Future Generation Computer Systems 91 (2019) 167–176

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

IOFollow: Improving the performance of VM live storage migration with IO following in the cloud

Bo Mao [a,*], Yaodong Yang [b], Suzhen Wu [a], Hong Jiang [c], Kuan-Ching Li [d]

[a] Xiamen University, Xiamen, 361005, China
[b] Microsoft, Redmond, WA, USA
[c] University of Texas at Arlington, Arlington, TX 76019, USA
[d] Providence University, Taiwan

HIGHLIGHTS

- We show IO interference problem between VM IO and migration IO during VM migration.
- We propose IOFollow that schedules the block migration sequence with VM I/O requests.
- We have built a prototype of IOFollow system and validated its efficiency.

ARTICLE INFO

Article history:
Received 17 June 2018
Received in revised form 5 August 2018
Accepted 20 August 2018
Available online 5 September 2018

Keywords:
VM live storage migration
Cloud computing
Resource contention
IO following

ABSTRACT

VM live migration becomes critical to deployment in the Cloud, as to enable fast and transparent workload rescheduling among unevenly utilized physical nodes, aimed at high performance and energy efficient utilization of computer server resources. Nevertheless, VM I/O intensity can significantly impact the performance of VM live storage migration due to contention for the shared disk bandwidth and system resources. Based on this observation, this paper proposes the IOFollow scheme to improve simultaneously the VM performance and migration performance. IOFollow mitigates the I/O interference between VM I/O requests and migration I/O requests, by scheduling the migration sequence according to the access pattern of VM I/O requests. In addition, it improves the cache management efficiency by exploiting the semantic information from the VMs and migration threads. Results obtained from trace-based experiments on a lightweight prototype implementation of IOFollow indicate that IOFollow system is highly competitive, improving both the VM performance and migration performance significantly when compared to existing design.

1. Introduction

Cloud computing has become one of the most important technologies poised to fundamentally change people's lives and IT ecosystems. As one of the most popular products in the cloud market, Virtual Machines (VMs) have been extensively deployed to execute different services and applications for customers, such as web service, mail service, database service and printing service [1, 2]. The underlying hypervisor for the virtualized environment is responsible for the management of physical devices and provision of all sorts of virtual devices to VMs. At the same time, hypervisors are supposed to guarantee the isolation and fairness among different VMs on top of a single shared physical server, whilst improving the overall performance for all VMs with accessibility to physical resources [3–5].

The VM live migration is a built-in module in modern hypervisors, that provide support to the migration of a running live VM from one physical server to another in cloud, either within same or across data centers in different locations worldwide [6,7]. The primary purpose of VM live migration is to meet the increasing need for load balancing and server consolidation, system maintenance and upgrade, VM mobility and manageability in cloud data centers. With the advent and wide deployment of the VM-based cloud computing infrastructure, VM live migration, as an essential functional component of the hypervisors, like ESX [8], XEN [9], QEMU-KVM [10] and HyperV [11], is even more popular technology. Nearly half of large enterprises have deployed hybrid cloud infrastructure in data centers by end of 2017 [12].

In a typical cloud infrastructure, data storage can be either shared or distributed depending on whether the data are stored in a centralized environment, where all servers share the physical storage, or distributed (shared-nothing) environment, and each server has its own dedicated storage [13]. In the shared-storage environment, VM live migration only involves the synchronization

* Corresponding author.
E-mail address: maobo@xmu.edu.cn (B. Mao).

https://doi.org/10.1016/j.future.2018.08.036
0167-739X/

of CPU states, memory states and network interfaces of the target VM between the source and the destination of migration. VM virtual disk images remain in the shared-storage that is accessible from both the source and the destination. With the growing trend of shared-nothing architectures in cloud data centers and the need for VM live migration across different clouds over WAN, it is de facto important to consider VM live migration in a distributed, or shared-nothing storage environment, where VM live storage migration must also migrates the state of VM virtual disk images as well snapshots from the source to the destination. In fact, VM live storage migration has become an integral part of VM live migration in modern hypervisors [14–16]. As the running VM is migrating from one node to another, both the VM's in-memory states and virtual disk images migrate from the source server to the destination server. Given the capacity of the VM virtual storage, which includes the VM's virtual disk images and snapshots, is much larger than that of other VM state information, such as memory state, CPU state and network state, that is crucial to improve the performance of VM live storage migration.

To be specific, any sound and desirable VM live storage migration scheme must possess the following properties:

- **Short Migration Time** [17]: In modern data centers, VMs are running 24/7 basis to serve customers globally. The time window for the system maintenance and upgrade is short, so it is crucial to live migrate VMs quickly. However, the size of a VM's storage image is commonly several to tens of GBs and it may take minutes or even hours to complete a VM live storage migration, which is likely to curtail the capabilities of system management in the cloud data centers.
- **High VM IO Performance** [17]: The IO performance within running VMs must comply with the Service-Level Agreement (SLA) at all times. During the VM live storage migration, applications within the migrating VM are still running and they should be oblivious of the migration process. However, available storage resource is severely stretched due to additional resource hungry migration threads. The cloud service provider should provide the same performance guarantee for all those co-located and concurrent running VMs in the physical server.
- **Capability to Migrate Multiple VMs Simultaneously** [18]: Given the wide deployment of VMs in data centers, it is common to migrate multiple VMs out of a single server, or migrate multiple VMs into one single server. It is much more challenging to achieve acceptable VM IO performance for all running VMs instead of migrating multiple VMs to their destinations at reasonable migration speed.

A number of approaches have been proposed to improve the live storage migration performance [19], including optimization on the data block transmission sequence [6,15] and the migration workflow [14,20], reducing redundant data transmission [21], and leveraging heterogeneous storage devices [22]. After an in-depth examination, we sought that all these approaches fail to address the vital problem of IO interference between the VM IO process and migration IO process, since both types of IO processes share the same critical IO path by reading from/writing to the same shared storage device. Owing to IO resource contention and requests interference between the two different types of IOs, not only will the I/O request queue lengthen in the disk, but the time-consuming disk seek operations will also become more frequent. As result, the performance of the VM IO process will be seriously degraded. Previous experimental results show that the VM IO throughput decreases by a factor of up to 6.42 [23,24].

The order of blocks being migrated, sequential or random, does not impact the performance significantly, since migrating VM's virtual disks cannot be reconstructed until all data blocks are available in the destination server. However, the order of VM I/O requests does affect the performance significantly, since I/O requests are consumed by applications during runtime. For example, if a VM I/O request accesses to a data block located in LBA-A and migrated data block locates in LBA-B, then the disk head will move from LBA-B to LBA-A. If multiple VM I/O streams are interactive with each other, the disk head will move back and forth which degrades both the VM IO performance and the migration performance. Based on these observations, this paper proposes IOFollow system that schedules the block migration sequence in the light of VM I/O requests, so that the time-consuming disk head movements can be reduced significantly. In addition, by selectively caching data blocks from migration threads, incoming VM I/O requests can be served in the memory, thus minimizing the number of memory accesses concurrently. In this way, both VM IO performance and migration performance can be improved significantly.

We highlight the contributions of our study as following:

- From the investigations and the preliminary analysis, we show a vital problem of IO interference between the VM IO process and migration IO process during VM migration in the cloud, since both types of IO processes share the same critical IO path by reading from/writing to the same shared storage device.
- We propose IOFollow system that schedules the block migration sequence in the light of VM I/O requests, so that the time-consuming disk head movements can be reduced significantly. In addition, by selectively caching data blocks from migration threads, incoming VM I/O requests can be served in the memory, thus minimizing the number of memory accesses concurrently.
- We have built a prototype of IOFollow system. Experimental results show that the performance improvement is more compelling in multiple concurrent VM migration scenarios by 55% for migration thread and 45% for VM IO thread, compared to the standard migration schemes.

The rest of this paper is organized as follows. Background and motivation are presented in Section 2, and the design of the IOFollow scheme is depicted in Section 3. The performance evaluation of a lightweight IOFollow prototype is presented in Section 4 and related work is provided in Section 5. We conclude this paper and give the directions for future research in Section 6.

2. Background and motivation

In this section, we provide the necessary background information for the proposed research IOFollow, including sequential IO property and the interference among I/O threads, to motivate this research.

2.1. Sequential IO property

Sequential IO property has been one of the most fundamental concepts in system research area [25], mainly due to the performance disparity between Sequential IOs and Random IOs in storage systems. Over the past few decades, data transfer bandwidth has greatly increased, due to higher density of bits in the surface of a disk drive as well compaction technologies. However, the costs of seek and rotation delay have slowly reduced, since it is much more challenging to speed up the mechanical movements of disk head and the spinning speed of the platters. Therefore, the performance of Sequential IOs is much better than that of Random IOs with frequent disk seeks and rotations [26].

The sequential IO property is determined by many metrics from both spatial and temporal dimensions [25,27–30]:

- **Spatial Dimension:** (1) Consecutive Addresses. The difference between the Logical Block Addresses (LBA) of consecutive I/O requests is within a predefined threshold. It could be further classified as *Strictly consecutive* (no gap between the LBAs of consecutive I/O requests) and *Strided access* (bounded gap between the LBAs of consecutive I/O requests). (2) Consecutive Bytes Accessed. The size of data to read or write in an I/O request on average.
- **Temporal Dimension:** (1) Interleaved Streams. The mixture of I/O requests from multiple threads, applications or VMs. The sequential IO property for individual IO streams may be affected by the interleaving with other IO streams. For instance, two consecutive I/O requests (requests 1 and 2) with consecutive addresses from stream A may experience poor IO performance, as a long disk head movement involved during which the disk serves another I/O request (request 3) from stream B and the LBA of request 3 is far away from the LBA of either request 1 or 2; (2) Inter-arrival Time. The time interval between consecutive I/O requests. Once there is a long waiting time between two I/O requests, some other background I/O requests may be issued in between, which will influence the sequential IO property of the original IO stream.

The interleaving of multiple IO streams will not only affect the Sequential IO property, but also decrease the IO performance significantly for each participating stream. Experiments indicate that a Random Write IO stream is destructive to all other kinds of interleaving IO streams, such as Sequential Read/Write stream [31]. The performance of the Random Write IO stream itself will also be degraded significantly. Other cases of performance degradation for multiple interleaving IO streams are found in [31].

Given the nature of disk access characteristics and sequential IO property of interleaving IO streams, modern file systems, both local file systems [32–34] and distributed file systems [35,36], improve the IO performance by aggressively sending sequential I/O requests to underlying disk drives. For instance, in the log-based file system [32], a sufficient number of updates are buffered in memory before they are sent to disks as a large sequential segment request, so that the disk throughput is improved significantly compared with individual small random write requests. Read requests are served in a similar manner, which will read a complete segment from disks at once. In Google File System (GFS) [36], the minimal size for each request is 64 KB by default, so that high IO throughput in disks can be achieved with sequential I/O requests. Unfortunately, file systems can only affect, but not determine the sequential IO property of IO streams that are produced by user applications themselves. Many optimization techniques have been designed to improve the sequential IO property by leveraging the semantic hints from the applications [37].

2.2. Interference among I/O threads

In a virtualized system, the IO stream at the gate of the storage system is a multi-layer interleaving of individual IO streams: First, an application-level IO stream is produced with the interleaving of multiple thread-level IO streams within the same application. Next, for each running VM, one VM-level IO stream is generated with the interleaving of multiple application-level IO streams as also the VM Operating system IO stream. Finally, the interleaving of all VM-level IO streams and the hypervisor IO stream will become the final IO stream for the storage system. Considering all metrics related to the Sequential IO property, VM-level IO Streams (named as VM IOs) are mostly determined by user applications and guest Operating Systems. Fig. 1 shows the interference among IO threads in a virtualized system during different phases of VM live storage migration.

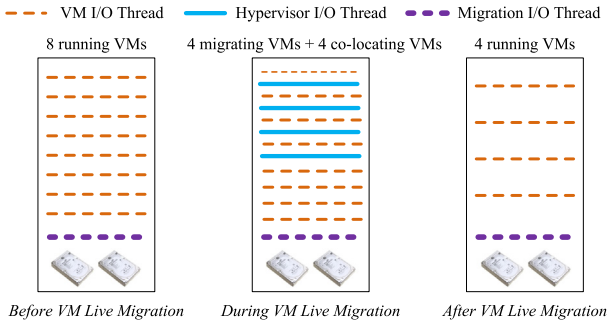


Fig. 1. The interference among IO threads in a virtualized system during different phases of VM live storage migration.

When it comes to the VM live storage migration, one migration thread will be assigned for each migrating VM, and it will migrate all state information of the migrating VM from the source server to the destination server. Most of the time, the migration thread will read data of the VM's virtual disk images from the beginning to the end, as well sync the updated data to the destination server. From the perspective of the Sequential IO property, the migration thread is a perfect sequential workload, as it will read data within the space of the virtual disk images purely sequentially and deserve good performance. Unfortunately, that is not the reality.

It is the interleaving with other IO streams, including VM IOs and other migration IOs, that undermines the sequential IO property of migration threads. Disk head must seek and rotate to other places to serve other I/O requests between two consecutive read requests for the migration thread. After the service to other I/O requests, disk head must seek and rotate back to the neighbor of previous location, to serve the next read request for the migration thread. Therefore, the IO performance of all participating IO streams is degraded significantly, and longer VM migration time and low VM performance cannot be avoided.

During the VM live storage migration, hypervisors are already overloaded because of the additional bandwidth hungry migration threads [38]. By further destroying the Sequential IO property of migration threads, additional weight will be placed on the migration engine, so that is much more challenging to migrate VMs fast and provide SLA for the IO performance of all participating VMs.

2.3. Motivation

Examining carefully the state-of-the-art of related research works and migration workflow internal mechanisms, we have two observations: (1) Individual I/O requests from the migrating VM are generated by applications, so that we have limited capabilities to manipulate the VM IO stream to improve the VM IO performance; (2) Only the total migration time and the accuracy of the VM state transmission matters for migration threads, while the individual request size, the starting address of each request or the sequence of migration I/O requests are not included as priority ones. Therefore, the migration engine has the full flexibility to generate different kinds of migration I/O requests, as long as all the state information of the migrating VM arrives in the destination correctly within a reasonable migration time window.

Inspired by these observations, it is proposed in this paper a novel VM live storage migration scheme, named *IOFollow*, that will improve both the VM IO performance and migration performance by generating and scheduling the migration sequence according to the IO stream of VM requests. Essentially, we will select the next block migration candidate based on two criteria: (1) this data block has not been migrated to the destination; (2) the address of this data block is close to the current access region or the position of

the disk head. In this way, we expect to get rid of unnecessary disk head movements, so that the interleaved I/O requests stream at the gate of the storage system turns sequential. The performance of all VM IO threads, hypervisor IO thread and migration IO threads increases significantly during the VM live storage migration process. Furthermore, we can selectively cache in memory data blocks read by migration threads from the storage system, and utilize these data to serve the predicted incoming VM I/O requests, so that these VM I/O requests will not need to reach the storage system at all. Therefore, both the VM IO performance and migration IO performance can be improved significantly.

3. System design and implementation

In this section, we first outline the architecture overview of the IOFollow system, followed by a description of migration blocks scheduling, IOFollow Block Cache Manager, and finally, issues related to data consistency of IOFollow are discussed.

3.1. Design objectives

The design of IOFollow aims to achieve the following three objectives:

- *Accelerating the VM live storage migration performance:* By removing most of the unnecessary and time consuming disk seek operations, the VM live storage process can be significantly accelerated.
- *Improving the VM IO performance:* By improving the block cache hit ratio and reducing disk seek operations, VM I/O requests can be either served by the block cache, or served by storage device faster due to fewer disk seek operations.
- *Providing high flexibility:* IOFollow is quite flexible and can be tuned with different parameters for different applications, such as cache replacement algorithms and migration chunk sizes.

3.2. IOFollow architecture overview

File-based virtual disk images have been extensively adopted in the virtualized environment [8,39]. From the VM's perspective, the virtual disk image is the same as the physical disk that supports all classes of block layer's API, such as iSCSI commands. Additionally, it is compatible to main stream Guest Operating Systems, such as Windows and Linux. From the storage system's point of view, virtual disk images are viewed as large and regular files that can be stored in most file systems. Therefore, all I/O requests from user applications of the running VM transform the I/O requests for the underlying large file that hold the virtual disk image. Similarly, the migration thread will read this large file to migrate the VM's storage state information. The performance of virtual disk images is crucial for the running VM's performance and the migration agility. To improve the performance of virtual disk images, several dedicated file/storage systems have been designed for virtual disk images only, such as VMWare's VMFS and Tintri VMStore [40,41].

Fig. 2 shows the architecture overview of the IOFollow system within Linux KVM framework [10]. IOFollow is a simple yet portable module that can be incorporated into any modern hypervisors, such as ESX [8] and XEN [9] hypervisors. Moreover, its parameters as migration chunk size and block cache replacement can be tuned to different application workloads. For the VM live storage migration jobs, only the server in the source side needs to incorporate the IOFollow module, while the server in the destination side remains at no changes. IOFollow is a performance boost layer that can be combined with conventional live migration approach, including Dirty Block Tracking and IO Mirroring, to further improve both VM IO performance and migration performance. IOFollow can

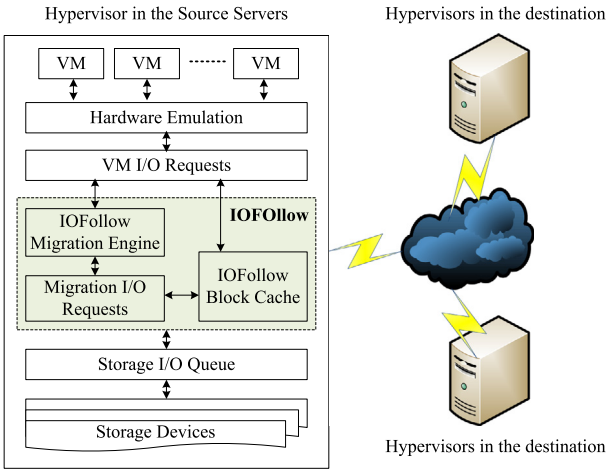


Fig. 2. The architecture of IOFollow System.

be applied to live migrate VMs between servers within the same cluster or across data centers in different locations.

In current virtualized systems, there are two level caches for each running VM: guest disk write cache (within the VM), and host page cache (within the hypervisor). According to the characteristics of different applications, users can select either to enable or disable one of two level caches or both, before the creation of a VM or during the lifecycle of a running VM. During the normal execution period, solely the VM IO stream from the running applications and guest operating system access these two level cache systems, economy that could save storage accesses for applications or guest operating system. When the live storage migration starts, an additional IO hungry stream of I/O requests from migration thread comes in, which will occupy a large portion of dedicated host page cache for the migrating VM. As traditional cache cannot improve performance for streaming I/O requests (e.g. online streaming video applications), improvement designs of cache system are necessary for the overall system performance.

IOFollow contains two major components: Migration Blocks Scheduler and Migration-Aware Block Cache Manager. Migration Blocks Scheduler will analyze the VM I/O requests traffic, identify the current VM access zone, predict the later IO access region, and then select the right data chunk to migrate. Once the data chunk is migrated to the destination server, it will be handed over to the Migration-aware Block Cache Manager. The selection of migration data chunks is based on two parts: 1. shorter seek time of the storage system for the migration I/O request; 2. this data block fetched by the migration thread may serve later VM I/O requests with higher possibility. Migration-aware Block Cache Manager is to manage the memory resource and intelligently cache data blocks for later VM I/O requests. As the migration thread only scan the virtual disk images once, it will not access the same data blocks more than once, except the updated data blocks. Nevertheless, these data blocks may be accessed by VM I/O requests, and therefore, as caching hot migration data blocks in memory, many incoming VM I/O requests can be served by Migration-aware Block Cache Manager in memory directly. Once the block cache is full, a cache replacement algorithm will take actions and cold data blocks will be evicted.

3.3. IOFollow migration blocks scheduling

In the Migration Blocks Scheduling component, there are three decisions to make for the VM live storage migration:

The first decision in which system administrators need to make is how much storage resource should be allocated for the migration

thread. Given the total storage resource remains unchanged, the more resource migration thread gets, the less storage bandwidth VM threads have. Essentially, this is a tradeoff between VM IO performance and migration performance, and it can be adjusted for different user cases or preferences.

As soon the storage resource allocation is determined, Migration Blocks Scheduling will select the next chunk to read from the storage system, whereas such a chunk will be migrated to the destination server. Ideally, we prefer to data chunks satisfying the following conditions: (1) The chunk has not been migrated to the destination server yet, (2) The starting address of this data chunk is close to current VM IO access region, (3) High probabilities that the incoming VM I/O requests will read partial or full of this data chunk. In order to intelligently pick the right data chunk candidate, spatial and temporal locality of the VM IO stream should be analyzed online, and the VM's working set is predicted by the Migration Blocks Scheduling component.

The last decision is the migration chunk size. While both fixed and dynamic chunk size are applicable for IOFollow system, the overall VM and migration performance can be noticeably affected by the sizes of individual migration data chunks. This is an old and common problem in many aspects of system design, which require different techniques, such as online profiling and trace analysis, to tackle this problem. For instance, dynamic chunk size has been applied to reduce the number of repeated data chunk migration [16]. In the proposed IOFollow system, we start from fixed chunk size and evaluate its performance improvement. There is no doubt that IOFollow can be combined with other chunk size determination mechanisms for different workloads. The full algorithm of Migration Blocks Scheduling is presented in Algorithm 1.

Algorithm 1: IOFollow Migration Blocks Scheduling.

```
1:  Initialization:
2:  setMigVM: the set of concurrent migrating VMs
3:  position: the disk head location of the storage system
4:  function IOScheduling(setMigVM)
5:      for VM1 in setMigVM do
6:          Serve I/O requests for VM1
7:          Update position with I/O requests' addresses and sizes
8:          vmrange = the block range of VM1 that needs to migrate
9:          SelectChunk(vmrange, position)
10:     end for
11: end function
12: function SelectChunk(vmrange, position)
13:     if vmrange is not empty then
14:         Select the data chunk from vmrange whose address is
    close to position
15:         Remove this chunk from vmrange
16:         Issue the I/O request for this data chunk
17:         Update the position to the current chunk address
18:     else
19:         Return Complete
20:     end if
21: end function
```

3.4. Migration-aware block cache manager (MABCM)

During the runtime, each running VM is assigned a number of memory pages by the hypervisor, so that the VM can cache information data in memory, rather than access the storage device. Normally, these memory pages are classified as guest write disk cache and host page cache [42], as described in previous section. As it comes to the VM live storage migration, the whole virtual disk images will be read from storage system to memory and then migrated to the destination server. *A question comes next: Do we need to cache these data blocks in memory or not?* It is not possible to cache all data blocks in memory, as the memory allocated to single running VM is much smaller than the size of the virtual disk images. In case we do not cache all data blocks in memory at all, we end up wasting good portion of storage bandwidth. Considering the cost it takes to read in data blocks from storage system to memory and additional read requests will be issued to the storage system by VM thread, even if the target data blocks have already been read in once by the migration thread. Therefore, caching a set of data chunks with higher possibility to be accessed by VM thread in memory will greatly improve the VM IO performance by the reduction of storage accesses. The challenge is how to identify these data chunks and how to evaluate and replace data chunks once the cache is full. As discussed in the previous section, the Migration Blocks Scheduling component takes charge of the first challenge, while the MABCM solves the second one.

In MABCM, each entry is a data chunk of virtual disk image in a specific address, and there are a number of such entries in the MABCM. For each entry, we can evaluate its liveness value based on the answers to following questions: (1) Whether the block has already been migrated to the destination server, (2) The time this block been in the cache, (3) The possibility that this data block will be accessed by VM IO threads in future. Upon with such information, MABCM can sort these entries and replace the entry with the least liveness value as the cache is full. Since such information is closely related to the spatial and temporal locality of workloads, such cache management algorithm has to be tuned to cater different applications. The more accurate locality we learn from workloads, the better cache hit ratio and IO performance we achieve. Compared to the baseline approach, in which traditional two level cache management algorithm is employed, the proposed migration-aware cache management scheme can significantly improve the VM live storage migration performance. The skeleton algorithm of Migration-aware Block Cache Manager is depicted in Algorithm 2.

4. Performance evaluation

In this section, we present the performance evaluation of the IOFollow scheme through extensive trace-driven experiments.

4.1. Evaluation methodology

As discussed before, both VM performance and VM live migration performance are fundamentally important for overall system performance. The VM live migration performance is determined by the network performance and storage performance, whilst the VM performance is mainly determined by the storage system. In the experimental process, we focus on the evaluations of storage performance for both VM threads and migration threads.

Specifically, we create a virtual disk image in the disk drive, and replay the published storage block level traces on top of the virtual disk image. Concurrently, we generate the migration of I/O requests on top of the same virtual disk image, in which all data blocks within this virtual disk image will be read. For both VM IO performance and migration performance, we use average IO response time as performance metrics. Shorter response time for VM I/O requests means higher IO performance for the applications within the running VM. In addition, shorter response time for the migration I/O requests indicate faster VM live storage migration. The proposed IOFollow scheme will be compared to the standard migration scheme that will ignore the characteristics of the VM IO stream and migrate the virtual disk images from the beginning to the end, such as the default VM migration in Linux KVM [10]. The normal state indicates that the system is running without VM migration.

Algorithm 2: Migration-aware Block Cache Manager.

```
1: Initialization:
2: Apply memory pages from the hypervisor.
3: Initialize metadata for the cache manager
4: Start to take requests from VM threads or migration threads
5: function ReadBlock(blockAddr)
6:     if blockAddr is in cache then
7:         Return the block and update liveness info for this block
8:     else
9:         Read data block from storage, return data block to the
   client
10:        Migrate this block to the destination if has not migrated
   yet
11:    end if
12: end function
13: function WriteBlock(blockAddr, dataBuf)
14:    if blockAddr is in cache then
15:        Update data block in memory and the liveness info
16:    else
17:        Read data blocks from storage, and update data in mem-
   ory
18:        Migration data blocks to destination if has not migrated
   yet
19:    end if
20: end function
21: function PutBlock(blockAddr, dataBuf)
22:    if cache is not full then
23:        Put data blocks to the cache and update the liveness info
24:    else
25:        Evict data blocks with least liveness information
26:        Put data blocks to the cache.
27:    end if
28: end function
```

Table 1
The experiment setup.

CPU	Intel(R) Xeon(R) CPU, X3440@2.53 GHz
Board	Winbond Electronics 0V52N7
Memory	8GB, AMI CMX8GX3M2A1333C9
Drives	1TB Seagate ST31000524AS, SATA
Network	1Gbps Ethernet
Traces	Storage Performance Council [43]

During experimentations, several parameters are considered. First, fixed migration chunk size is considered in the experiments, and the IOFollow system performance can be further improved aiming at sophisticated dynamic chunk size migration approaches. Second, we allocate the storage resource between VM threads and migration threads with pre-determined ratio. For instance, 2:1 means that the storage system will serve two VM I/O requests before performing one migration I/O request, unless there is no VM I/O request waiting in the IO queue. This approach is simple but effective to achieve a tradeoff between the VM performance and migration performance. The proposed IOFollow system can also improve the VM live storage migration performance under other storage resource allocation policies. Finally, the IOFollow system is evaluated under different number of concurrent VM live storage migrations.

The experimental platform consists of a server configured with one Intel Xeon X3440 processor, 8GB DDR memory and two 1TB hard drives, 12.10 Ubuntu operating system. Detailed information of hardware is depicted in Table 1.

In order to measure the performance improvement of IOFollow scheme, we replay block level traces and collect IO performance information during the migration process. The Storage Performance Council [43] has made available several block level traces for research purposes, and they have been widely employed to evaluate storage system performance [44,45]. The WebSearch traces are collected from a web search engine and the Financial2 trace is collected from OLTP applications in a large financial institution. The characteristics of the four traces are summarized in Table 2. From our experiments, we implemented a trace replay tool that will read trace files and generate the migration I/O requests according to the VM IO stream.

Table 2
The characteristics of four traces.

Trace	Read Ratio	IOPS	Avg. Req. Size (KB)
WebSearch1	100%	113	15.1
WebSearch2	100%	100.3	14.9
WebSearch3	100%	63.52	15.2
Financial2	82.4%	125	2.2

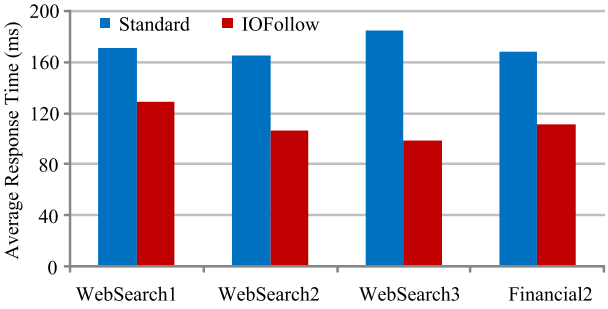


Fig. 3. The Performance Comparison in Single VM Migration.

4.2. Result analysis

In this scenario, there is only one VM migrating from the source to the destination. The migration chunk size is 1MB and the storage resource allocation ratio is 2:1 between VM threads and migration threads. Fig. 3 shows that IOFollow scheme reduces the average IO response time for the migration thread by $\frac{172-129}{172} = 25.0\%$, $\frac{166-107}{166} = 35.5\%$, $\frac{185-99}{185} = 46.5\%$ and $\frac{169-112}{169} = 33.7\%$ driven by different traces, compared to the standard migration scheme. The main reason is that, with the simple dynamic scheduling of migration block sequence, individual blocks are migrated when the disk head is moving close to it, so that unnecessary seek and rotation operations are reduced noticeably in migration I/O requests. Moreover, such scheduling makes easier for the disk controller to invoke internal optimizations, such as the requests merge, in order to further improve the IO performance. Therefore, the overall migration IO performance is improved significantly.

Fig. 4 shows the performance evaluation result for the VM IO thread. We have the following observations: (1) The Standard scheme increases the average IO response time by $\frac{60-35}{35} = 71.4\%$, $\frac{52-22}{22} = 136.4\%$, $\frac{68-25}{25} = 172.0\%$, $\frac{62-37}{37} = 67.6\%$ than that in the normal state driven by the four traces. This clearly indicates that the VM IO performance is significantly affected by the live storage migration jobs. Not only more I/O requests are generated by the migration thread, but also the access locality of the applications is destroyed but the new coming migration I/O requests. For instance, two consecutive read requests from the application become non-consecutive requests when a migration request is served in between, and the address of the migration request is far from that of the application requests, and (2) Compared to the standard migration approach, the average IO response

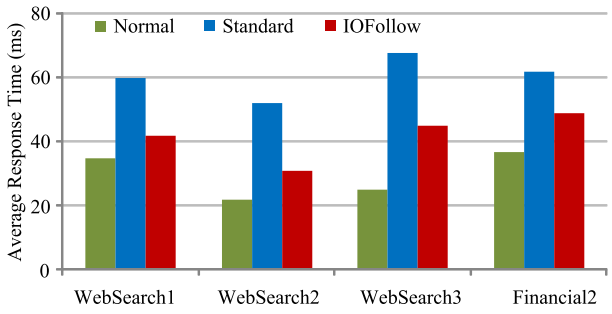


Fig. 4. The VM Performance Comparison in Single VM Migration.

time in the proposed IOFollow scheme is decreased by $\frac{60-42}{60} = 30.0\%$, $\frac{52-31}{52} = 40.4\%$, $\frac{68-45}{68} = 33.8\%$ and $\frac{62-49}{62} = 20.9\%$ driven by the four traces. The reason behind such improvement is that the interleaved IO stream at the gate of the storage system is becoming more sequential in the IOFollow scheme than that in the standard scheme. Therefore, the access locality of the applications can be reserved.

4.3. Sensitivity studies

In order to investigate how the migration chunk size, the storage allocation policy and the number of concurrent VM migrations affect the performance of IOFollow system, trace driven experiments are performed. The normalized response time of the IOFollow scheme based on the standard scheme under the same experiment configuration is analyzed.

4.3.1. Chunk size

The migration performance and VM performance between IOFollow and standard migration approach are analyzed and compared for traces under different migration chunk sizes (128 KB, 256 KB, 512 KB, 1MB and 2MB). The IO response time in IOFollow is normalized based on that in the standard migration. Fig. 5(a) shows IOFollow reduces the average IO response time for the migration thread from 45% to 65% with a mean value of 57.4%, for all the traces under different chunk sizes. Meanwhile, Fig. 5(b) shows IOFollow reduces the IO response time for the VM thread from 70% to 88% with a mean value of 80.3%. As a result, IOFollow can improve the VM live storage migration performance under different migration chunk sizes.

4.3.2. Resource allocation policy

Before the start of the VM live storage migration, the policy of storage resource allocation between VM IO threads and migration threads is determined either by system administrators or automatically. In this group of experiments, we evaluate the performance improvement of IOFollow scheme over standard migration under different storage allocation ratio (VM IO resource: migration resource) from 1:1, to 2:1, 3:1 and 4:1. As Fig. 6 shows, IOFollow scheme can improve the VM IO performance by up to 74% with a mean value of 58.7%, and reduce the IO response time for the migration thread by up to 50% with a mean value of 80.6%. As a number of running VMs share a single server and the server cannot satisfy the requirement of storage IO bandwidths for each individual VMs, one or more VMs will need to be live migrated to other servers. However, there is very limited storage IO bandwidth available for the new migration threads, as the total storage IO bandwidth is fixed. In this scenario, IOFollow is far more important for the VM IO performance, as it introduce less bandwidth consumption for the storage server when compared to the standard migration.

4.3.3. Concurrent VM migration

As discussed in previous sections, multiple concurrent VM live storage migration is common in current data center design. In such scenarios, the source server will undergo larger pressure as multiple IO hungry migration streams are introduced and the overall storage capacity for the source server remains the same as previous state. In order to investigate how much can IOFollow scheme improve the VM live storage migration performance compared to standard migration scheme in these scenarios, we conduct experiments to evaluate performance improvement under different traces as also different concurrent VM live storage migration.

Fig. 7 shows the migration performance and the VM performance by IOFollow, normalized to the standard migration scheme. We can see that IOFollow reduces the average IO response time by 55% for migration thread and 45% for VM IO thread, compared to standard migration. In addition, as the number of the concurrent migrating VMs increase, the performance improvement by IOFollow highlights. As multiple migration threads are introduced, it is better to schedule the VM I/O requests and migration I/O request of a single VM as a unit, and then round robin scheduling between multiple concurrent migrating VMs. Due to the targeting addresses of VM I/O requests and migration I/O requests for a single VM are close to each other, they will make the final I/O request stream in the disk control more sequential. Therefore, the IO performance is improved and the VM migration speed is also accelerated accordantly.

5. Related work

Most existing researches on the VM live migration can be classified into three categories: Live VM Memory Migration, Live Storage Migration, and Live Multiple VM Migrations.

Live VM Memory Migration: The VM live memory migration is very popular in the shared-storage environment, where both source and destination servers retain the access to the shared storage system. Besides the transfer of a variety of VM states, such as CPU state, network state, memory state and device state, transferring VM's memory state usually takes the largest portion of migration time and a number of approaches are proposed to accelerate the memory migration [46,47]. Changyeon et al. [48] proposed to track the duplicated memory pages in the source server during the runtime. As migration is triggered, instead of migrating these duplicated pages over the rate-limited connection to the destination server, it fetches these pages directly from the shared storage server. Therefore, the total data transmission is reduced significantly, whilst the live migration performance is improved as well [48]. Hai et al. [49] proposed to classify memory pages to several types according to different characteristics, such as high/low word similarity, number of zero bytes, and then adopt different compression algorithms to compress memory pages with different properties.

Live VM Storage Migration: Shrinker [21] is a distributed system that is capable of migrating a virtual cluster over WAN. It has two built-in services: Coordination Service (in the source site) and Indexing Service (in the destination site). The Coordination Service tracks the hash values of memory pages and virtual disk blocks that have already been transferred to the destination site, so that hypervisors in the source side can perform data deduplication by replacing duplicated transmission of memory pages and disk blocks with their corresponding hash values. The Indexing Service records the hash values and the location information of the memory pages and disk blocks in the destination side. Hypervisors in the destination can reconstruct the VM's memory and virtual disk images with the communication between Indexing Service and other hypervisors that hold the real data. Zhou et al. [50] take the speed discrepancy between HDDs and SSDs, and the wear-out issue of SSDs into consideration in order to optimize the live

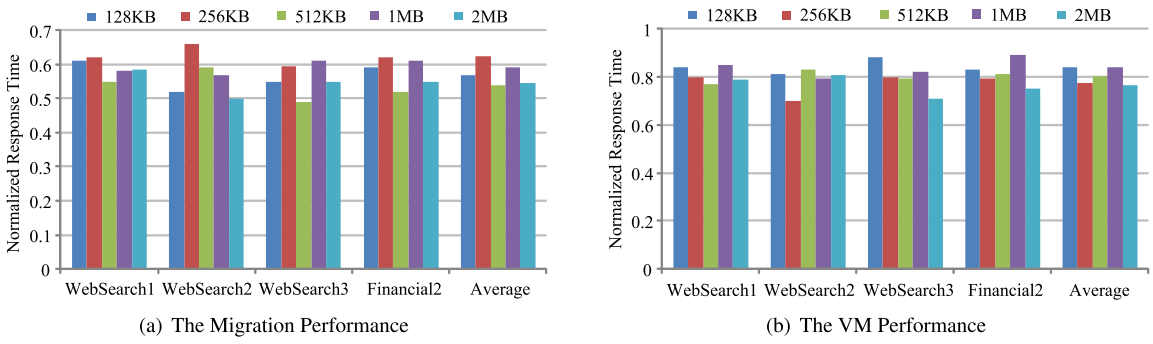


Fig. 5. The Migration Performance and the VM Performance Comparison in Single VM Migration among Different Migration Chunk Size.

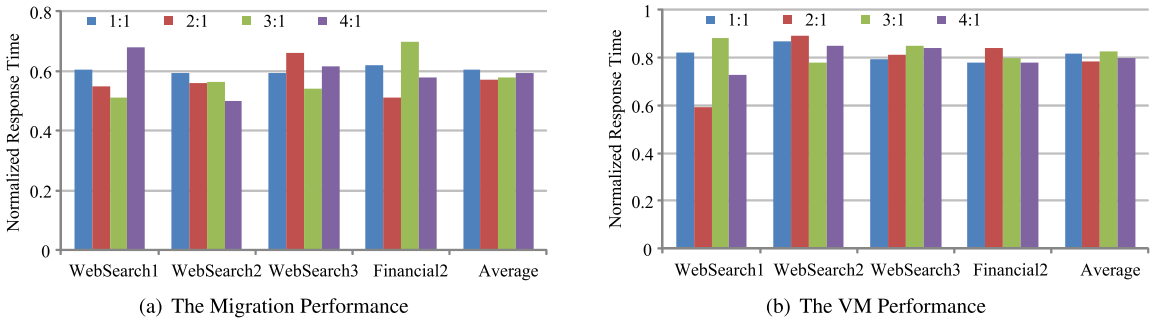


Fig. 6. The Migration Performance and the VM Performance Comparison in Single VM Migration among Different Storage Allocation Policy.

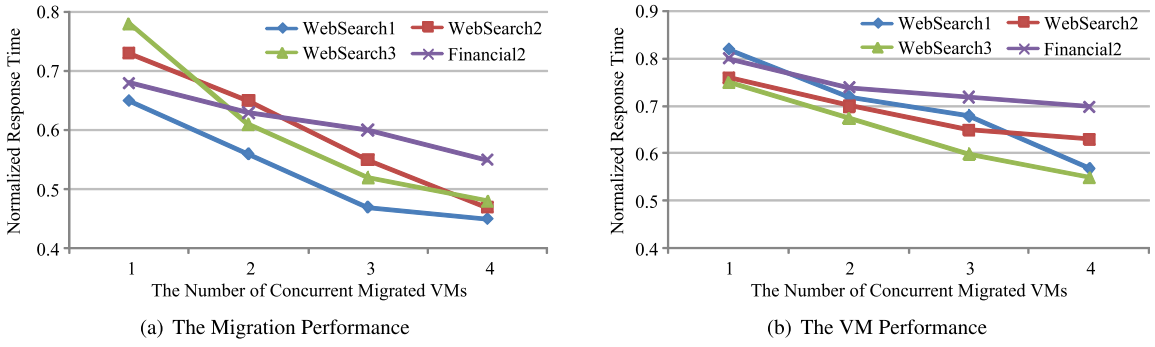


Fig. 7. The Migration Performance and the VM Performance Comparison among Multiple VMs Migration.

storage migration. Ali et al. [51] build XvMotion, a VM live storage migration system to migrate VMs over long distance across heterogeneous systems, with performance similar to that of VM migration in Local Area Networks. Xu et al. [52] propose SRVM that enables live migration with passthrough SRIOV VFs. Both XvMotion and SRVM improves the live VM storage migration by improving the network performance. Noting that the performance of the storage system in virtualized environments plays a vital role for the VM live storage migration performance. Several other studies also indirectly improve the VM live storage migration performance with contributions in IO scheduler [53], file systems [54], Solid State Drives [55] and data deduplication techniques [56].

Live Multiple VM Migrations: Given the widely deployment of VMs in current cloud data centers, it is not uncommon to migrate multiple VMs from/to a single server simultaneously. VMScatter [18] is a multicast-based VM live migration system, which can efficiently migrate a group of VMs from one shared source server to multiple destination servers. Sandpiper et al. [57] is designed and implemented as a system that contains two components: hotspot detection algorithm and hotspot migration algorithm. The hotspot detection algorithm will decide when to migrate VMs, whilst the hotspot migration algorithm will determine where to migrate and how much resources to allocate after the migration. Live Gang Migration [58] is inspired by the fact that co-located VMs often have many identical memory pages, such as same operating system, applications and libraries, same Java Virtual Machine (JVM). In this approach [58], identical memory pages will be duplicated prior to

the transmission of VM state, so that only a single memory page copy is migrated.

In principle, IOFollow system shares the same goal as previous research works, which is to speed up the VM live storage migration performance and provide reasonable IO performance for the migrating VM. However, this goal can be achieved in a few of different ways. IOFollow system leverages the existing idle base images and previous snapshots in backup servers for transmission of VM state information. Previous research works focused on the performance optimization techniques of the aggregate IO streams in the source server, such as leveraging the fast read performance of SSD and removing the redundant data transmission. As IOFollow system introduce very limited migration traffic to the source server during multiple VM live storage migrations, its performance is higher than the existing approaches. In addition, the proposed IOFollow system can be combined with these research works together to achieve higher VM live storage migration performance. The inevitable disadvantage involves more workloads generated on the backup servers. Given that such backup servers are available outside of the backup window, the workload rebalancing is admissible.

6. Conclusions and future work

Current VM live storage migration approaches ignore the Sequential IO property of the interleaving IO streams at the gate of the storage server, by simply migrating the VM's virtual disk images sequentially, regardless of the concurrent VM IO streams as well other migration streams. Therefore, many unnecessary time consuming disk head seek and rotation operations that degrades both the VM IO performance and migration performance were introduced. Based upon these observations, this paper presents IOFollow, a novel VM live storage migration scheme that improves both the VM IO performance and migration performance by generating and scheduling migration sequence according to the IO stream of VM requests. In this way, it is expected to minimize and get rid of unnecessary disk head movements, so the overall VM live storage migration performance can significantly be improved. Evaluations of trace-based experiments demonstrated that IOFollow can reduce the average I/O response time by 55% for migration thread and 45% for VM IO thread, compared to the standard migration.

IOFollow is an ongoing research project and we are currently exploring several directions as future research. First, we will leverage filesystem semantics, such as the block liveness, to reduce the amount of data transferred. Second, we will eliminate unnecessary transfer of redundant and similar data with the judicious combination of data deduplication and delta compression, thus reducing the total data transferred during the VM live storage migration [56]. Third, we will evaluate the IOFollow on SSDs and other non-volatile memory devices to investigate how does the performance of storage devices affect the VM migration efficiency.

Acknowledgment

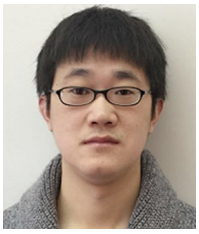
This work is supported by the National Natural Science Foundation of China under Grant No. 61872305, No. U1705261, No. 61772439, and No. 61472336, the US NSF under Grant No. CCF-1704504 and CCF-1629625.

References

[1] The virtualization techniques. https://www.fluxlabs.net/solutions/virtualization.
[2] T. Saber, J. Thorburn, L. Murphy, A. Ventresque, VM reassignment in hybrid clouds for large decentralised companies: A multi-objective challenge, Future Gener. Comput. Syst. 73 (3) (2018) 751–764.
[3] S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, E. Thereska, End-to-end performance isolation through virtual datacenters, in: Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation, 2014, pp. 233–248.
[4] C. Jo, E. Gustafsson, J. Son, B. Egger, Efficient live migration of virtual machines using shared storage, in: Proceedings of the 9th International Conference on Virtual Execution Environments, VEE'13, 2013, pp. 41–50.
[5] T. Hirofuchi, H. Ogawa, H. Nakada, S. Itoh, S. Sekiguchi, A live storage migration mechanism over WAN for relocatable virtual machine services on clouds, in: Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGrid'09, 2009, pp. 41–50.
[6] T. Huang, Y. Zhu, Y. Wu, S. Bressan, G. Dobbie, Anomaly detection and identification scheme for vm live migration in cloud infrastructure, Future Gener. Comput. Syst. 56 (2016) 736–745.
[7] F. Zhang, X. Fu, R. Yahyapour, LayerMover: Fast virtual machine migration over WAN with three-layer image structure, Future Gener. Comput. Syst. 83 (2018) 37–49.
[8] VMware vSphere ESX Support. https://www.vmware.com/support/esx.html.
[9] XenServer: open source virtualization. http://xenserver.org/. (Accessed 14 March 2016).
[10] Kernel based virtual machine. http://www.linux-kvm.org/page/Main_Page.
[11] Microsoft virtualization platform. https://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx. (Accessed 19 April 2016).
[12] Gartner special report. http://www.gartner.com/newsroom/id/2599315.
[13] K. Tsakalozos, V. Verroios, M. Roussopoulos, A. Delis, Live VM migration under time-constraints in share-nothing IaaS-clouds, IEEE Trans. Parallel Distrib. Syst. 28 (2017) 2285–2298.
[14] A. Mashtizadeh, E. Celebi, T. Garfinkel, M. Cai, et al., The design and evolution of live storage migration in VMware ESX, in: 2011 USENIX Annual Technical Conference, USENIX, 2011.
[15] J. Zheng, T.S.E. Ng, K. Sripanidkulchai, Workload-aware live storage migration for clouds, in: ACM Sigplan Notices, vol. 46(7), ACM, 2011, pp. 133–144.
[16] J. Zheng, T.S.E. Ng, K. Sripanidkulchai, Z. Liu, Comma: coordinating the migration of multi-tier applications, in: ACM SIGPLAN Notices, vol. 49(7), ACM, 2014, pp. 153–164.
[17] V. Medina, J. García, A survey of migration mechanisms of virtual machines, ACM Comput. Surv. 46 (2014) Article No. 30.
[18] L. Cui, J. Li, B. Li, J. Huai, C. Hu, T. Wo, H. Al-Aqrabi, L. Liu, VMScatter: migrate virtual machines to many hosts, in: ACM SIGPLAN Notices, vol. 48(7), ACM, 2013, pp. 63–72.
[19] S. Nathan, U. Bellur, P. Kulkarni, On selecting the right optimizations for virtual machine migration, in: Proceedings of the 12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'16, 2016, pp. 37–49.
[20] J. Zhang, E. Dong, J. Li, H. Guan, MigVisor: Accurate prediction of VM live migration behavior using a working-set pattern model, in: Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'17, 2017, pp. 30–43.
[21] P. Riteau, C. Morin, T. Priol, Shrinker: improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing, in: Proceedings of the International European Conference on Parallel and Distributed Computing, Springer, 2011, pp. 431–442.
[22] R. Zhou, F. Liu, C. Li, T. Li, Optimizing virtual machine live storage migration in heterogeneous storage environment, in: ACM SIGPLAN Notices, vol. 48(7), ACM, 2013, pp. 73–84.
[23] Y. Yang, H. Jiang, B. Mao, L. Tian, Y. Yang, J. Qian, WAIO: Improving virtual machine live storage migration for the cloud by workload-aware IO outsourcing, in: Proceedings of the IEEE 7th International Conference on Cloud Computing Technology and Science, 2015, pp. 314–321.
[24] Y. Yang, B. Mao, H. Jiang, H. Luo, Y. Yang, S. Wu, SnapMig: Accelerating VM live storage migration by leveraging the existing VM snapshots in the cloud, IEEE Trans. Parallel Distrib. Syst. 29 (2018) 1416–1427.
[25] C. Li, P. Shilane, F. Douglis, D. Sawyer, H. Shim, Assert (! Defined (Sequential I/O)), in: Proceedings of the USENIX conference on Hot Topics in Storage and File Systems, 2014.
[26] R.H. Arpaci-Dusseau, A.C. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, Arpaci-Dusseau Books, 2015.
[27] M. Lillibridge, K. Eshghi, D. Bhagwat, Improving restore speed for backup systems that use inline chunk-based deduplication, in: Proceedings of the 11th USENIX Conference on File and Storage Technologies, 2013, pp. 183–197.
[28] S. Jiang, X. Ding, F. Chen, E. Tan, X. Zhang, DULO: an effective buffer cache management scheme to exploit both temporal and spatial locality, in: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies, vol. 4, 2005, p. 8.
[29] Y. Zhou, J. Philbin, K. Li, The multi-queue replacement algorithm for second level buffer caches, in: USENIX Annual Technical Conference, General Track, 2001, pp. 91–104.
[30] A. Wildani, E.L. Miller, L. Ward, Efficiently identifying working sets in block i/o streams, in: Proceedings of the 4th Annual International Conference on Systems and Storage, ACM, 2011, p. 5.

[31] X. Lin, Y. Mao, F. Li, R. Ricci, Towards fair sharing of block storage in a multi-tenant cloud, in: Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing, 2012, p. 15.
[32] M. Rosenblum, J.K. Ousterhout, The design and implementation of a log-structured file system, ACM Trans. Comput. Syst. 10 (1) (1992) 26–52.
[33] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, L. Vivier, The new ext4 filesystem: current status and future plans, in: Proceedings of the Linux symposium, vol. 2, Citeseer, 2007, pp. 21–33.
[34] O. Rodeh, J. Bacik, C. Mason, BTRFS: The Linux B-tree filesystem, ACM Trans. Storage 9 (3) (2013) 9.
[35] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies, IEEE, 2010, pp. 1–10.
[36] S. Ghemawat, H. Gobioff, S.T. Leung, The Google file system, in: ACM SIGOPS operating systems review, vol. 37(5), ACM, 2003, pp. 29–43.
[37] M. Sivathanu, L.N. Bairavasundaram, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, Database-Aware semantically-smart storage, in: Proceedings of the USENIX Conference on File and Storage Technologies, vol. 5, 2005, p. 18.
[38] A. Ruprecht, D. Jones, D. Shiraev, G. Harmon, M. Spivak, M. Krebs, M. Baker-Harvey, T. Sanderson, VM live migration at scale, in: Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'18, 2018, pp. 45–56.
[39] QEMU system main page. http://wiki.qemu.org/Main_Page.
[40] VMWare virtual machine file system overview. https://www.vmware.com/pdf/vmfs-best-practices-wp.pdf.
[41] Tintri VMStore storage system. https://www.tintri.com.
[42] QEMU storage stack. https://events.linuxfoundation.org/slides/2011/linuxcon-japan/lcj2011_hajnoczi.pdf.
[43] Umass storage trace repository. http://traces.cs.umass.edu/index.php/Storage/Storage.
[44] A. Miranda, T. Cortes, CRAID: online RAID upgrades using dynamic hot data reorganization, in: Proceedings of the USENIX Conference on File and Storage Technologies, 2014, pp. 133–146.
[45] S. Wu, H. Jiang, D. Feng, L. Tian, B. Mao, WorkOut: I/O workload outsourcing for boosting RAID reconstruction performance, in: Proceedings of the USENIX Conference on File and Storage Technologies, vol. 9, 2009, pp. 239–252.
[46] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines, in: Proceedings of the USENIX Symposium on Networked Systems Design & Implementation, 2005, pp. 273–286.
[47] M. Nelson, B.H. Lim, G. Hutchins, et al., Fast transparent migration for virtual machines, in: USENIX Annual Technical Conference, General Track, 2005, pp. 391–394.
[48] C. Jo, E. Gustafsson, J. Son, B. Egger, Efficient live migration of virtual machines using shared storage, in: ACM Sigplan Notices, vol. 48(7), ACM, 2013, pp. 41–50.
[49] H. Jin, L. Deng, S. Wu, X. Shi, X. Pan, Live virtual machine migration with adaptive, memory compression, in: Proceedings of the International Conference on Cluster Computing, IEEE, 2009, pp. 1–10.
[50] R. Zhou, F. Liu, C. Li, T. Li, Optimizing virtual machine live storage migration in heterogeneous storage environment, in: ACM SIGPLAN Notices, vol. 48(7), ACM, 2013, pp. 73–84.
[51] A.J. Mashtizadeh, M. Cai, G. Tarasuk-Levin, R. Koller, T. Garfinkel, S. Setty, XvMotion: unified virtual machine migration over long distance, in: Proceedings of the 2014 USENIX Annual Technical Conference, 2014, pp. 97–108.
[52] X. Xu, B. Davda, SRVM: Hypervisor support for live migration with passthrough SR-IOV network devices, in: Proceedings of the 12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'16, 2016, pp. 65–77.
[53] S. Yang, T. Harter, N. Agrawal, S.S. Kowsalya, A. Krishnamurthy, S. Al-Kiswany, R.T. Kaushik, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, Split-level I/O scheduling, in: Proceedings of the 25th Symposium on Operating Systems Principles, ACM, 2015, pp. 474–489.
[54] D. Le, H. Huang, H. Wang, Understanding performance implications of nested file systems in a virtualized environment, in: Proceedings of the USENIX Conference on File and Storage Technologies, 2012, p. 8.
[55] T.Y. Kim, D.H. Kang, D. Lee, Y.I. Eom, Improving performance by bridging the semantic gap between multi-queue SSD and I/O virtualization framework, in: Proceedings of the 31st Symposium on Mass Storage Systems and Technologies, IEEE, 2015, pp. 1–11.
[56] H. Li, W. Li, Q. Feng, S. Zhang, H. Wang, J. Wang, Leveraging content similarity among vmi files to allocate virtual machines in cloud, Future Gener. Comput. Syst. 79 (2018) 528–542.
[57] T. Wood, P.J. Shenoy, A. Venkataramani, M.S. Yousif, Black-box and gray-box strategies for virtual machine migration, in: Proceedings of the USENIX Symposium on Networked Systems Design & Implementation, vol. 7, 2007, p. 17.
[58] U. Deshpande, X. Wang, K. Gopalan, Live gang migration of virtual machines, in: Proceedings of the 20th International Symposium on High Performance Distributed Computing, ACM, 2011, pp. 135–146.

Bo Mao received the B.E. degree in Computer Science and Technology in 2005 from Northeast University; and the Ph.D degree in Computer Architecture in 2010 from the Huazhong University of Science and Technology. His research interests include storage system, Cloud computing and Big Data. He is an associate professor at the Software School of Xiamen University. He has over 40 publications in IEEE-TC, IEEE-TPDS, IEEE-TCAD, ACM-TOS, FGCS, USENIX FAST, LISA, ICS, IPDPS, ICCD, and Cluster. Dr. Mao is a Member of IEEE and ACM.



Yaodong Yang received the B.Sc. degree in Computer Engineering in 2008 from Tianjin University, China; the M.A.Sc. degree in Computer Engineering in 2011 from the Huazhong University of Science and Technology, China; and the Ph.D. degree in Computer Science and Engineering Department in 2016 from the University of Nebraska-Lincoln, USA. His research interests include flash-based storage systems, VM storage migration, and cloud storage. He is currently a software engineer in the Azure Service Fabric division within Microsoft, in Redmond, USA.

Suzhen Wu received the B.E. and Ph.D. degrees in Computer Science and Technology and Computer Architecture from Huazhong University of Science and Technology, in 2005 and 2010 respectively. She is an associate professor at Computer Science Department of Xiamen University since August 2014. Her research interests include computer architecture and storage system. She has over 40 publications in IEEE-TC, IEEE-TPDS, IEEE-TCAD, ACM-TOS, FGCS, USENIX FAST, LISA, IPDPS, ICS, ICCD, and ICPADS. Dr. Wu is a Member of IEEE and ACM.

Hong Jiang received the B.Sc. degree in Computer Engineering in 1982 from Huazhong University of Science and Technology, China; the M.A.Sc. degree in Computer Engineering in 1987 from the University of Toronto, Canada; and the PhD degree in Computer Science in 1991 from the Texas A&M University, USA. He is currently Chair and Wendell H. Nedderman Endowed Professor of Computer Science and Engineering Department at the University of Texas at Arlington. Prior to joining UTA, he served as a Program Director at National Science Foundation (2013.1-2015.8) and he was at University of Nebraska-Lincoln since 1991, where he was Willa Cather Professor of Computer Science and Engineering. His present research interests include computer architecture, computer storage systems and parallel I/O, high performance computing, big data computing, cloud computing. He has over 200 publications in major journals and international Conferences in these areas, including IEEE-TPDS, IEEE-TC, PIEEE, ACM-TACO, JPDC, ISCA, MICRO, USENIX ATC, FAST, EuroSys, LISA, SIGMETRICS, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc., and his research has been supported by NSF, DOD, the State of Texas and the State of Nebraska. Dr. Jiang is a Fellow of IEEE, Member of ACM and USENIX.

Kuan-Ching Li is a Professor of Computer Science and Engineering at Providence University, Taiwan. He is a recipient of guest and distinguished chair professorships from universities in China and other countries, and awards and funding support from a number of agencies and industrial companies. He is a fellow of the IET, a senior member of the IEEE and a member of the AAAS. Besides publishing numerous research papers, he is the co-author/co-editor of more than 10 technical professional books published by CRC Press, Springer, McGraw-Hill and IGI Global. His research interests include GPU/many-core computing, Big Data and Cloud.