SDN-based Order-aware Live Migration of Virtual Machines

Dinuni Fernando Ping Yang Hui Lu
Department of Computer Science, State University of New York at Binghamton, New York, USA, 13850
Email: {dferna15,pyang,huilu}@binghamton.edu

***Abstract*—Live migration is a key technique to transfer virtual machines (VMs) from one machine to another. Often multiple VMs need to be migrated in response to events such as server maintenance, load balancing, and impending failures. However, VM migration is a resource intensive operation that pressures the CPU, memory, and network resources of the source and destination hosts as well as intermediate network links. The live migration mechanism ends up contending for finite resources with the VMs that it needs to migrate, which prolongs the total migration time and worsens the performance of applications running inside the VMs. In this paper, we propose SOLive, a new approach to reduce resource contention between the migration process and the VMs being migrated. First, by considering the nature of VM workloads, SOLive manages the order in which multiple VMs are migrated to significantly reduce the total migration time. Secondly, to reduce the network contention between the migration process and the VMs, SOLive uses a combination of software-defined networking-based resource reservation and scatter gather-based VM migration to quickly deprovision the source host. A prototype implementation of our approach in KVM/QEMU platform shows that SOLive quickly evicts VMs from the source host with low impact on VMs' performance.**

I. Introduction

Cloud computing platforms increasingly use virtualization to improve service availability and resiliency. Live migration of virtual machines (VMs) is a key technique to migrate running VMs from one machine (called "source host") to another machine (called "destination host") in response to events such as system maintenance, load balancing, and impending failures. Most hypervisors such as VMware [1], Hyper-V [2], KVM [3], and Xen [4] support live VM migration, as do a number of service providers in datacenters and cloud infrastructure. For example, Google uses live migration in their cloud infrastructure to perform over a million migrations per month [5]. Amazon launched Server Migration Service (SMS) [6], which provides services in EC2 to migrate virtual machines running in VMware environment to the cloud.

It is not uncommon for servers or racks to fail in datacenters. For example, [7] discusses the failure rate of Google datacenters, which states that "In each cluster's first year, it's typical that 1,000 machine failures will occur; one power distribution unit will fail, bringing down 500 to 1,000 machines; 20 racks will fail, each time causing 40 to 80 machines to vanish from the network; there is about a 50 percent chance that the cluster will overheat, taking down most of the servers in less than 5 minutes". Upon imminent failure of a server, it is critical to migrate all VMs running on the server quickly to prevent the interruption of services to the users. As a result, a number of optimizations have been applied to pre-copy [8], [9] and post-copy [10], [11], two most widely used VM migration techniques, to reduce the migration time, including deduplication [12]–[15], compression [12], [16], [17], swapping [18], [19], and avoiding transfer of cached pages [20].

In cloud environment, a physical machine often hosts multiple VMs. Upon imminent failure of a physical host, all of the co-located VMs need to be migrated to other machines. *Gang migration techniques* [12], [13] were proposed to simultaneously migrate a group of co-located VMs from one machine to another. However, live VM migration itself is a resource-intensive operation since the migration process continuously utilizes CPU cycles, memory, and communication bandwidth throughout the migration. Migration of multiple VMs pressures the CPU, memory, and network resources of the source and destination hosts as well as intermediate network links. The live migration mechanism ends up contending for finite resources with the VMs that it needs to migrate, which prolongs the total migration time and worsens the performance of applications running inside the VMs. In this paper, we propose new techniques to reduce the resource contention between the migration process and the VMs being migrated.

Specifically, existing gang migration techniques often treat all VMs equally without considering the migration order based on the VMs' resource usage. However, our experimental results (Section IV) show that different VMs often have different workload characteristics (e.g., memory, network, and CPU intensiveness) and the order in which VMs are migrated out of the source may lead to significant variations in the total migration time. In this paper, we study the impact of the migration order on the total migration time and propose techniques to manage the migration order to minimize the contention between the migration process and the VMs running on the source host. Our experimental results show that our *order-aware* live migration approach can identify the migration order with significantly reduced total migration time (e.g. by 80%).

We further propose to leverage the recent advances in software-defined networking technology to dynamically reserve bandwidth on both the source and the destination nodes to reduce the total migration time. The bandwidth reservation is done through the dynamic traffic shaping (QoS) feature in Open vSwitch [21]. The bandwidth reservation algorithm must balance the total VM migration time and the performance of applications running inside the VMs in order *not* to disrupt services to users. To meet this requirement, our bandwidth reservation mechanism takes into account how urgent it is

to migrate the VM, the available bandwidth, and the impact of bandwidth reservation on the performance of applications running inside the VMs. When it is infeasible to reserve the same bandwidth on the destination as the source, we propose a scatter gather mechanism to evict VMs from the source to one or more intermediaries so that the source can still quickly evict VMs and the destination can retrieve memory pages from the intermediaries at its own pace. Our experimental results show that, with 2-3 intermediaries, our scatter gather mechanism enables us to quickly evict VMs from the source host to intermediaries (up to 48% lower eviction time).

Organization: The rest of the paper is organized as follows. Section II provides an overview of the pre-copy and post-copy migration mechanisms and the software-defined networking technology. Section III describes the architecture of SOLive, our proposed SDN-based order-aware live VM migration technique. Section IV presents algorithms for managing the VM migration order in pre-copy and post-copy. Our bandwidth reservation algorithm is given in Section V. Section VI discusses the related work and Section VII concludes the paper.

II. Background

This section provides a brief overview of live VM migration techniques and the software-defined networking technology.

A. Live VM Migration

Pre-copy [8], [9] and Post-copy [10], [11] are two widely used live VM migration techniques. They are "live" because the VM continues running during the migration.

Pre-copy live migration: With pre-copy, the VM's memory pages are transferred from the source host to the destination host over multiple iterations. The first iteration transfers all memory pages to the destination and the subsequent iterations transfer only the pages modified in the previous iterations. When the estimated downtime is less than a threshold, the source machine pauses the VM and transmits the remaining dirty pages, the hardware device state, and the CPU state to the destination. The VM is then resumed on the destination.

Post-copy live migration: In post-copy, the VM is first suspended on the source host. The CPU state is then transferred to the destination host where the VM is resumed immediately. Next, the source actively sends the remaining memory pages of the VM to the destination host. If the VM accesses a memory page that has not been received by the destination, then a page fault occurs and the source sends the faulted page to the destination. Compared to pre-copy in which a memory page may be sent to the destination multiple times, post-copy sends each page over the network only once. This yields lower network overhead for memory write-intensive workloads.

B. Software Defined Networking

Traditional datacenters separate network for tenant and management traffic. A major drawback of this approach is the growth in complexity and administrative costs as the VM population grows. As a result, many datacenters such as Google cloud [22] and Facebook [23] start using Software-defined networking (SDN) technology [24], [25] to configure and manage

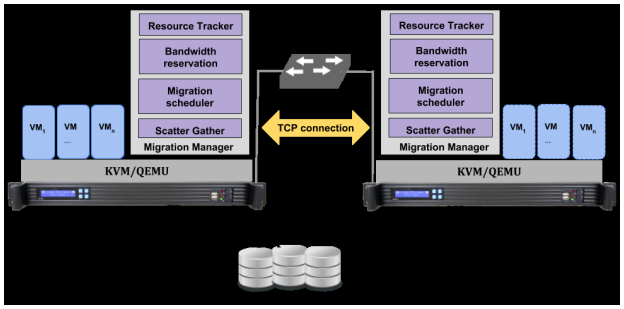


Fig. 1: The architecture of SOLive.

network infrastructure. SDN can be used to facilitate efficient network management to improve the network performance and monitoring. SDN also provides the flexibility to install packet processing rules and split network traffic dynamically through OpenFlow [25]. OpenFlow uses QoS to tune the network traffic in both inbound and outbound directions. The queuing mechanism is used to apply traffic shaping policies on outbound traffic. To control the inbound traffic, ingress policing rules are added to the network interface file. With the help of the centralized controller, OVS virtual switches and OpenFlow protocol provide the provision to dynamically vary the network bandwidth to a single or multiple routes by adding policy rules and queues to the network topology.

III. SOLive : SDN-based Order-aware Live VM Migration

Figure 1 gives the architecture of SOLive, an order-aware live VM migration technique that leverages the software-defined networking (SDN) technology to reserve bandwidth for the migration process. SOLive aims to reduce the total migration time while at the same time, minimizing the performance impact on applications running inside the VMs.

SOLive contains a *resource tracking module* that computes the resource usages of each VM prior to migration. The *migration scheduler* manages the migration order of VMs based on the resource usages of the VMs. The *bandwidth reservation module* reserves network bandwidth on both source and destination hosts based on how urgent the VMs need to be migrated, the resource usage of the VMs, and the available bandwidth. The bandwidth reservation module also monitors changes to the available bandwidth and dynamically adjusts the reserved bandwidth during the migration. When it is infeasible to reserve the same bandwidth on the destination as the source (e.g. when the destination has very high incoming network traffic), the *scatter gather module* deprovisions the VMs' memory state to one or more network middle-boxes (network intermediaries), which temporarily hold VMs' memory so that the destination node can retrieve the VMs' memory at its own pace. The scatter gather module also uses SDN to reserve the incoming network bandwidth on the intermediaries. The bandwidth reserved on each intermediate node depends on the incoming network traffic on the intermediate node, the bandwidth reserved on the source, and the number of intermediate nodes.

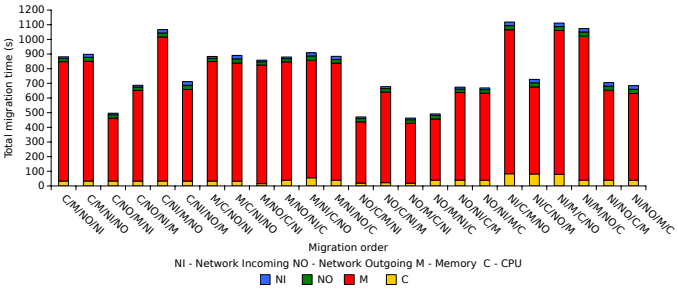


Fig. 2: Time for migrating four VMs with 24 different migration orders (pre-copy).

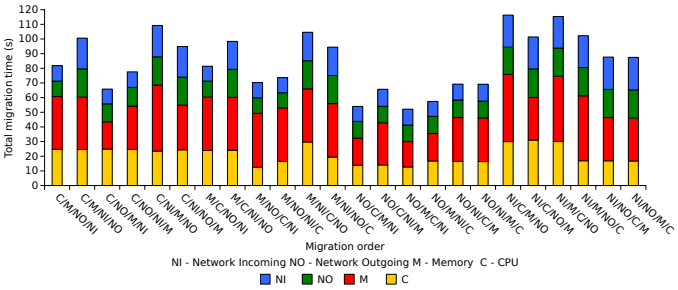


Fig. 3: Time for migrating four VMs with 24 different migration orders (post-copy).

IV. Order-aware Live VM Migration

Existing gang migration techniques [12], [13] often treat all VMs equally without considering the migration order based on the VMs' resource usage. However, different VMs may have different memory size and often show different workload characteristics such as memory, network, and CPU intensiveness. Thus, different VM migration order may lead to significant variations in the total migration time (TMT). In this section, we study the impact of the migration order on the TMT and propose techniques to manage the migration order to minimize the contention between the migration process and the VMs running on the source host.

A. Performance Impact of Migration Order

In post-copy, the time taken to migrate a VM depends on the number of the VM's non-zero pages and the migration bandwidth. Let N be the size of the VM's non-zero pages, B be the migration bandwidth, and $down$ be the downtime (i.e. the time between suspending the VM on the source and resuming the VM on the destination). The post-copy TMT can be estimated as

$$\frac{N}{B} + down \tag{1}$$

In pre-copy, the TMT can be estimated as

$$\Sigma_{i=1}^{n}(\frac{N}{B} \times (\frac{S}{B})^{i-1}) \tag{2}$$

where n is the number of iterations and S is the unique page dirty rate (i.e. the number of unique pages dirtied per second) [26]. The unique page dirty rate of a VM depends on the VM's workload, which is usually high for memory-intensive VMs with a large working set. In addition, the number of iterations is usually high when the VM's unique page dirty rate is high.

The migration bandwidth depends on the residual (i.e. unused) bandwidth available on the source and the destination hosts. The migration traffic mainly competes with the outgoing VM application traffic at the source and the incoming VM application traffic at the destination. This contention may prolong the migration time and degrade the performance of applications running inside the VM. Therefore, if a VM on the source host has high outgoing network traffic but low incoming network traffic, then migrating such a VM first may help increase the network bandwidth available for migrating the rest of the VMs and hence may reduce the TMT. Similarly, if a VM has high incoming network traffic but low outgoing network traffic, then after migrating the VM, the VM may compete with the migration process on the incoming network bandwidth and hence should be migrated at the end.

We conducted experiments to evaluate the impact of VM migration order on the pre-copy and post-copy migration. Our test environment consists of dual six-core 2.1 GHz Intel Xeon machines with $128GB$ memory connected through a Gigabit Ethernet switch with 1 Gbps full-duplex ports. A single network link is used for both migration and VM application traffic. VMs are configured with 1 vCPU and $2GB$ of memory. Virtual disks are accessed over the network from an NFS server which enables each VM to access its storage from both source and destination hosts. We used pre-copy and post-copy implementations that come bundled with KVM/QEMU version 2.5.0 to migrate VMs. Each VM runs a CPU-intensive, a memory write-intensive, an outgoing network-intensive, and an incoming network-intensive application, respectively. Each data point reported is an average of 3 runs of experiments.

CPU-intensive workload: We used Kernbench [27] as the CPU-intensive workload, which compiles Linux kernel version 4.18.16 inside the VM during the migration.

Memory write-intensive workload: We used a C program that repeatedly writes random numbers to $1.8GB$ main memory as the memory write-intensive workload. We choose this synthetic benchmark because it enables us to vary the size of the working set to dirty a large region of memory quickly.

Outgoing/incoming network-intensive workload: We used iPerf [28] as network intensive workload. To generate outgoing network traffic, the iPerf server runs on an external machine (i.e. neither source nor destination) and the iPerf client runs inside the VM. To generate incoming network traffic, the iPerf server runs inside the VM while the iPerf client runs on an external machine. The iPerf client continuously sends data to the iPerf server through a TCP connection.

In the rest of the paper, we use *NO*, *M*, *C*, and *NI* to represent the above outgoing network-intensive, memory-intensive, CPU-intensive, and incoming network-intensive VMs, respectively. Columns 2–5 in Table I give the number of non-zero pages and the page dirty rate of the four VMs, which shows that the memory-intensive VM has the highest number of unique page dirty rate and non-zero pages.

Total migration time: Figures 2 and 3 give the TMT for migrating the four VMs using pre-copy and post-copy, respec-

	NO	M	C	NI	NO1	M1	C1	NI1
Number of non-zero pages	268694	518280	334725	276913	317696	430071	392307	322825
Unique page dirty rate	103	21062	3146	1118	926	4165	1825	1502

TABLE I: The number of non-zero pages and unique page dirty rate of eight VMs.

tively, with 24 different migration orders. The x-axis specifies the migration order of VMs. For example, *NO/NI/C/M* specifies that *NO* migrates first, followed by *NI*, *C*, and *M*.

Figure 2 shows that, when migrating the four VMs using pre-copy, the migration order *NO/M/C/NI* has the lowest TMT, which means that the network bandwidth affects the TMT of the memory-intensive VM most. The TMT of *NO/C/M/NI* is slightly higher than that of *NO/M/C/NI*, which means that the migration order of CPU-intensive and memory-intensive VMs is relatively less important when enough network bandwidth is available. Our experimental results also show that migrating a non-memory intensive VM requires much less network bandwidth than migrating a memory-intensive VM and hence the TMT of *C/NO/M/NI* and *NO/M/NI/C* is slightly higher than that of *NO/M/C/NI*. When *M* is migrated after *NI* or before *NO*, the TMT is high. Order *NI/C/M/NO* has the highest TMT (about 2.5X higher than *NO/M/C/NI*) followed by *NI/M/C/NO*.

Figure 3 shows that the migration order also affects the TMT of post-copy. Orders *NO/M/C/NI* and *NO/C/M/NI* have the lowest TMT (about $53s$), because migrating *NO* first increases the network bandwidth available for migrating other VMs and hence reduces the TMT of other VMs. The figure also shows that the order of *C* and *M* does not affect the TMT. This is because migrating *C* or *M* does not increase the network bandwidth and the TMT of post-copy depends only on the number of non-zero pages and the network bandwidth, but not the dirty page rate. Similar to pre-copy, order *NI/C/M/NO* has the highest TMT ($116.39s$).

Downtime: The downtime of migrating four VMs is $17s - 44s$ for pre-copy and $14ms - 19ms$ for post-copy. The high downtime in pre-copy is due to the migration of the memory-intensive VM *M*. In pre-copy, when *M* is suspended, almost the entire working set of the VM is transferred. In post-copy, the downtime is the time for transferring VM's execution state, which is often small. Our experimental results show that the migration order *NO/M/C/NI* has the lowest downtime.

B. Migration Scheduling Algorithm

This section presents algorithms for managing VMs' migration order based on the VMs' workloads to reduce the TMT. The resource usage of each VM was captured as follows.

CPU usage: The CPU usage of each VM was captured using the *PS* Linux utility from the underlying source host. We captured the CPU usage of all processes running inside a VM for a period of time and then computed the average.

Memory usage: The number of non-zero pages of a VM is computed by subtracting the number of zero pages (computed using the is_zero_range function in QEMU) from the VM's total number of pages. In addition, we utilize the dirty page tracking mechanism in KVM/QEMU to compute the unique page dirty rate prior to migration.

Network usage: The network usage was measured by capturing the total data packets received and transmitted over the tap interface of each connected VM using the *ifconfig* utility.

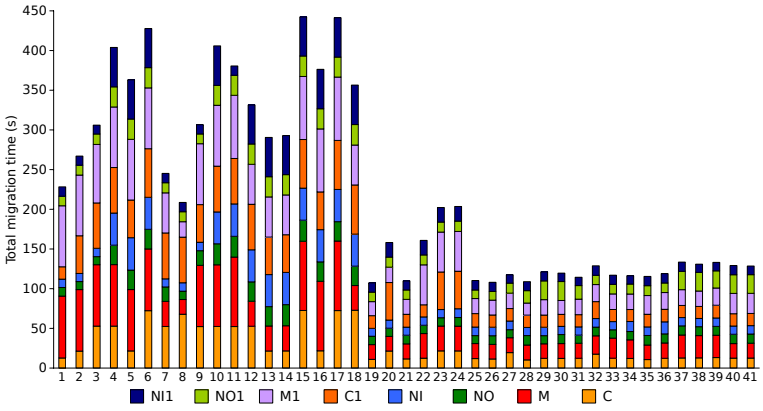
Figures 2 and 3 consider only one VM in each resource-intensive category. In cloud environment, multiple VMs running on the same host may belong to the same resource intensive category. It is also possible that each VM running on a host is not resource intensive, but all VMs together are resource intensive. In addition, a VM can be both memory and CPU intensive, both network and memory intensive, or both incoming and outgoing network intensive. Below, we present algorithms for managing the migration order to reduce the TMT in post-copy and pre-copy.

C. Post-copy Migration Scheduling

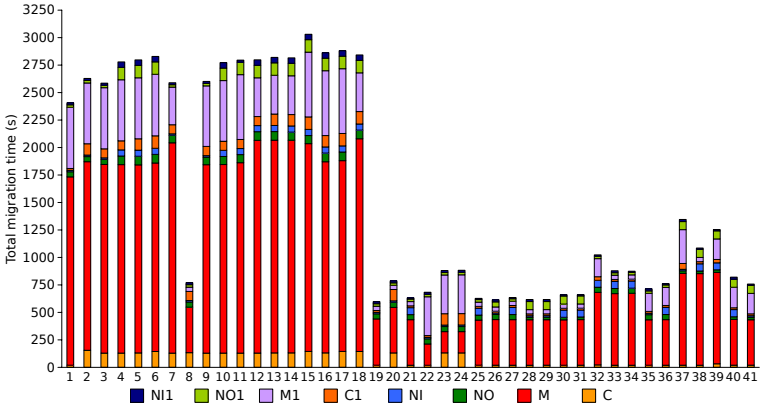
As described in Section IV-A, the TMT of post-copy can be estimated as $\frac{N}{B} + down$, where N is the size of the VM's non-zero pages, B is the migration bandwidth, and $down$ is the post-copy downtime (which is usually very low). Intuitively, the TMT could be reduced by first migrating VMs with fewer non-zero pages and whose migration may increase the bandwidth available for migrating the rest of the VMs. Based on the above equation and our experimental results in Figure 3, we propose a heuristic order-aware migration algorithm to reduce the TMT in post-copy.

The algorithm first migrates VMs whose outgoing bandwidth is higher than the incoming bandwidth in ascending order of $\frac{N}{outgoing-incoming}$, where $outgoing$ and $incoming$ represent the outgoing and incoming network usage of the VM, respectively. This is because, the lower the N, the less time it takes to migrate the VM, and the higher the $outgoing - incoming$, the more outgoing network bandwidth is released after the migration. Next, the algorithm migrates VMs that have about the same incoming and outgoing network usage, which does not affect the available bandwidth. Finally, the VMs whose incoming bandwidth is higher than the outgoing bandwidth are migrated in descending order of $\frac{N}{incoming-outgoing}$. Such VMs are migrated at the end as their new placement would increase the incoming bandwidth usage on the destination and hence may reduce the bandwidth available for migrating the rest of the VMs. In addition, VMs with higher non-zero pages should be migrated when the available bandwidth is higher to reduce the TMT.

We conducted experiments to evaluate our VM scheduling algorithm. Our test environment is the same as that described in Section IV-A. We migrated 8 VMs in which at least two VMs belong to the same resource intensive category with different working set size or network usage. The top table of Figure 4 gives the benchmarks used in our experiment and their resource intensive categories. C memory-intensive, kernbench, iPerf outgoing-intensive, and iPerf incoming-intensive are the



(a)



(b)

Abbr.	Workload description
C	Kernbench (CPU-intensive)
C1	perlbench (CPU-intensive)
M	C memory-intensive ($1.8GB$ working set)
M1	mcf_r (memory-intensive, $0.6GB$ working set)
NO	iPerf outgoing network-intensive
NO1	httperf outgoing network-intensive
NI	iPerf incoming network-intensive
NI1	httperf incoming network-intensive

#	Migration order	#	Migration order
1	M/M1/NO/NO1/C/C1/NI/NI1	22	NO/NO1/C/C1/NI/NI1/M/M1
2	M/M1/NO/NO1/NI/NI1/C/C1	23	NO/NO1/NI/NI1/M/M1/C/C1
3	M/M1/C/C1/NO/NO1/NI/NI1	24	NO/NO1/NI/NI1/C/C1/M/M1
4	M/M1/C/C1/NI/NI1/NO/NO1	25	NO/NO1/M/M1/C/C1/NI1/NI
5	M/M1/NI/NI1/NO/NO1/C/C1	26	NO/NO1/M1/M/C/C1/NI/NI1
6	M/M1/NI/NI1/C/C1/NO/NO1	27	NO/NO1/M1/M/C/C1/NI1/NI
7	C/C1/NO/NO1/NI/NI1/M/M1	28	NO1/NO/M/M1/C/C1/NI/NI1
8	C/C1/NO/NO1/M/M1/NI/NI1	29	NO1/NO/M1/M/C/C1/NI/NI1
9	C/C1/M/M1/NO/NO1/NI/NI1	30	NO1/NO/M1/M/C/C1/NI1/NI
10	C/C1/M/M1/NI/NI1/NO/NO1	31	NO1/NO/M/M1/C/C1/NI1/NI
11	C/C1/NI/NI1/M/M1/NO/NO1	32	NO/M/M1/C/C1/NO1/NI1/NI
12	C/C1/NI/NI1/NO/NO1/M/M1	33	NO/M/NO1/M1/C/C1/NI/NI1
13	NI/NI1/NO/NO1/M/M1/C/C1	34	NO/M/NO1/M1/C/C1/NI1/NI
14	NI/NI1/NO/NO1/C/C1/M/M1	35	NO/M1/NO1/M/C/C1/NI/NI1
15	NI/NI1/M/M1/C/C1/NO/NO1	36	NO/M1/NO1/M/C/C1/NI1/NI
16	NI/NI1/M/M1/NO/NO1/C/C1	37	NO1/M/NO/M1/C/C1/NI/NI1
17	NI/NI1/C/C1/M/M1/NO/NO1	38	NO1/M/NO/M1/C/C1/NI1/NI
18	NI/NI1/C/C1/NO/NO1/M/M1	39	NO1/M/M1/C/C1/NO/NI1/NI
19	NO/NO1/M/M1/C/C1/NI/NI1	40	NO1/M1/NO/M/C/C1/NI/NI1
20	NO/NO1/M/M1/NI/NI1/C/C1	41	NO1/M1/NO/M/C/C1/NI1/NI
21	NO/NO1/C/C1/M/M1/NI1/NI		

Fig. 4: Total migration time for migrating 8 VMs with 24 different migration orders for (a) post-copy and (b) pre-copy.

benchmarks used in Figures 2 and 3. mcf_r is a memory-intensive benchmark in SPEC CPU 2017 [29], which has a similar page dirty rate as the C memory-intensive program, but with a smaller working set size ($0.6GB$). perlbench_r is CPU-intensive benchmark in SPEC CPU 2017 [29]. httperf [30] is a benchmark for measuring the performance of Apache server (version 2.4.18) in terms of outgoing and incoming network traffic. When running separately on the source machine, httperf outgoing-intensive (*NO1*) uses 43% of the outgoing network bandwidth and httperf incoming-intensive (*NI1*) uses 78% of the incoming network bandwidth.

When all the VMs are running on the source host at the same time, the outgoing network usage of *NO*, *NO1*, *NI*, and *NI1* is 67.4%, 26.3%, 0.18%, and 1.8%, respectively, and the incoming network usage of *NO*, *NO1*, *NI*, and *NI1* is 0.28%, 0.64%, 14.52%, and 78%. The network usage of *M*, *M1*, *C*, and *C1* is about 0. Table I gives the number of non-zero pages of all VMs. Figure 4(a) gives the time for migrating 8 VMs using post-copy. The x-axis specifies the migration order. In our experiment, we selected 41 orders out of $8!(40320)$ possible orders, which are shown in the bottom table of Figure 4. The orders were selected as follows. First, we measured the time taken for migrating VMs that are in the same resource intensiveness category together (orders 1-24 in Figure 4). Similar to our 4-VM experiment in Figure 3, the migration time is the lowest when the outgoing network-intensive VMs are migrated first and the incoming network-intensive VMs are migrated last. Based on the above results, we chose another 17 orders that may have the lowest migration time, in which at least one outgoing network-intensive VM is migrated first and at least one incoming network-intensive VM is migrated last.

Observe from the figure that, the TMT of order 19, 21, 26, and 28 are the lowest (about $108s$). Our scheduling algorithm chooses order 19 or 26, because *NO* has lower $\frac{N}{outgoing-incoming}$ than *NO1* and *NI* has higher $\frac{N}{incoming-outgoing}$ than *NI1*.

D. Pre-copy Migration Scheduling

As shown in equation (2), the TMT of pre-copy depends on the size of the VM's non-zero pages N, the migration bandwidth B, and the unique page dirty rate S. The lower the N and S and the higher the B, the lower the TMT. This section presents a heuristic algorithm to manage the migration order of pre-copy based on the resource usage of VMs. The algorithm is similar to the post-copy migration scheduling algorithm except that pre-copy considers the unique page dirty rate, while post-copy does not.

The algorithm first migrates the VMs that have higher outgoing network usage than incoming network usage in ascending order of $\frac{N}{outgoing-incoming}$, with VMs having low

unique page dirty rate migrated first. Such VMs are chosen first because migrating them would increase the network bandwidth available for migrating the rest of the VMs and VMs having lower unique page dirty rate need lower network bandwidth during the migration. Next, the algorithm migrates VMs that have about the same outgoing and incoming network usage. Finally, VMs with higher incoming network usage than outgoing network usage are migrated in descending order of $\frac{N}{incoming-outgoing}$ with VMs having high unique page dirty rate migrated first. This is because the new placement of such VMs may reduce the available migration bandwidth and VMs having higher unique page dirty rate need higher network bandwidth during the migration.

Figure 4(b) shows that when migrating the 8 VMs using pre-copy, order 19 has the lowest total migration time ($597s$) followed by orders 26 ($614s$), 29 ($617s$), and 28 ($617s$). Our scheduling algorithm chooses order 19 or 26, because *NO* and *NO1* are outgoing network-intensive whose unique page dirty rate is low, *NO* has lower $\frac{N}{outgoing-incoming}$ than *NI* prior to the migration, and *NI* has higher $\frac{N}{incoming-outgoing}$ then *NI1* just before *NI* and *NI1* are migrated. When one or both memory-intensive VMs are migrated between two outgoing network-intensive VMs (orders 32–41), the TMT is higher than migrating all outgoing network intensive VMs before memory-intensive VMs.

E. Discussion

Migrating all VMs simultaneously does not always reduce the TMT: Our experimental results show that the TMT of migrating the four VMs in Figure 2 simultaneously using pre-copy is 1.55x higher than migrating the 4 VMs sequentially using order *NO/M/C/NI* due to the increase in the TMT of *M*. When migrating *NO* and *C* simultaneously, followed by *M*, and then *NI*, the TMT is slightly higher than *NO/M/C/NI*. Similar observation has been presented in [31]. Studying the impact of the migration order on parallel migration is part of our future work.

V. Migration Bandwidth Reservation

There can be a contention of network resources between the migration process and the VMs running on the same host. If the source has heavy outgoing network traffic or the destination has heavy incoming network traffic, then the bandwidth available for the migration process may be low and hence may significantly increase the TMT. This section presents a software-defined networking (SDN) based bandwidth reservation algorithm, which reserves bandwidth for the migration process and dynamically adjusts the bandwidth reserved during the migration based on the available bandwidth.

Our bandwidth reservation was implemented using the QoS traffic shaping feature of Open VSwitch [21], which provides an interface to add network rules and policies dynamically during VM migration. System administrators can use network policies to change the network bandwidth dynamically for a single or multiple interfaces. We reserve bandwidth for the migration process by configuring the ingress traffic shaping. For example, in order to reserve 75% outgoing bandwidth for the migration process, we just need to distribute the remainder of 75% (i.e. 25%), to applications running inside the VMs. In addition, we also reserve higher outgoing bandwidth for outgoing network-intensive VMs than other VMs.

A. Automatic Bandwidth Reservation Algorithm

This section presents an algorithm for automatically reserving the network bandwidth for the migration process on the source and the destination.

a) Bandwidth reservation on the source: Algorithm 1 gives the pseudocode for reserving the outgoing network bandwidth for the migration process on the source host. The initial bandwidth reserved for the migration process is determined based on the available outgoing bandwidth on the source prior to the migration and how urgent the VMs need to be migrated. It is called "initial" because the bandwidth reserved will be adjusted during the migration. We use the term *urgency level* (argument *urgency* in Algorithm 1), to specify how quickly the VMs need to be migrated. The urgency level is provided by system administrators, which specifies the percentage of network bandwidth that should be reserved for the migration process. Upon impending failure of the source host, the VMs on the source host should be migrated as soon as possible, and hence the urgency level should be high. When the source host is taken down for routine server maintenance, the urgency level should be low in order to minimize the performance impact on applications running inside the VM. The algorithm compares the available bandwidth and the urgency level, and picks the larger number as the initial bandwidth (lines 2–4).

The reserved bandwidth needs to be dynamically adjusted during the migration due to the following reasons. First, the available bandwidth may change during the migration either because a network-intensive VM has completed its migration or because the network usage of some VMs changes during migration. Secondly, the network bandwidth needed for migrating different VMs may be different. For example, migrating memory-intensive VMs requires higher bandwidth than migrating CPU-intensive VMs. Our bandwidth reservation process periodically computes the available bandwidth on the source host during migration. If the available bandwidth is higher than a threshold defined by the system administrator (argument *threshold*), then the reserved bandwidth is increased so that the migration process can use the available bandwidth (Lines 9 - 10). Our bandwidth reservation process also periodically computes the network bandwidth used by the migration process. If the bandwidth used by the migration process is close to the reserved bandwidth (which means that all reserved bandwidth is used) and the urgency level is higher than the reserved bandwidth, then the reserved bandwidth is assigned the urgency level (lines 11 - 12). Otherwise, if too much bandwidth is reserved for the migration process, then the reserved bandwidth is assigned the current migration bandwidth so that applications running inside the VMs can use the released bandwidth (lines 13-14).

Algorithm 1 Bandwidth Reservation Algorithm

procedure reserve(*threshold*, *urgency*)
avail_band = (1 - $\frac{outgoing\_bandwidth\_usage}{total\_outgoing\_bandwidth}$) $*$ 100%;
//initial bandwidth reservation
reserve_band = max(*avail_band*, *urgency*);
//dynamically adjust reserved bandwidth during migration
while(migration is not completed)
    avail_band = (1- $\frac{outgoing\_bandwidth\_usage}{total\ outgoing\ bandwidth}$) $*$ 100%;
    mig_band = $\frac{current\_migration\_bandwidth}{total\_outgoing\_bandwidth}$ $*$ 100%;
    if(*avail_band* $>$ *threshold*)
        reserve_band = *reserve_band* + *avail_band*;
    else if($reserve\_band - mig\_band) < 1\%$) and $urgency > reserve\_band$)
        reserve_band = *urgency*;
        else if ($reserve\_band - mig\_band > threshold$)
            $reserve\_band = mig\_band$;
        endif
    endif
    sleep for 1 second;
end while

b) Bandwidth reservation on the destination: The network bandwidth available to the migration process depends on both the available outgoing bandwidth on the source and the available incoming bandwidth on the destination. As a result, our bandwidth reservation process tries to reserve the same bandwidth on the source and on the destination, if possible. However, if the destination host has very high incoming network usage, it may be infeasible to reserve the same bandwidth on the source and on the destination, in order not to significantly degrade the performance of network-intensive applications running inside the destination VMs. The scatter gather mechanism proposed in the next section is used to handle the above situation.

B. Distributed In-memory Scatter Gathering

This section presents a Redis and SDN-based scatter gather mechanism (ReS-based scatter gather), to handle the situation where the destination has lower incoming bandwidth than the outgoing bandwidth of the source. We use a metric called *eviction time* introduced in [32] to represent how quickly the source host can be taken offline or free resource that can be re-purposed for other VMs.

Our ReS-based scatter gather was implemented on top of the vanilla post-copy implementation on KVM/QEMU version 2.5.0. The basic idea is to scatter the memory pages from the source to one or more intermediaries at its maximum speed while the destination gathers memory pages from the intermediaries at its own pace. ReS-based scatter gather enables the source host to quickly evict VMs upon impending failure. The intermediaries can be any network middle-boxes that can work together to accept pages at the same or higher bandwidth as the transferring rate of the source, and have sufficient memory and CPU resources to store and transfer memory pages. The selection of intermediaries depends on what the migration is used for. For instance, if the migration is used to rapidly deprovision an entire rack of machines, then the intermediaries should be located at the destination rack so that the source can be deprovisioned quickly.

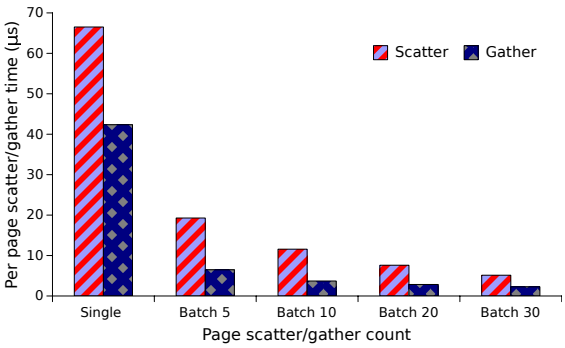


Fig. 5: Page scatter/gather time with single and batch retrieval

In our ReS-based scatter gather, the source host first sends the VM's CPU and I/O state to the destination where the VM is resumed immediately, and then actively pushes the rest of pages to the intermediaries. To ensure that the destination host reads each page from an intermediate node only after the page has been written to the node, we installed Redis, an in-memory distributed key-value store, on each intermediate node to store the memory pages transferred from the source to the node. With Redis, VM's memory content can be staged without having to deal with individual connections with multiple hosts. The source host (i.e. the Redis client) writes VM's memory pages to Redis as key-value pairs. To uniquely identify each page, the page address is used as the key. When the source writes a memory page to Redis, it also sends the corresponding control information directly to the migration manager on the destination over a TCP connection. The control information consists of the address of the corresponding page and the optimizations applied (e.g. compression). This information is used by the migration manager at the destination to gather VMs' memory pages from Redis. When migrating multiple VMs using our ReS-based scatter gather mechanism, the keys (addresses) of memory pages in different VMs may be the same. To avoid key clashes, we create one Redis store for each VM. Once a VM's entire memory is evicted to Redis, the VM can be deprovisioned at the source.

When a page fault occurs at the destination host, the destination sends a page request to the source and the source then sends the corresponding page to an intermediate node. Each faulted request is transferred by the destination over independent threads. If the VM at the destination faults on a page that has already been written to Redis or the VM has been deprovisioned at the source, then the faulted page is retrieved from Redis.

Optimizations: Our experimental results show significant delay when transferring pages one by one to Redis via the synchronous write operation. To address this issue, we store the memory pages locally in a buffer until the buffer reaches to a user-defined batch size. The source machine then transfers all pages in the buffer to the intermediaries as a single batch write operation. Figure 5 shows that the batched transferring has over 20x speedup for batch size 10. Although increasing

the batch size reduces the average time taken to transfer each VM's memory page, the batch size should not be too large. Otherwise, the pages are held too long, which may in turn trigger more remote page faults on the destination, and hence may increase the TMT. Similarly, fetching pages in batches helps reduce the average time spent on retrieving each page.

C. Evaluation

This section presents the experimental results of our bandwidth reservation and ReS-based scatter gather techniques. Our experimental environment is the same as that described in Section IV-A. We obtained the performance results of migrating four VMs in Figures 2 and 3.

Bandwidth reservation: As our bandwidth reservation algorithm does not change the best migration order for the same machine and network configuration, this section presents only the experimental results of our bandwidth reservation mechanism for the best migration order *NO/M/C/NI*. We measured the bandwidth reserved during migration, the TMT, the downtime, and the application performance.

Figure 6(a) gives the bandwidth reserved using our bandwidth reservation algorithm when migrating four VMs using *NO/M/C/NI*. The x-axis specifies the number of VMs running on the source host, which is 4 prior to the migration. We migrated the four VMs using urgency level 95%, 75%, and 55%. Prior to the migration, the available bandwidth is less than the urgency level because of the outgoing network-intensive VM running on the source. As a result, the initial bandwidth reserved is the same as the urgency level. After migrating the outgoing network-intensive VM, the available bandwidth went up to almost 100%, and hence the reserved bandwidth was changed to the available bandwidth. As a result, when the urgency-level decreases from 95% to 55%, the TMT increases only slightly in both pre-copy and post-copy (Figures 6(b) and 6(c)), and the increase is due to the difference in the TMT of the outgoing network-intensive VM. In addition, the TMT with bandwidth reservation is about 13% lower than that without bandwidth reservation in pre-copy, and is about 7% lower in post-copy for all urgency levels.

When the urgency level decreases from 95% to 55%, the downtime of pre-copy increases from $14s$ to $16s$ and the downtime of post-copy increases from $10ms$ to $15ms$.

We have also measured the impact of our bandwidth reservation algorithm on the performance of Kernbench and the C memory-intensive application. Our experimental results show that there is no obvious degradation on the kernel compilation time and the performance of C memory-intensive application with and without bandwidth reservation.

Our bandwidth reservation vs. fixed bandwidth reservation: Figures 7(a) and 7(b) give the TMT of *NO/M/C/NI* in pre-copy and post-copy, respectively, when fixed bandwidth (55%, 75%, and 95%) is reserved for the migration process. The figure show that when the reserved bandwidth is fixed at 75% and 55%, the TMT is significantly higher than that without bandwidth reservation and that with our bandwidth reservation algorithm.

ReS-based scatter gather: Figure 8 compares the eviction time of our ReS-based scatter gather migration and the post-copy migration for order *NO/M/C/NI*. We conducted experiments on $1-3$ intermediaries.

When one intermediate node is used during the migration, we reserved 900Mbits/s outgoing bandwidth on the source and 900Mbits/s incoming bandwidth on the intermediate node, and vary the incoming bandwidth reserved on the destination from 500Mbits/s to 900Mbits/s. Our experimental results show that, the eviction time of ReS-based scatter gather is better than post-copy when the incoming bandwidth of the destination is 500Mbits/s and 600Mbits/s, and is worse otherwise.

When two intermediaries are used during the VM migration, we reserved 900Mbits/s outgoing bandwidth on the source and 450Mbits/s incoming bandwidth on each of the intermediaries so that the bandwidth reserved on the two intermediaries together is the same as the bandwidth reserved on the source, and vary the bandwidth on the destination from 500Mbits/s to 900Mbits/s. Our experimental results show that, the eviction time of our ReS-based scatter gather with two intermediaries is $12\%-43\%$ lower than that of post-copy, and $32\%-36.6\%$ lower than that with one intermediary.

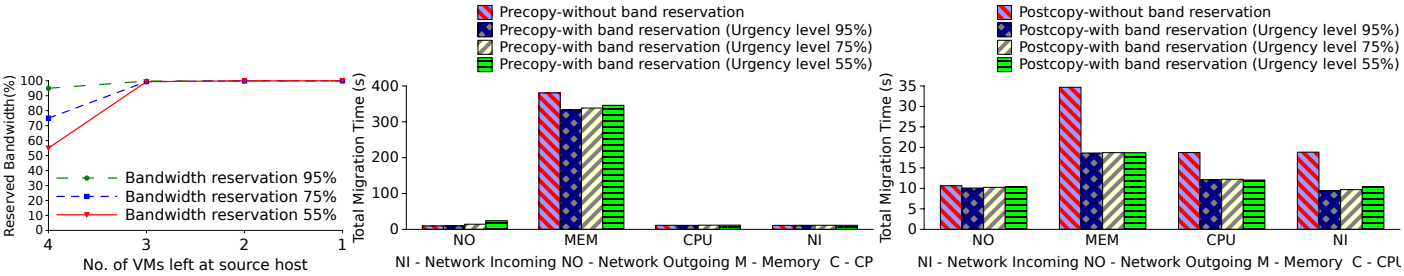
Figure 8 also shows that, when the number of intermediaries increases, the eviction time is not always better if not enough bandwidth is reserved on the intermediaries. For example, when reserving 300Mbits/s incoming bandwidth for each of the three intermediaries, which is $1/3$ of the outgoing bandwidth reserved on the source host, the eviction time is better than post-copy when the incoming bandwidth of the destination is 500Mbits/s and 600Mbits/s, and is worse otherwise. The eviction time is also worse than that with 2 intermediaries. When the bandwidth reserved on each intermediate node increases to 450Mbits/s, the eviction time is better than that with postcopy and with 2 intermediaries.

VI. Related Work

A number of optimizations were proposed to reduce the TMT and the network traffic of pre-copy and post-copy, including deduplication [12], [32], [33], compression [12], ballooning [11], [34], and enlightenment [35]. Techniques have been developed to quickly evict VMs from the source host upon imminent failure of the source [36], [37] and to reduce the time for migrating multiple VMs [12], [18], [38]–[41]. A comprehensive study on selection of optimizations for pre-copy is given in [42]. None of the above approaches considered the migration order or used SDN to reserve bandwidth. The above optimizations can also be used in SOLive to further reduce the total migration time and downtime.

A number of researchers proposed VM placement techniques to find the optimal placement candidates [43]–[46] for migrating VMs. These techniques were used to identify optimal destination candidates prior to VM migration, instead of reducing the TMT during the migration.

vHaul [31] is used to migrate multiple VMs running multi-tier web applications. vHaul uses separate network links for migration traffic and workload traffic. Our work, in contrast,



(a) (b) (c)

Fig. 6: Experimental results for *NO/M/C/NI*: (a) how reserved bandwidth changes after migrating each VM, (b) the TMT of pre-copy for different urgency levels, (c) the TMT of post-copy for different urgency levels.

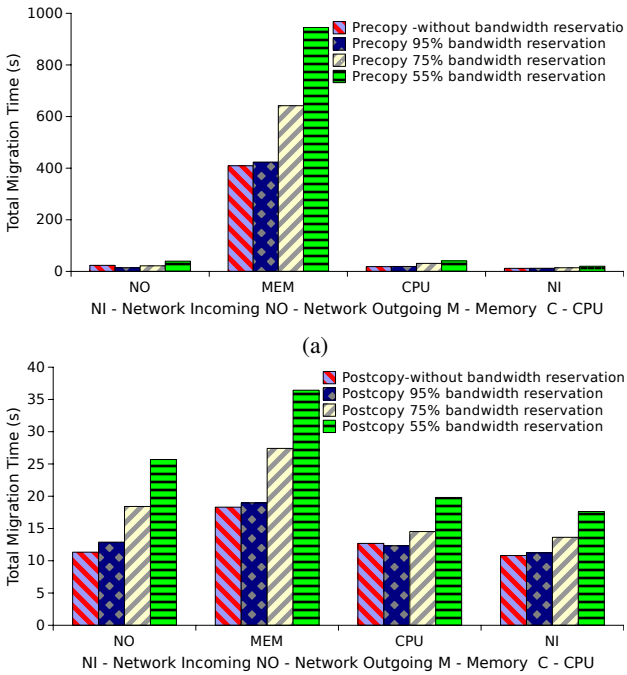


Fig. 7: The TMT with fixed bandwidth reservation using (a) pre-copy and (b) post-copy.

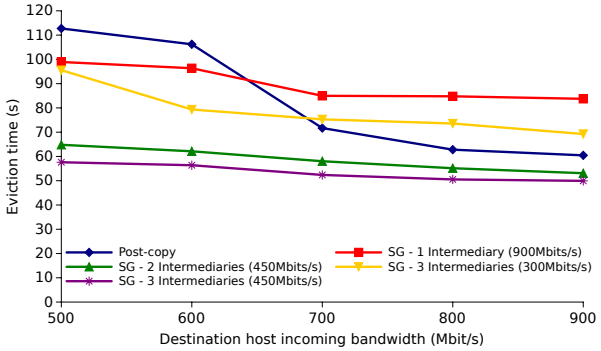


Fig. 8: The eviction time of Post-copy and ReS-based Scatter-Gather migration for order *NO/M/C/NI*.

uses the same network link for the migration and workload traffic. Lu et al. [47] proposed to partition VMs into subgroups based on the traffic affinities among VMs in wide area network and migrate VMs collaborating on a job to the same data center. Their goal is to reduce the inter-cloud traffic and shorten the period of application performance degradation. SOLive , in contrast, aims to reduce the TMT of VM migration in local area network. Sharma et al. [48] developed techniques to migrate nested VMs. Their work neither considered the migration order nor used the SDN to reserve bandwidth.

A heuristic based algorithm was proposed in [39] to mitigate network inconsistencies during VM migration using OpenFlow based networks, which aims to maximize the number of VMs migrated without violating the bandwidth reservation policy. [49] and [50] estimate the required bandwidth for migration based on the memory usage of applications running inside the VM. In [40], [51], the authors proposed to estimate the bandwidth needed to reserve for each iterative pre-copy cycle by dividing the outstanding dirty memory pages over the estimated migration time for a single iterative cycle. However, the above bandwidth estimation approaches were rooted from remedy [49] and can only be applied to the pre-copy migration. Our approach, in contract, can be applied to reserve bandwidth for both pre-copy and post-copy. The above approaches also did not consider the migration order and the situation in which the reserved bandwidth on the destination is less than that on the source. In addition, [51] used simulation based testing environment for bandwidth reservation, and [40] used Linux traffic shaping interface, instead SDN, to reserve bandwidth.

VII. Conclusion

This paper presents SOLive, an SDN-based order-aware live VM migration approach, to reduce the resource contention between the migration process and the VMs being simultaneously migrated. SOLive manages the order in which multiple VMs are migrated and leverages the recent advances in the software-defined networking technology to reduce the total migration time. A prototype implementation of our approach in KVM/QEMU platform shows that SOLive quickly evicts VMs from the source host with low impact on VMs' performance.

Acknowledgement: This work is supported in part by the National Science Foundation under grant OAC-1738929.

References

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

[2] Windows Hyper-V Architecture, https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture.

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

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[5] A. Ruprecht, D. Jones, D. Shiraev, G. Harmon, M. Spivak, M. Krebs, M. Baker-Harvey, and T. Sanderson, "Vm live migration at scale," in *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp. 45–56.

[6] Amazon Inc., "AWS Server Migration Service," https://docs.aws.amazon.com/server-migration-service/latest/userguide/server-migration-ug.pdf.

[7] R. Miller, "Failure rates in google data centers," https://www.datacenterknowledge.com/archives/2008/05/30/failure-rates-in-google-data-centers, 2008.

[8] M. Nelson, B. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *USENIX Annual Technical Conference*, 2005.

[9] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *NSDI*, 2005, pp. 273–286.

[10] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *VEE*, 2009, pp. 51–60.

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

[12] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines," in *High Performance Distributed Computing*, 2010.

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

[14] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. V. D. Merwe, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in *Proc. of Virtual Execution Environments*, 2011.

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

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

[17] P. Svard, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," in *VEE*, 2011.

[18] U. Deshpande, D. Chan, T. Y. Guh, J. Edouard, K. Gopalan, and N. Bila, "Agile live migration of virtual machines," in *IEEE International Parallel and Distributed Processing Symposium*, 2016, pp. 1061–1070.

[19] I. Banerjee, P. Moltmann, K. Tati, and R. Venkatasubramanian, "VMware ESX Memory Resource Management: Swap," in *VMWare Technical Journal*, 2014.

[20] C. Jo, E. Gustafsson, J. Son, and B. Egger, "Efficient live migration of virtual machines using shared storage," in *VEE*, 2013, pp. 41–50.

[21] Linux Foundation, "http://openvswitch.org/."

[22] G. C. P. for Data Center Professionals, https://cloud.google.com/docs/compare/data-centers/, 2017.

[23] S. M. Kerner, "Why facebook does sdn," http://www.enterprisenetworkingplanet.com/datacenter/why-facebook-does-sdn.html, 2014.

[24] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2007, pp. 1–12.

[25] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Computer Communication Revview*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[26] S. Nathan, U. Bellur, and P. Kulkarni, "Towards a comprehensive performance model of virtual machine live migration," in *SOCC*, 2015, pp. 288–301.

[27] C. Kolivas., "Kernbench," http://ck.kolivas.org/apps/kernbench/kernbench-0.50/.

[28] iPerf, http://dast.nlanr.net/Projects/Iperf/.

[29] SPEC, https://www.spec.org/cpu2017/.

[30] D. Mosberger and T. Jin, "Httperf—a tool for measuring web server performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, Dec. 1998.

[31] H. Lu, C. Xu, C. Cheng, R. Kompella, and D. Xu, "vhaul: Towards optimal scheduling of live multi-vm migration for multi-tier applications," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 453–460.

[32] U. Deshpande, Y. You, D. Chan, N. Bila, and K. Gopalan, "Fast server deprovisioning through scatter-gather live migration of virtual machines," in *2014 IEEE 7th International Conference on Cloud Computing*, June 2014, pp. 376–383.

[33] U. Deshpande, B. Wang, S. Haque, M. Hines, and K. Gopalan, "Memx: Virtualization of cluster-wide memory," in *Proc. of International Conference on Parallel Processing*, September 2010.

[34] C. A. Waldspurger, "Memory resource management in VMware ESX server," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, p. 181–194, 2002.

[35] Y. Abe, R. Geambasu, K. Joshi, and M. Satyanarayanan, "Urgent virtual machine eviction with enlightened post-copy," in *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2016, pp. 51–64.

[36] D. Fernando, H. Bagdi, Y. Hu, P. Yang, K. Gopalan, C. Kamhoua, and K. Kwiat, "Quick eviction of virtual machines through proactive live snapshots," in *9th International Conference on Utility and Cloud Computing*, 2016, pp. 99–107.

[37] D. Fernando, J. Terner, P. Yang, and K. Gopalan, "Live migration ate my vm: Recovering a virtual machine after failure of live migration," in *IEEE INFOCOM*, 2019.

[38] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "Cqncr: Optimal vm migration planning in cloud data centers," in *2014 IFIP Networking Conference*, June 2014, pp. 1–9.

[39] S. Ghorbani and M. Caesar, "Walk the line: Consistent network updates with bandwidth guarantees," in *Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 67–72.

[40] H. Liu and B. He, "Vmbuddies: Coordinating live migration of multi-tier applications in cloud environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 1192–1205, 2015.

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

[42] S. Nathan, U. Bellur, and P. Kulkarni, "On selecting the right optimizations for virtual machine migration," in *VEE*, 2016, pp. 37–49.

[43] F. Hermenier, J. Lawall, and G. Muller, "Btrplace: A flexible consolidation manager for highly available applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 5, pp. 273–286, 2013.

[44] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: A consolidation manager for clusters," in *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2009, pp. 41–50.

[45] M. F. Zhani, Q. Zhang, G. Simona, and R. Boutaba, "Vdc planner: Dynamic migration-aware virtual data center embedding for clouds," in *IFIP/IEEE International Symposium on Integrated Network Management*, 2013, pp. 18–25.

[46] S. Al-Haj and E. Al-Shaer, "A formal approach for virtual machine migration planning," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013, pp. 51–58.

[47] T. Lu, M. Stuart, K. Tang, and X. He, "Clique migration: Affinity grouping of virtual machines for inter-cloud live migration," in *International Conference on Networking, Architecture, and Storage*, 2014.

[48] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy, "Tenth european conference on computer systems," 2015, pp. 1–15.

[49] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, "Remedy: Network-aware steady state vm management for data centers," in *International IFIP TC 6 Conference on Networking*, 2012, pp. 190–204.

[50] H. Wang, Y. Li, Y. Zhang, and D. Jin, "Virtual machine migration planning in software-defined networks," *IEEE Transactions on Cloud Computing*, 2017.

[51] V. Mann, A. Vishnoi, A. Iyer, and P. Bhattacharya, "Vmpatrol: Dynamic and automated qos for virtual machine migrations," in *workshop on systems virtualiztion management (svm)*, 2012, pp. 174–178.