generic vs. SLM (a) pre-copy and (b) post-copy. We observe a significant increase in memory footprint for generic pre-copy/post-copy, but no significant increase for SLM pre-copy/post-copy.

(after migration). This is measured using the `free` command and includes the memory used by both the QEMU processes and the VMs. Recording the `use` column value from the `free` command before and after spawning a process gives the memory usage of that process.

- **Total Migration Time (TMT):** The total migration time (TMT) refers to the time from the start to the end of the entire migration process. For single VM migration, in pre-copy, TMT is measured from the start of migration on the source machine to when the VM resumes on the destination machine. In post-copy, it is measured from the initiation of migration on the source to the release of the VM's resources at the source after all pages have been transferred. For multiple VM migration, in pre-copy, TMT is calculated from the beginning of the first VM's migration on the source to the resumption of the last VM on the destination. In post-copy, it is calculated from the start of the first VM's migration to the release of the last VM's resources at the source.
- **Downtime:** Downtime refers to the period during which a VM's execution is fully suspended during live migration. In pre-copy, downtime is used to transfer the VM's remaining Dirty pages, I/O device states, and VCPU states to the destination. In post-copy, the processor state and the essential execution state necessary to start the VM on the destination are transferred during downtime.
- **Network Traffic Reduction:** The reduction in the total number of pages transferred during live migration by eliminating the transfer of COW-shared pages.
- **Application performance degradation:** The extent to which live migration slows down the performance of an application running inside VMs during migration.

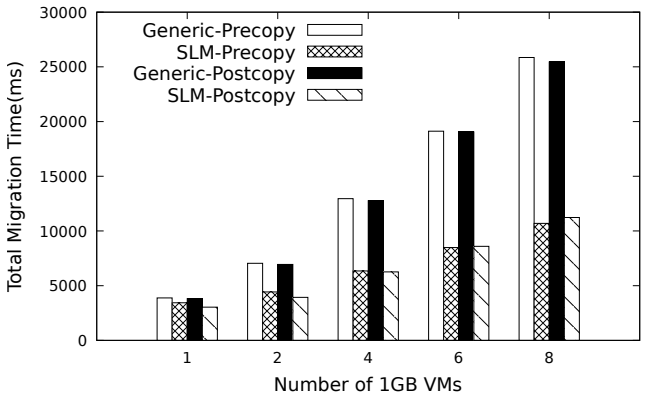
We demonstrate that migration of VMs using SLM maintains memory footprint of VMs and application performance during migration, besides reducing TMT and network traffic.

In order to accurately measure TMT and downtime in QEMU, we employ a more precise method instead of solely relying on the source QEMU's measurement (which we found to be inaccurate). Specifically, we send UDP packets to a third, separate measurement node, at the start of migration and at the end of migration. For downtime measurement and pre-copy TMT measurement, the start message is sent from source node and the end message is sent from the destination node, whereas for post-copy TMT measurement, both messages are sent from the source node. The measurement node observes the arrival times of these packets using the *tcpdump* tool, and the difference in these arrival times represents TMT or downtime. This method provides more accurate timings, as the dedicated measurement node has a better view of the end-to-end live migration timeline than the source node alone.

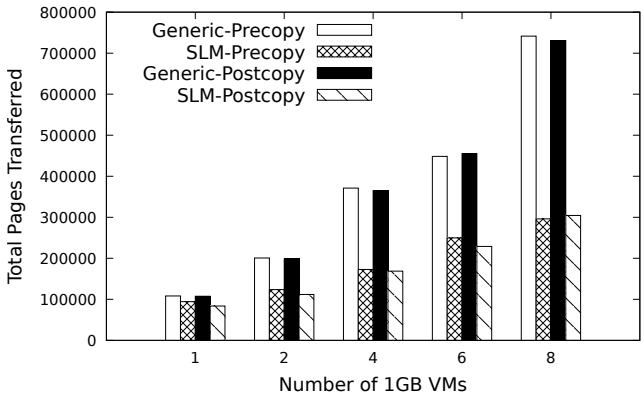
Throughout the evaluation, we use the term *generic* to refer to the traditional versions of pre-copy and post-copy. In our experiments with KSM, we initiate live migration only after allowing the KSM daemon [10] to run for a sufficient period of time, ensuring that total memory usage has stabilized. This stabilization is confirmed by monitoring the output of the `free` command in the host system. Doing this ensures that our results capture all COW-shared pages among VMs.

A. Memory Footprint of VMs After Migration

Figures 4(a) and (b) compare the memory footprint at source vs. destination when multiple VMs are migrated concurrently using generic vs. SLM techniques for pre-copy and post-copy. The X-axis indicates the number of concurrent 1GB VMs, and the Y-axis displays their collective memory footprint. Generic live migration, despite reducing memory usage at the source through KSM, leads to a significantly expanded memory footprint at the destination because the live migration



(a) Total Migration Time (TMT)



(b) Total Pages Transferred (TPT)

Fig. 5: (a) Total Migration Time (b) Total Pages Transferred of multiple VMs concurrently migrated using generic vs. SLM pre-copy/post-copy. We observe a significant increase in both TMT and TPT for generic pre-copy/post-copy vs. no significant increase for SLM pre-copy/post-copy.

mechanism is unaware of COW-shared pages among VMs at the source. In contrast, SLM preserves any pre-existing COW page mappings at the destination for both pre-copy and post-copy, resulting in no significant memory expansion at the destination. For SLM, slight differences in memory footprint between source and destination are due to differences in memory usage of the QEMU process associated with VM.

B. TMT, Downtime, and Network Traffic Reduction

In this section, we evaluate the performance of concurrently migrating multiple idle VMs between two hosts in terms of TMT, downtime, and network traffic reduction. We increase the number of VMs keeping the memory size of each VM constant at 1GB.

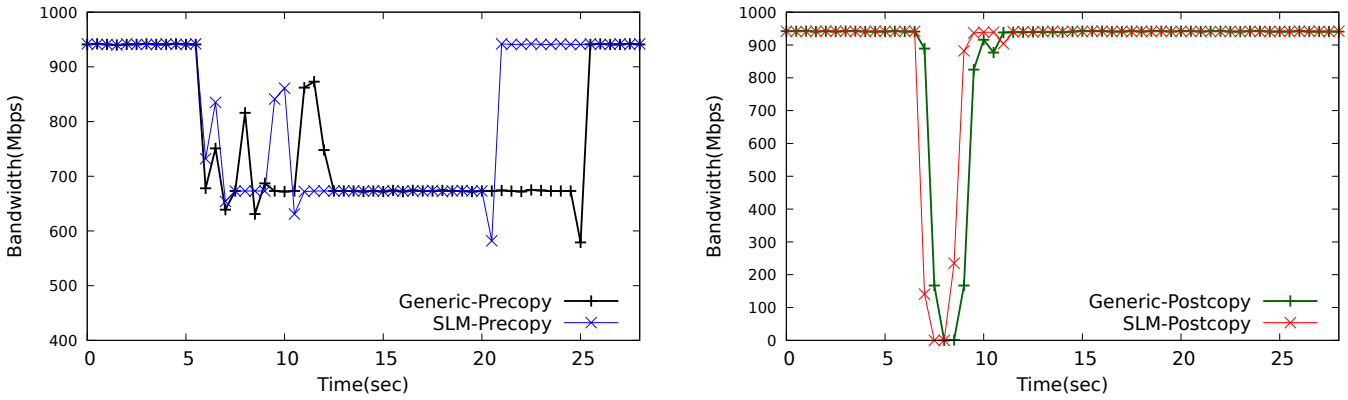
TMT is compared between generic and SLM versions of pre-copy and post-copy in Figure 5 (a). The X-axis is the number of concurrent 1GB VMs being migrated, and the Y-axis shows the TMT in milliseconds. The results show up to 59% and 57% reduction in TMT for SLM pre-copy and post-copy, respectively, compared to their generic coureduction in is due to SLM eliminating the retransmission ofnterparts. This reduction in is due to SLM eliminating the retransmission of COW-shared pages from source to destination. Figure 5 (b) compares the total number of pages transferred during generic and SLM pre-copy and post-copy. The experiments show a reduction of up to 60% and 62% in total pages transferred for SLM pre-copy and post-copy, respectively.

We compare downtime of 8 idle 1GB VMs during their concurrent migration using generic and SLM versions of pre-copy and post-copy. The maximum number of pages transferred during downtime is capped at 512 (2MB). The results indicate that VMs experience a comparable average downtime of around 93ms for generic pre-copy and 96ms for SLM pre-copy. Generic and SLM post-copy transfer minimal processor states and non-pageable memory, causing downtime of around 290ms and 300ms respectively. The higher downtime of post-copy for both generic and SLM versions may be attributed to various factors including VCPU thread invocation and demand-paging, leading to more remote page faults at resumption time. Application-observed downtimes for non-idle VMs (discussed later) tend to be higher than these numbers for idle VMs because of network state recovery.

C. Network Bandwidth Using iPerf

VM migration is a network-intensive procedure that can lead to network contention between the migration process and the applications running inside the VM. To measure the available bandwidth for the VM's applications during migration, we use iPerf [11], a network-intensive application benchmark. An iPerf server is set up on a third machine (i.e., neither the source nor the destination) within the same network, while the iPerf client is run inside the VM being migrated. The client then sends data to the server during migration through a TCP connection. All these machines are connected using a Gigabit Ethernet switch and the link is shared between the host and VM.

Figures 6 (a) and (b) show network bandwidth measurements during live migration using pre-copy and post-copy techniques. At the beginning of the migration, both pre-copy techniques experienced a drop in network bandwidth from 940 Mbps to approximately 650 Mbps. However, the decline was more significant for both post-copy versions when the bandwidth dropped close to zero due to downtime. The sudden drop to 670 Mbps in both pre-copy versions is attributed to network contention between the migration thread and the iPerf client running inside the VM. Meanwhile, the fluctuations are a result of QEMU's optimization for zero pages, where only 8 bytes are sent to indicate a zero page instead of the entire page, freeing up bandwidth for iPerf traffic. In contrast, in both versions of post-copy, the fluctuations are due to page faults caused by the post-copy thread resulting in the retrieval of



(a) Pre-copy

(b) Post-copy

Fig. 6: iPerf bandwidth for Generic and SLM pre-copy/post-copy.

pages from the source via demand-paging and active-pushing of pages from the source to avoid network faults.

SLM pre-copy migration was completed in 14.8s, whereas generic took a longer time, approximately 19.1s. This reduction in TMT for SLM pre-copy is due to a reduced number of pages that needed to be transferred, a consequence of eliminating redundant transfers of COW-shared pages. However, the active-pushing nature of both generic and SLM post-copy techniques, in tandem with demand-paging, aided in the faster recovery of network bandwidth for both post-copy methods. SLM post-copy required approximately 2s, whereas generic took around 3.5s to complete their migration process. They reached full bandwidth faster without significant fluctuations when compared to their pre-copy counterparts. While SLM pre-copy experienced a downtime of approximately 182ms, generic pre-copy exhibited a comparatively lower downtime of around 106ms. This overhead is due to the busy waiting synchronization, as detailed in section III.D. Although both SLM and generic versions of post-copy exhibit a similar downtime of around 500ms due to network state recovery, this duration becomes significant when compared to their pre-copy counterparts. Our SLM technique for both pre-copy and post-copy doesn't introduce any significant overhead in terms of application performance; in fact, it reduces TMT by eliminating the redundant transfer of shared pages.

D. Redis Cluster Benchmark

Redis is a real-world in-memory key–value database. Redis cluster is a way to run a Redis server by evenly distributing data across multiple nodes. We set up all three Redis cluster nodes as one Redis cluster server. Each VM is configured with 4GB RAM sharing a gigabit link and running a Redis cluster node instance. The Redis cluster server contains 5 million random key-value data entries, which are evenly distributed across all three nodes. We use Redis-Benchmark [28] to emulate 50 clients sending a `GET` command that randomly reads key-value data from the target Redis cluster server. Our experiments used a Redis pipeline of 16, allowing clients to send concurrent requests without waiting for server responses [29].

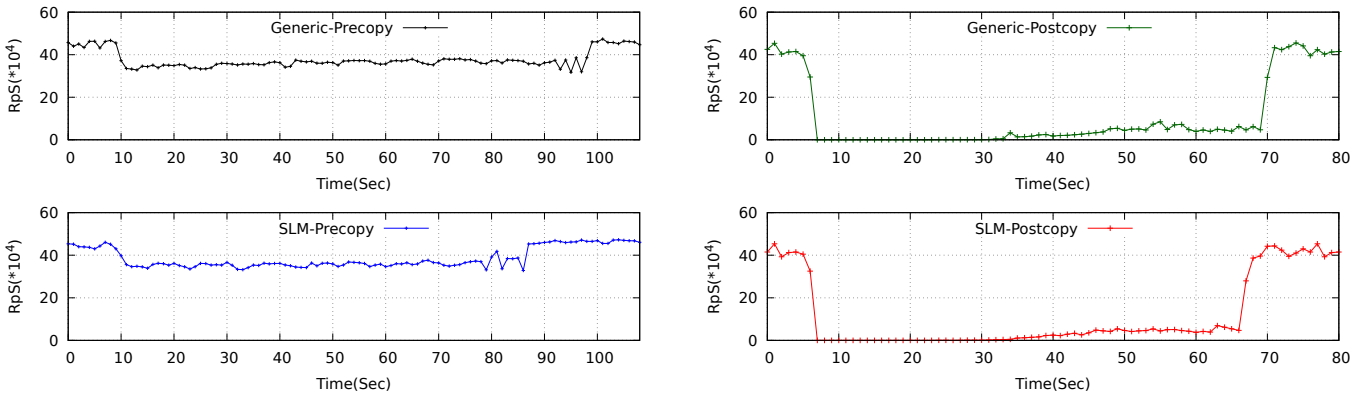
To ensure a fair comparison, we synchronized the start time of migration for Figures 7 (a) and (b). Initially, Redis demonstrated comparable throughput with both pre-copy and post-copy migration. However, during pre-copy migration, Redis experienced a 40% reduction in throughput due to network contention with migration traffic. Conversely, Redis' throughput during post-copy migration dropped to zero and consistently remained lower than pre-copy, primarily due to downtime and remote page faults which resulted in fetching pages across the network. During the downtime, with pre-copy, there were three brief drops in throughput towards the end of migration whereas. With post-copy, Redis experienced a significant downtime causing a complete disconnection.

In SLM versions of both pre-copy and post-copy, the advantages of COW page sharing are preserved during migration, resulting in a reduced TMT compared to their generic counterparts. SLM pre-copy took approximately 77s to complete migration, whereas generic pre-copy required 87s with almost the same application-level downtime of around 5s. Due to multiple remote page faults, SLM post-copy took 58s to complete migration, while generic post-copy took about 65s. While post-copy had shorter TMT, the additional pages required by demand paging and active-push mechanisms took significantly longer to fetch, leading to a 30-second application-level downtime before full throughput was restored.

V. Related Work

We first discuss existing techniques for memory footprint reduction among co-located VMs within a single node followed by works related to page sharing in live migration.

Page sharing within a single node: Disco [3] was one of the first systems to propose and implement transparent page sharing to map multiple identical virtual pages to a single physical page. Satori [32], modifies guest OSes to identify sharing opportunities and communicate them to the hypervisor.



(a) Pre-copy

(b) Post-copy

Fig. 7: Redis-cluster read throughput when migrating 3 VMs using generic and SLM (a) pre-copy and (b) post-copy. RpS stands for Responses per Second.

KSM [1] uses a red-black tree indexed with page content to find identical pages. Unlike Disco, it doesn't require any modification to the guest OS and doesn't need hash computation of page content. Performance of KSM depends on the location of the identical pages in the virtual address space. Since KSM sequentially looks for the potential candidate for merging, the further down the pages in the process address space, the less likely they will be merged. Difference Engine [14], in addition to standard COW full-page sharing, also supports sub-page level sharing and compression to improve memory savings through deduplication. Catalyst [13] offloads the identification of identical pages for deduplication to a GPU and eliminates sequential scanning of pages. Several techniques have been developed to efficiently launch multiple lightweight VMs from a common template image, which is COW-mapped into each VM's memory [24], [9], [30], [44], [36].

Live Migration with Page Sharing: Traditional pre-copy and post-copy [5], [17] are unaware of preexisting COW page sharings at the source node. Because of this limitation, they send identical pages multiple times as if they are different, causing bloated memory footprint at the destination besides higher TMT and network traffic. Several prior techniques [8], [7], [47] have used content hashing to find identical and similar memory pages across multiple VMs to reduce/eliminate their transfer during live migration. While hashing-based techniques may be useful to identify identical pages that are not COW-shared, they do not preserve *pre-existing COW page sharing* among co-located VMs when pages are transferred to a common destination. Work in [12] addresses pre-copy migration of templated VM instances with page sharing. However it does not work for non-templated VMs, does not work with post-copy migration, and does not address pages shared via other mechanisms (such as COW mappings using KSM, process fork, and mmap). In contrast, SLM addresses all types of page sharing, works with both templated and non-templated VMs, and works with both pre-copy and post-copy.

6. Conclusion

In this paper, we addressed the problem that traditional live VM migration techniques do not preserve COW page sharing among co-located VMs. The resulting expanded memory footprint at the destination can lead to failed migrations, longer migration times, and increased network traffic. We presented the design, implementation, and evaluation of Sharing-aware Live Migration (SLM) to address this problem for both pre-copy and post-copy. SLM preserves all pre-existing page sharings among VMs at the destination machine irrespective of the underlying sharing mechanism. Our evaluation of SLM on the KVM/QEMU platform shows that SLM not only prevents memory footprint expansion but also significantly reduces the migration time by up to 59% and the amount of data transferred by up to 62% with no significant impact on application performance.

References

[1] Andrea Arcangeli, Izik Eidus, and Chris Wright. Increasing memory density by using KSM. In *Proc. of the Linux Symposium*, pages 19–28, 2009.

[2] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing SLA violations. In *Proc. of IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128, 2007.

[3] Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems (TOCS)*, 15(4):412–447, 1997.

[4] Jui-Hao Chiang, Han-Lin Li, and Tzi-cker Chiueh. Introspection-based memory de-duplication and migration. *ACM SIGPLAN Notices*, 48(7):51–62, 2013.

[5] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005.

[6] Andreas Costi, Brian Johannesmeyer, Erik Bosman, Cristiano Giuffrida, and Herbert Bos. On the effectiveness of same-domain memory deduplication. In *Proc. of European Workshop on System Security*, pages 29–35, 2022.

[7] Umesh Deshpande, Brandon Schlinker, Eitan Adler, and Kartik Gopalan. Gang migration of virtual machines using cluster-wide deduplication. In *Proc. of International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2013.

[8] Umesh Deshpande, Wang Xiaoshuang, and Kartik Gopalan. Live gang migration of virtual machines. In *Proc. of High Performance Parallel and Distributed Computing (HPDC)*, pages 135–146, 2011.

[9] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[10] Izik Eidus and Hugh Dickins. Kernel Samepage Merging. https://www.kernel.org/doc/Documentation/vm/ksm.txt, 2009.

[11] ESNet/LBNL. iPerf: The TCP/UDP bandwidth measurement tool. https://iperf.fr.

[12] Roja Eswaran, Mingjie Yan, and Kartik Gopalan. Template-aware live migration of virtual machines. In Proc. of ACM/IEEE Symposium on Edge Computing EdgeComm Workshop, 2023.

[13] Anshuj Garg, Debadatta Mishra, and Purushottam Kulkarni. Catalyst: GPU-assisted rapid memory deduplication in virtualization environments. In *Proc. of ACM International Conference on Virtual Execution Environments (VEE)*, 2017.

[14] Diwaker Gupta, Sangmin Lee, Michael Vrable, Stefan Savage, Alex C Snoeren, George Varghese, Geoffrey M Voelker, and Amin Vahdat. Difference engine: Harnessing memory redundancy in virtual machines. In *Communications of the ACM*, volume 53, pages 85–93. ACM New York, NY, USA, 2010.

[15] Jacob Gorm Hansen and Eric Jul. Self-migration of operating systems. In *Proc. of ACM SIGOPS European Workshop*, 2004.

[16] David Hildenbrand, Martin Schulz, and Nadav Amit. Copy-on-pin: The missing piece for correct copy-on-write. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[17] Michael R Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *ACM SIGOPS Operating Systems Review*, 43(3):14–26, 2009.

[18] Jinhua Hu, Jianhua Gu, Guofei Sun, and Tianhai Zhao. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *International Symposium on Parallel Architectures, Algorithms and Programming*, 2010.

[19] Dong Huang, Bingsheng He, and Chunyan Miao. A survey of resource management in multi-tier web applications. *IEEE Transactions on Communications Surveys & Tutorials*, 2014.

[20] Changyeon Jo, Erik Gustafsson, Jeongseok Son, and Bernhard Egger. Efficient live migration of virtual machines using shared storage. *ACM SIGPLAN Notices*, 48(7):41–50, 2013.

[21] Shinji Kikuchi and Yasuhide Matsumoto. Impact of live migration on multi-tier application performance in clouds. In *Proc. of IEEE International Conference on Cloud Computing*, 2012.

[22] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. Kvm: the linux virtual machine monitor. In *Proc. of the Linux Symposium*, 2007.

[23] Santhosh Kumar T, Debadatta Mishra, Biswabandan Panda, and Nayan Deshmukh. CoWLight: Hardware assisted copy-on-write fault handling for secure deduplication. In *Proc. of Intl. Workshop on Hardware and Architectural Support for Security and Privacy*, 2019.

[24] Horacio Andrés Lagar-Cavilla, Joseph Andrew Whitney, Adin Matthew Scannell, Philip Patchin, Stephen M Rumble, Eyal De Lara, Michael Brudno, and Mahadev Satyanarayanan. SnowFlock: Rapid virtual machine cloning for cloud computing. In *Proc. of the ACM European Conference on Computer Systems (EuroSys)*, 2009.

[25] Seung-Hwan Lim, Jae-Seok Huh, Youngjae Kim, and Chita R Das. Migration, assignment, and scheduling of jobs in virtualized environment. In *Proc. of USENIX Hotcloud Workshop*, 2011.

[26] Jens Lindemann and Mathias Fischer. A memory-deduplication side-channel attack to detect applications in co-resident virtual machines. In *Proc. of the Annual ACM Symposium on Applied Computing (SAC)*, pages 183–192, 2018.

[27] Haikun Liu and Bingsheng He. Vmbuddies: Coordinating live migration of multi-tier applications in cloud environments. *IEEE Transactions on Parallel and Distributed Systems*, 2014.

[28] Redis Ltd. Redis benchmark. https://redis.io/docs/management/optimization/benchmarks/.

[29] Redis Ltd. Redis pipelining. https://redis.io/docs/manual/pipelining/.

[30] Costin Lupu, Radu Nichita, Doru-Florin Blânzeanu, Mihai Pogonaru, Răzvan Deaconescu, and Costin Raiciu. Nephele: Extending virtualization environments for cloning unikernel-based VMs. In *Proc. of the ACM European Conference on Computer Systems (EuroSys)*, 2023.

[31] Man-pages. Memory-Mapping. https://man7.org/linux/man-pages/man2/mmap.2.html.

[32] Grzegorz Miłós, Derek G Murray, Steven Hand, and Michael A Fetterman. Satori: Enlightened page sharing. In *Proc. of USENIX Annual Technical Conference (ATC)*, pages 1–1, 2009.

[33] Saad Mubeen, Sara Abbaspour Asadollah, Alessandro Vittorio Papadopoulos, Mohammad Ashjaei, Hongyu Pei-Breivold, and Moris Behnam. Management of service level agreements for cloud services in IoT: A systematic mapping study. *IEEE access*, 6:30184–30207, 2017.

[34] Michael Nelson, Beng-Hong Lim, and Greg Hutchins. Fast transparent migration for virtual machines. In *Proc. of USENIX Annual Technical Conference (ATC)*, pages 391–394, 2005.

[35] Fangxiao Ning, Min Zhu, Ruibang You, Gang Shi, and Dan Meng. Group-based memory deduplication against covert channel attacks in virtualized environments. In *Proc. of IEEE Trustcom/BigDataSE/ISPA*, pages 194–200, 2016.

[36] Open Infrastructure Foundation. Kata Containers. https://katacontainers.io/.

[37] Shashank Rachamalla, Debadatta Mishra, and Purushottam Kulkarni. Share-o-meter: An empirical analysis of KSM based memory sharing in virtualized systems. In *International Conference on High Performance Computing*, 2013.

[38] Yi Ren, Renshi Liu, Qi Zhang, Jianbo Guan, Ziqi You, Yusong Tan, and Qingbo Wu. An efficient and transparent approach for adaptive intra-and inter-node virtual machine communication in virtualized clouds. In *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2019.

[39] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. VM live migration at scale. *ACM SIGPLAN Notices*, 53(3):45–56, 2018.

[40] The Linux kernel user's and administrator's guide. Examining process page tables. https://www.kernel.org/doc/html/latest/admin-guide/mm/pagemap.html.

[41] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pMapper: Power and migration cost aware application placement in virtualized systems. In *Proc. of Middleware*, 2008.

[42] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C Snoeren, Geoffrey M Voelker, and Stefan Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, pages 148–162, 2005.

[43] Jian Wang, Kwame-Lante Wright, and Kartik Gopalan. XenLoop: A transparent high performance inter-VM network loopback. In *Proc. of High Performance Parallel and Distributed Computing (HPDC)*, 2008.

[44] Kun Wang, Jia Rao, and Cheng-Zhong Xu. Rethink the virtual machine template. *ACM SIGPLAN Notices*, 46(7):39–50, 2011.

[45] Jidong Xiao, Zhang Xu, Hai Huang, and Haining Wang. Security implications of memory deduplication in a virtualized environment. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013.

[46] Fei Xu, Fangming Liu, Linghui Liu, Hai Jin, Bo Li, and Baochun Li. iaware: Making live migration of virtual machines interference-aware in the cloud. *IEEE Transactions on Cloud Computing*, 2013.

[47] Xiang Zhang, Zhigang Huo, Jie Ma, and Dan Meng. Exploiting data deduplication to accelerate live virtual machine migration. In *Proc. of IEEE International Conference on Cluster Computing*, 2010.