Alleviating I/O Interference via Caching and Rate-Controlled Prefetching without Degrading Migration Performance

Morgan Stuart, Tao Lu, Xubin He
Department of Electrical & Computer Engineering
Virginia Commonwealth University
{stuartms, lut2, xhe2}@vcu.edu

Abstract—**The process of migrating a virtual machine and its virtual storage can greatly degrade the performance of other guests and applications running on the same host, including the migrating machine itself. Through experimental evaluation, we investigate the I/O performance degradation imposed by storage migration on co-located machines. We examine naive approaches for mitigating this interference by adjusting host system settings and migration parameters. While effective in some contexts, our analysis demonstrates that performing a migration using these I/O constraining techniques will increase migration latency and limit its ability to converge. Therefore, we present a design and analysis of *Storage Migration Offloading*, a migration method that reduces I/O interference, maintains lower migration latency, and converges under higher dirty rates. Storage Migration Offloading utilizes a buffer store populated during migration using a dynamic cache policy and rate controlled prefetching. Data is transferred to the destination host from both the buffer and primary disks in a way that minimizes interference on the primary disk while attempting to maintain the desired migration speed.**

I. Introduction

The Virtual Machine's (VM) [1][2] decoupling of hardware resources and an OS kernel has rightfully garnered attention from both consumer-oriented enterprises and High Performance Computing (HPC) practitioners. Corporations can scale infrastructure in a public cloud to meet changing demands, without large up-front cost [3], and performance oriented researchers can implement purpose-built operating systems atop a generalized platform [4][5]. Although these pursuits seem disassociated, VM interference remains an intractable property of any virtualized topology [6].

Interference among co-located VMs occurs when the hypervisor cannot satisfy the aggregated requests being issued by its guests. This is typically due to the over-utilization of a particular resource or set of resources, such as storage bandwidth or CPU time. Inevitably, a set of, if not all of, the VMs will be impacted by this, resulting in a period of degraded machine performance. A VM user will experience this low performance, even though their guest machine may not be issuing demanding requests.

The cloud [3][7][8] and HPC [5][9][10] environments both leverage VMs for their flexibility and black-box characteristics. A key enabler of this flexibility for both use-cases is VM live migration [11][12]. Live migration allows an infrastructure administrator to re-locate a VM to another physical host, with no discernible guest downtime. This transfer can either move the entirety of the VM, including storage, [13][14] or simply re-locate the memory and CPU-state, which requires shared storage. Most of the work that expands upon migration focuses on minimizing overall latency and migration downtime, with little or no consideration for interference [15][16].

The process of live migrating a VM levies a considerable amount of interference across co-located VMs, including the migrating machine itself. Research has shown that primary resources, such as CPU and network I/O, undergo significant contention during a live migration [17]. More recent work has begun to address this issue in the form of placement-based avoidance [7] or data-transfer reduction [18]. However, we argue that neither of these approaches directly address the storage interference that occurs during live migration. Storage I/O interference occurs because the migration process must perform at least one full read of the virtual disk in order to complete the transfer. This large and burdensome access takes place alongside the normal operations issued by the migrating VM and the co-located VMs. Furthermore, the I/O issued by cloud VMs has been shown to be both large and persistent [19], making this type of interference even more problematic. The method we propose seeks to directly mitigate storage interference during pre-copy live migration.

In this work, we demonstrate the large storage performance degradation incurred during a VM storage migration for I/O-bound workloads. We explore basic means of remediation by adjusting the static migration bandwidth and reducing the migration's I/O priority. Our results show that I/O performance degradation is unavoidable in current designs without some compromise to the migration's latency and downtime. We believe that the performance of both the co-located VMs and the ongoing live migration must be taken into consideration when approaching low-interference storage migration methodologies.

Thus, we propose a new strategy for low interference live storage migration which we refer to as *Storage Migration Offloading* (SMO). SMO is designed to leverage additional storage to offload a migrating VM's virtual disk during its migration. The migration process can then direct portions of its large sequential read to this smaller secondary store, avoiding storage-level interference. However, transferring the bulk of the virtual disk from the primary storage device to the secondary SMO store must be performed prudently, without inflicting the very interference we aim to eliminate.

We accomplish this by only reading from the primary storage device at a rate that does not cause significant interference. If this rate controlled primary storage read exceeds the desired transfer rate configured by the administrator, the excess is directed to the SMO buffer. When the rate controlled read cannot meet the desired rate without interference, the migration can source non-migrated data from the buffer instead, maintaining migration performance. Preliminary equations for determining these rates are presented in section III-B. The buffer also caches the migrating VM's I/O accesses using a dynamic policy, presented in section III-A, that compliments our rate controlled prefetching. Moreover, by caching writes, we can enable high bandwidth reads of dirtied data by sourcing from the buffer. This enables convergence for dirty rates that would normally be limited by the overburdened primary storage.

Our work has two primary contributions:

1) Tests illustrating the impact storage migration on co-located I/O workloads.
2) The design and analysis of Storage Migration Offloading, a method of migration that reduces interference without undue sacrifice to the migration's performance.

II. Investigating Storage Interference During VM Migration

In order to understand the characteristics of I/O interference during full machine migration, we perform storage migration under various conditions. We begin by investigating the interplay between static migration rate and interference severity. We also show that changing the migration process's I/O priority is ineffective.

The test bed is designed to imitate large-scale datacenter topologies with shared storage. All physical machines in these experiments use an Intel Xeon E5-2630 with 64 GB of RAM. Each are connected via both 1 Gbps Ethernet and Infiniband. The virtual disks are stored on a RAID-6 storage server that is exported over NFS to the test machines. Each host is running CentOS Linux 6.3 with QEMU-KVM version 0.12.1.2 and libvirt 0.9.10. Migrations are performed over the Infiniband link, but NFS traffic is directed over standard 1 Gbps Ethernet. All VMs are configured with 10 GB of raw storage, 2 VCPUs, and 2 GB of RAM.

This test uses a pair of VMs, an application VM and an idle migrating VM, to show the potential for co-located VM performance loss during a live migration. During each test, the application VM executes an *fio* benchmark while the migrating VM simultaneously migrates to a separate physical host. *Fio* is configured to perform random reads and writes across a 1 GB file using the *sync* ioengine at approximately 2 MB/s. During each test, the bandwidth parameter for the migration is adjusted. This effectively controls the I/O rate of the migration process, and thus shows the impact of an increasingly aggressive migration methodology. The results of these tests are presented in figure 1 and the decline in the application VM's disk performance is clearly illustrated. At a configured bandwidth of 32Mbps, the default for QEMU-KVM, we begin to see considerable degradation, with *fio* performance dropping by 76% from the its performance during the 8Mbps migration. When the mgiration is configured to 512Mbps, *fio* running in the colocated VM is operating at less than 5% the performance of its 8Mbps counterpart.

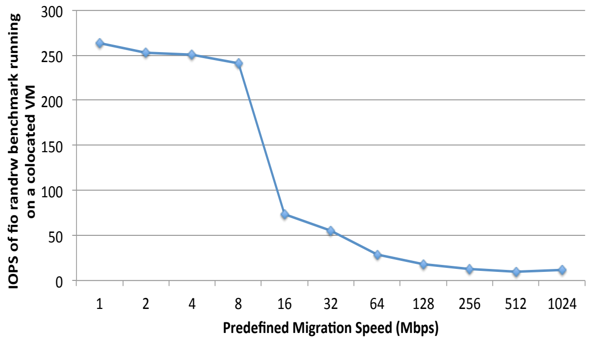

Fig. 1. Migration interference imposed on *fio* benchmark.

For these tests, bandwidth settings at 8Mbps or below do not appear to cause unacceptable I/O interference. Using a low static rate, while effective in reducing disk interference in most scenarios, has several critical drawbacks. First, the migration latency will increase due to the constrained bandwidth. Additionally, this means that any resource interference that is occurring, however small, will be significantly prolonged. Second, the stop-and-copy phase of pre-copy live migration cannot begin until the remaining working set of data at the source is sufficiently small. This working set must now be smaller with the reduced bandwidth in order to achieve a desired downtime. Finally, with low bandwidth allocation, a migration may have difficulty overcoming the migrating VM's dirty rate. If this convergence cannot occur, then the VM's migration will fail.

Therefore, these results clearly show that data-oriented VMs, popular in today's clouds [19], cannot be migrated without interference when only static bandwidth reduction is employed. Instead, the results provide support for a rate controlled migration methodology. During a rate controlled migration, the migration process only issues its reads when utilization of the backing store allows for it. This allows a migration process to respond to I/O workload changes in an effort to maintain low interference or instead to fully exploit an under-utilized disk. However, a rate controlled migration with high primary disk contention will force the rate low, causing convergence issues and high latency. Therefore, rate controlled migration can suffer from the same convergence and downtime problems faced by the static method.

TABLE I
fio IOPS (BLOCK SIZE: 4KB) FOR VARYING MIGRATION PROCESS PRIORITIES AND *fio* I/O PATTERNS

	Migration Priority		
fio I/O Type	best effort:0	best effort:4	idle priority
Rand. read	20	19	20
Rand. write	58	53	49
Rand. read/write	16	15	16
Seq. read	125	126	130
Seq. write	106	104	103

We also perform similar tests examining the effect of the migration's I/O priority levels in the Linux CFQ scheduler. We test both the high (0) and low (4) priority *best effort* schemes, as well as the *idle* priority. This coarse-grained method has little impact, as can be seen in table II, making it unable to effectively control interference.

III. Storage Migration Offloading

One of the most challenging aspects of reducing interference is doing so while still maintaining the expected performance of the migration itself. To this end, we propose Storage Migration Offloading (SMO), an approach to live migration intended to reduce storage interference while still maintaining expected migration latency and downtime. SMO leverages an additional storage device during migration to buffer the relocating VM's virtual disk. With careful combination of buffer policies and rate controlled primary disk reads, SMO can maintain higher migration speeds without incurring the performance degrading interference outlined in section II. The overall architecture of SMO is presented in figure 2.

In this section, we'll first detail the SMO buffer and the caching methodologies that it employs. We'll then discuss the use of a rate controlled prefetching mechanism within the SMO design, followed by its analysis.

A. Caching Data in the Buffer

Fundamental to the design of SMO is recognition of the redundant I/O that is expected to occur during a live migration: the migrating VM's workload will likely access its own storage and the migration procedure will ultimately require at least one full read of this same virtual disk. Therefore, an important component of the SMO design is a caching disk, inserted at the beginning of a migration, that allows both the migration process and migrating guest access to data that has been read or written during the migration period. However, contrary to typical cache objectives, the SMO cache is designed to behave as a bulk data buffer. The SMO buffer's goal is to become populated with the majority of the migrating VM's virtual disk quickly, but without incurring abnormal I/O on the primary backing storage. It follows then, that our application of traditional caching methodologies is not oriented towards general application improvement, but is instead used as a means to leverage workload I/O to transparently offload bulk data from the primary storage.

The notion of *hot* data in cache design is contrary to the well explored method of pre-copy live migration. During pre-copy live migration, the storage and memory are transferred to the destination first, while the VM continues to run on the source machine. Typically, the migration process iterates over these sets, returning to copy any piece of data that may have been *dirtied* during the last round. For this reason, advanced pre-copy migration schemes attempt to move *cold* data earliest in the migration process, leaving the more popular data for last [16]. In doing this, the migration process can avoid unnecessary re-transmissions of dirtied data to the destination host. Therefore, in opposition to decades of cache design, our SMO buffer must be populated with cold data as

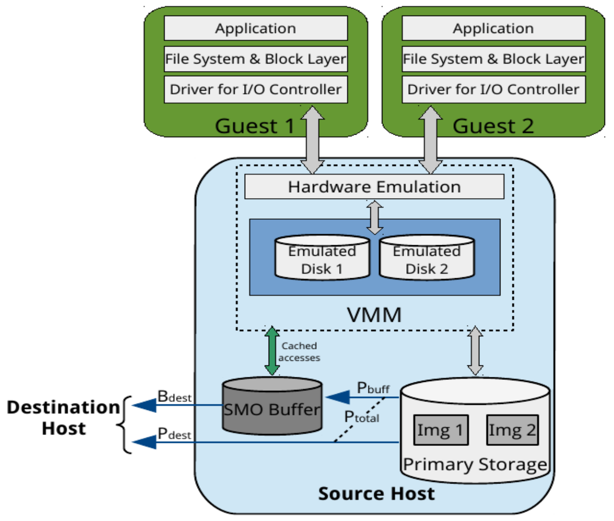

Fig. 2. Architecture of Storage Migration Offloading and a logical view of data movement.

quickly as possible in order to facilitate efficient migration sourced from the buffer. Our current design does not detail any tracking or logging methodology for cold data. For our purposes, cold data simply refers to data that has not yet been naturally cached via the workload's I/O. In general, cold data that is more spatially distant from hot data, data that has been naturally cached, can be prioritized when prefetching data into the buffer.

The additional buffer device is ideally large enough to fit the migrating machine's entire virtual disk. This allows simple and quick direct-mapped caching, meaning no set searches and only one possible line per a set, eliminating evictions. However, to save costs, we design for a buffer smaller than the virtual disk. Due to storage I/O's higher latency, and the need for low conflict rates, we also assume a fully associative policy. Additionally, in order to support consistent write-back modes, the cache must also store sufficient line meta-data.

The design, shown in figure 2, dictates that this disk be attached as any other block device would, such that the hypervisor can access it via the host OS. For instance, the buffer might be connected locally over SATA or mounted via NFS. The buffer interfaces at the hypervisor-level since the machine manager is in complete control of both the migration and the hosted VM's I/O.

The buffer has four combinations of two properties. The buffer may be *fully offloaded* or *partially offloaded* with non-migrated data; that is, the buffer may contain either all data yet to be migrated to the destination host, or the buffer contains only a portion of said data, with some amount still remaining on the primary storage. We consider an empty buffer to be partially offloaded. The buffer may also be either *at capacity* or *under capacity*, referring to whether or not the buffer has space remaining for any additional data. This data must be non-migrated for it to be counted towards the current usage, since we allow migrated data to be evicted in all scenarios.

In each combination of states, we must consider the caching policy of the buffer. We must also develop an algorithm dictating the source of the migration data and the operation of primary disk's rate controlled transfer. It should be understood that the data transferred off of the primary disk has two possible destinations; it can either be prefetched into the buffer or migrated to the destination host machine. Furthermore, data being relocated to the destination host has two possible sources. Data can be sourced from either the SMO buffer or from the rate controlled access to the primary disk. The diagram in figure 2 illustrates this relationship. When data is sourced from the buffer, the cache line is marked as *migrated*. Migrated lines are still valid and can continue to serve reads, but are preferred for eviction in favor of either hot or cold data.

The equations that dictate data transfer speeds are discussed in detail in section III-B. The SMO cache policy during the four possible states is described below.

1) Partially Offloaded & Under Capacity: When the migrating VM issues read operations that hit in the buffer, our design serves the query from the buffer as expected. As a result, we anticipate a natural drop in the primary disks utilization due to these cached accesses. Our rate control mechanism will then leverage this drop in primary disk utilization to increase the primary disk transfer rate.

Writes issued by the VM are only issued to the buffer, making the buffer a write-back cache when in this state. This further decreases the utilization on the backing storage, which in turn allows for a higher transfer rate off the primary storage.

2) Partially Offloaded & At Capacity: In this state, no available cache lines in the buffer exist, so misses must be considered mindfully. Furthermore, the cache policy must favor the enablement of a migration read in order to quickly reclaim space.

Therefore, write misses and all read accesses bypass the buffer, making them bound for the backing storage device. The increased utilization on the backing storage serves two purposes. First, it forces the rate controlled prefetching and transfer algorithm to favor sourcing migration from the buffer. This should increase the speed at which the buffer returns to under capacity. Second, this decreases the utilization that would have been levied on the buffer for evictions or read hits. Still, write hits are allowed to proceed as write-through operations, further increasing utilization while still keeping non-migrated data. Writing through also maintains consistency in support of bypassed reads.

3) Fully Offloaded & Under Capacity: When all non-migrated data resides in the buffer, the cache policy is similar to the policy described in policy 2. However, because the buffer has space remaining, write misses are write-through cached to better support convergence. Again, by writing through the cache, reads can access the same data directly from the backing storage, instead of the buffer.

4) Fully Offloaded & At Capacity: The buffer's cache policy of this state is identical to policy 2.

B. Rate Controlled Prefetching and Transfers

As motivated by our tests, a rate controlled data transfer can be used to read virtual disk data without causing interference during a migration.

In our SMO design, the data from a rate controlled read can be sent either directly to the destination or stored for later transfer in the buffer. Any data sent to the buffer is later used to provide a source for the migration transfer when primary disk utilization restricts the rate control mechanism. The design also ensures that an SMO migration can never perform any worse than a migration employing only rate control. This aspect of the design reduces to several data transfer rates defined below. The logical source and destination for these rates can be seen in figure 2.

First, SMO must track how full the buffer is in order to adjust where data from the primary storage device should be sent. In equation 2, k is a configurable transfer rate that adjusts how eagerly SMO should prioritize sending additional migration data to the buffer. The resulting *Rate Offset*, R, provides a means to trade average migration speed for decreased migration bandwidth fluctuations.

$$F = \frac{Buffer\ in\ use}{Buffer\ size} \tag{1}$$

$$R = k \times (1 - F) \tag{2}$$

where

$$D_0 \geq k \geq 0 \tag{3}$$

Next, in eq. 4 we determine how much of the data being transferred directly from the primary disk, P_{total}, should be transferred to the destination host. The value of P_{total} is the maximum read rate the migration process can issue to the primary storage without causing interference, which is provided by the rate control mechanism previously discussed. Rate P_{dest} will at most be equal to the desired network rate D_0. The *Rate Offset*, R, reduces P_{dest} if the buffer's space is under utilized.

$$P_{dest} = max(min(D_0, P_{total}) - R, 0) \tag{4}$$

Eq. 5 defines P_{buff}, the amount of data read from the primary disk that is bound for the SMO buffer. Functionally, this means that any additional I/O bandwidth beyond rate D_0 or removed by rate R is prefetched to the buffer for use later.

$$P_{buff} = P_{total} - P_{dest} \tag{5}$$

The data being migrated to the destination can also be sourced from the buffer. The rate of B_{dest} in eq. 6 makes up for any bandwidth needed to achieve D_0.

$$B_{dest} = D_0 - min(D_0, P_{total}) \tag{6}$$

Finally, the total rate that data is being sent to the destination host machine is defined in eq. 7.

$$D = P_{dest} + B_{dest} \tag{7}$$

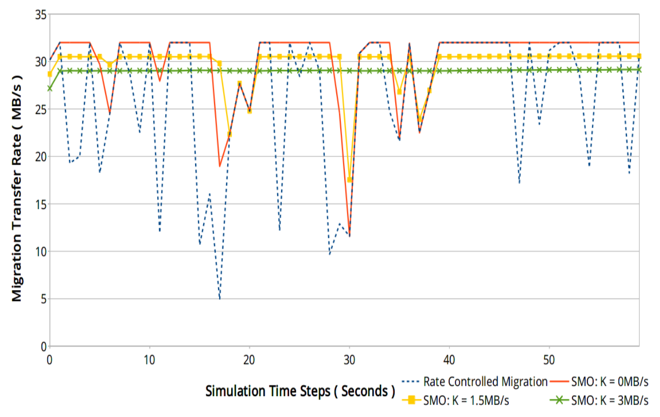

Fig. 3. Comparison of migration strategies (D_0=32 MB/s).

C. *SMO Prefetching Analysis*

We perform an analysis of the rate control methodology with buffering, described in section III-B, to demonstrate the ability of SMO to stabilize and maintain migration speed while still striving to reduce interference. We compare SMO's migration ability with a method that employs only rate control, making $D = min(P_{total}, D_0)$. We generate 250 time steps, each with a P_{total} value randomly calculated between 0 and 64 MB/s. This analysis does not demonstrate the added benefit of the buffer's caching mechanism.

Figure 3 is a plot of each method's achievable destination-bound transfer, D, for the first 60 time steps in the analysis. The desired rate, D_0, is configured to 32 MB/s and SMO's k is varied between 0 MB/s, 1.5 MB/s, and 3 MB/s in order to illustrate its effect. For the full 250 steps, the migration using only rate control achieves an average transfer rate of 26.37 MB/s with a standard deviation of 7.83 MB/s. With no rate offset, meaning k = 0 MB/s, SMO achieves the highest average bandwidth of 31.32 MB/s, with a standard deviation of 2.82 MB/s. However, by increasing the value of k, SMO can drastically reduce fluctuations, with only marginal impact to the average bandwidth. At k=1.5 MB/s, SMO maintains an average 30.38 MB/s with only 1.18 MB/s standard deviation. Increasing k further to 3 MB/s minimizes the deviation to 0.20 MB/s, but the average falls to 29.26 MB/s. Therefore, by including k and the resulting R_O, we create a convenient way to control the migrations volatility characteristics by leveraging the buffer.

These improvements come even without SMO's proposed caching methodology, which would further increase P_{total} and populate the buffer through cached accesses. This design aspect would result in even lower fluctuations and an average rate closer to the desired rate. Furthermore, by caching write accesses, SMO can help a migration process converge even when the primary storage device cannot sustain the required read rate. This additional benefit is also not seen in the results.

In this analysis, both of the proposed methods can reduce interference by adjusting the read rate, but only SMO can maintain the expected migration performance. Therefore, this analysis supports our claim that SMO can avoid interference without sacrificing migration performance.

IV. Related Work

A significant amount of work has studied virtualization and it's impact on resources. The tests performed in [6] examined the effect of different pairs of co-located applications running inside separate VMs, demonstrating the lack of isolation. Q-Clouds [8] implements run-time modifications of VM resources, proposing that resources be reserved in case of unexpected workload increases. The authors of TRACON [20] predict application interference and then assign applications in a way that minimizes expected degradation. Bubble-Up [21] determines susceptibility to interference by examining an application's response to increased memory pressure. This information is then used to place executing jobs on hosts. The authors of FIOS [22] recognize I/O interference across co-located guests and address the issue using a VM I/O cache. Their work targets the general I/O interference among VMs without considering the unique case of migration.

More recent publications have examined the negative effects of migration and possible approaches to their reduction. The work in [17] examines the trade-off between pre-copy phase length and the network performance impact. SonicMigration [18] prunes a VM's memory pages, reducing the transfer size and period of interference. Extensive analysis of VM migration interference, along with co-location interference, was presented by iAware [7]. However, their work did not focus on storage migration and therefore did not include storage interference in their approach. Additionally, their scheme uses interference avoidance, rather than direct mitigation through design alterations, like in SMO.

VMware's vMotion migration uses a *Transmit Buffer* in order to preserve synchrony in their I/O mirroring technique [23]. The necessity for this additional storage is driven by the need to maintain write performance while continuing to duplicate I/O on another host. No specific consideration is given to storage interference reduction, though their transmit buffer would reduce interference during migration of write-heavy workloads.

Storage contention is a long standing issue in computing with many proposed solutions. In Everest [24], write-bursts are directed to underutilized storage devices within the data-center, enabling reads to continue with maintained performance. In this work, the SMO buffer is a generalized storage device. Therefore, a virtual storage device such as those discussed in Everest may provide a cost efficient means to implement SMO's buffer.

V. Conclusion and Future Work

Machine virtualization is quickly reaching every major computing market, just as large-scale data begins to proliferate many of the same industries. Live migration of VMs enables a significant portion of virtualization's touted flexibility, but not without a cost to performance. Full machine live migration can rapidly degrade co-located I/O by more than 95%. In the era of data at scale, this performance loss at the storage level is unacceptable. To combat the issue, we've presented our initial design for Storage Migration Offloading, which aims to reduce interference and maintain migration performance, even during migration of I/O intensive workloads. Our offloading design leverages dynamic cache policies augmented with rate controlled prefetching in order to populate a virtual disk buffer during migration. The migration can then issue reads from this buffer as needed. Our preliminary analysis presented here shows that our SMO buffer can improve the transfer rate by 17% when compared against migration with only rate control.

Although SMO is promising, it has several areas that still need to be considered and fully evaluated. For instance, when the migrating machine issues very little I/O to its backing storage, but is co-located with high I/O guests, SMO will not perform well. In this lamb-among-wolves scenario, the low I/O guest provides no disk usage for SMO to leverage in caching, forcing the rate controlled transfer to a minimum. In theses scenarios, SMO may opt to instead cache the VMs with the highest I/O rates. This is simply a caching scheme to reduce utilization, which has been previously explored [25][22]. There is also potential for a migration *staging* phase. Staging would occur as soon as a guest is targeted for migration but before transfers to the destination begin. Staging would allow an SMO scheme to begin offloading immediately, only to begin the migration transfer once the buffer is sufficiently full.

Acknowledgments

The authors are grateful to the anonymous reviewers for their detailed feedback. This work was supported in part by the U.S. National Science Foundation under Grants CCF-1102624 and CNS-1218960. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agency.

References

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[2] Kivity, Avi. KVM: the Linux Virtual Machine Monitor. In *OLS '07: The 2007 Ottawa Linux Symposium*, pages 225–230, July 2007.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A View of Cloud Computing. *Commun. ACM*, 53(4):50–58, April 2010.

[4] Mark F. Mergen, Volkmar Uhlig, Orran Krieger, and Jimi Xenidis. Virtualization for High-performance Computing. *SIGOPS Oper. Syst. Rev.*, 40(2):8–11, April 2006.

[5] W. Huang, J. Liu, B. Abali, and D. K. Panda. A Case for High Performance Computing with Virtual Machines. In *ICS'06*, Cairns, Australia, June 2006.

[6] Younggyun Koh, R. Knauerhase, P. Brett, M. Bowman, Zhihua Wen, and C. Pu. An Analysis of Performance Interference Effects in Virtual Environments. In *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 200–209, April 2007.

[7] 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. *TRANSACTIONS ON COMPUTERS*, PP, 2013.

[8] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-Clouds: Managing Performance Interference Effects for QoS-aware Clouds. In *EuroSys*, pages 237–250. ACM, 2010.

[9] John R. Lange, Kevin Pedretti, Peter Dinda, Patrick G. Bridges, Chang Bae, Philip Soltero, and Alexander Merritt. Minimal-overhead Virtualization of a Large Scale Supercomputer. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '11, pages 169–180, New York, NY, USA, 2011. ACM.

[10] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Zheng Cui, Lei Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, and R. Brightwell. Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, April 2010.

[11] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Wareld. Live Migration of Virtual Machines. In *NSDI'05*, CA, USA, July 2005.

[12] P. Anedda, S. Leo, S. Manca, M. Gaggero, and G. Zanetti. Suspending, Migrating and Resuming HPC Virtual Clusters. *Future Generation Computer Systems*, 26(8):1063–1072, 2010.

[13] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live Wide-area Migration of Virtual Machines Including Local Persistent State. In *Proceedings of the 3rd International Conference on Virtual Execution Environments*, VEE '07, pages 169–179, New York, NY, USA, 2007. ACM.

[14] Ali Mashtizadeh, Emre Celebi, Tal Garfinkel, and Min Cai. The Design and Evolution of Live Storage Migration in VMware ESX. In *ATC'11*, Portland, USA, June 2011.

[15] Pierre Riteau, Christine Morin, and Thierry Priol. Shrinker: Improving Live Migration of Virtual Clusters over WANs with Distributed Data Deduplication and Content-Based Addressing. In *Euro-Par'11*, Bordeaux, France, August 2011.

[16] Jie Zheng, T. S. Eugene Ng, and Kunwadee Sripanidkulchai. Workload-Aware Live Storage Migration for Clouds. In *VEE'11*, Newport Beach, USA, March 2011.

[17] David Breitgand, Gilad Kutiel, and Danny Raz. Cost-aware Live Migration of Services in the Cloud. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, SYSTOR '10, pages 11:1–11:1, New York, NY, USA, 2010. ACM.

[18] Akane Koto, Hiroshi Yamada, Kei Ohmura, and Kenji Kono. Towards Unobtrusive VM Live Migration for Cloud Computing Platforms. In *Proceedings of the Asia-Pacific Workshop on Systems*, APSYS '12, pages 7:1–7:6, New York, NY, USA, 2012. ACM.

[19] Robert Birke, Mathias Björkqvist, Lydia Y. Chen, Evgenia Smirni, and Ton Engbersen. (Big)Data in a Virtualized World: Volume, Velocity, and Variety in Cloud Datacenters. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, FAST'14, pages 177–189, Berkeley, CA, USA, 2014. USENIX Association.

[20] R.C. Chiang and H.H. Huang. TRACON: Interference-aware Scheduling for Data-intensive Applications in Virtualized Environments. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12, Nov 2011.

[21] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 248–259, New York, NY, USA, 2011. ACM.

[22] Qi Zhang, Hai Jin, Xiaofei Liao, Dingding Li, and Wei Deng. FIOS: A Flexible Virtualized I/O Subsystem to Alleviate Interference Among Virtual Machines. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '12, pages 61:1–61:10, New York, NY, USA, 2012. ACM.

[23] Vmware vsphere vmotion architecture, performance and best practices. White Paper, 2012.

[24] Dushyanth Narayanan, Austin Donnelly, Eno Thereska, Sameh Elnikety, and Antony Rowstron. Everest: Scaling Down Peak Loads Through I/O Off-loading. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 15–28, Berkeley, CA, USA, 2008. USENIX Association.

[25] Tian Luo, Siyuan Ma, Rubao Lee, Xiaodong Zhang, Deng Liu, and Li Zhou. S-CAVE: Effective SSD Caching to Improve Virtual Machine Storage Performance. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, PACT '13, pages 103–112, Piscataway, NJ, USA, 2013. IEEE Press.