BAC: Bandwidth-Aware Compression for Efficient Live Migration of Virtual Machines

Chunguang Li, Dan Feng*, Yu Hua, Wen Xia, Leihua Qin, Yue Huang, Yukun Zhou
Wuhan National Laboratory for Optoelectronics, School of Computer
Key Laboratory of Information Storage System, Ministry of Education
Huazhong University of Science and Technology
*Corresponding author: dfeng@hust.edu.cn

***Abstract*—Live migration of virtual machines (VM) is one of the key characteristics of virtualization for load balancing, system maintenance, power management, etc., in data centers or clusters. In order to reduce the data transferred and shorten the migration time, the compression techniques have been widely used to accelerate VM migration. However, different compression approaches have different compression ratios and speeds. Because there is a trade-off between compression and transmission, the migration performance improvements obtained from different compression approaches are differentiated, and the improvements vary with the network bandwidth. Besides, the compression window sizes used in most compression algorithms are typically much larger than a single page size, so the traditional single page compression loses some potential compression benefits. In this paper, we design and implement a Bandwidth-Aware Compression (BAC) scheme for VM migration. BAC chooses suitable compression approach according to the network bandwidth available for the migration process, and employs multi-page compression. These features make BAC obtain more migration performance improvements from compression. Experiments under various network scenarios demonstrate that, compared with conventional compression approaches, BAC shortens the total migration time while achieving comparable performance for the total data transferred and the downtime.**

I. Introduction

As a key technique to implement cloud computing systems, virtualization [1] offers many features, such as abstraction from heterogeneous hardware, security isolation, convenience for management, etc. Live migration [2], which is interpreted as moving a running virtual machine (VM) from one physical host to another, is an important characteristic of virtualization. Live migration is powerful in data centers due to the use for load balancing [3], [4], system maintenance [5], datacenter network optimization [6], [7], [8], fault tolerance [9], [10], power management [11], [12], etc.

Inside data centers, the source and target hosts for migration often share the same disk storage (e.g., through SAN or NFS), so only the VCPU state and the content of VM memory need to be transferred. Pre-copy [2], [13] is the prevailing approach for VM migration with shared storage. Pre-copy sends the memory content from the source node to the target node in several iterations to minimize the *downtime*, during which the VM is not running. In each iteration, only the pages that are modified during last round are sent. Because of its iterative process, pre-copy may cause large amount of data transferred on the network and long migration time. To accelerate the migration process, previous schemes use the compression techniques [14], [15], [16].

When choosing a suitable compression approach for migration, we look forward to obtaining the goals of both high compression ratio and fast compression speed. However, for the various compression approaches, there is always a trade-off between the compression ratios and speeds. As a result, different compression approaches lead to differentiated migration performance. When we consider the trade-off between the compression of the VM memory and the transmission of the compressed data, a suitable compression approach should be chosen according to the network bandwidth allocated to the migration process. For example, when network bandwidth is low, a slow compression approach with good compression ratio is suitable, because the size of data needed to be transferred is significantly reduced. Otherwise, when the network bandwidth is high, the same compression approach would become the bottleneck of migration. So we design a Bandwidth-Aware Compression (BAC) scheme for live VM migration, using a *list of compression strategies*, based on the analysis of the memory contents of VMs carrying out various workloads.

Because the VM memory is organized in the unit of pages, existing scheme [14] compresses the memory contents during migration at the granularity of a single page. However, the compression window sizes used in most compression algorithms (e.g. 64KB for LZ4) are typically much larger than a single page size (typically 4KB), and compression algorithms identify redundant strings within the area of this compression window. Hence conventional single page compression method loses some potential compression benefits. So we propose the use of multi-page compression to get more performance improvements from compression during VM migration.

The main contributions of this paper are listed as follows:

(1) We propose a Bandwidth-Aware Compression (BAC) scheme for live VM migration, which chooses the suitable compression approach according to the network bandwidth available for migration and employs the multi-page compression method. Evaluation results with various network scenarios show that, our proposed scheme leads to significant performance improvements comparing with the conventional compression approaches.

(2) We derive a *list of compression strategies* through the analysis of the memory contents of VMs carrying out various workloads. The analysis suggests that, although the memory contents with different workloads differ in compression ratios and speeds, this *list* applies to all of them. This makes our BAC scheme practical for implementation, so that we do not have to store separate *lists of compression strategies* for different

workloads.

The rest of this paper is organized as follows: Section II gives an overview of background on live VM migration. Our motivation is shown in Section III. In Section IV, we analyze the memory contents of VMs running various workloads. Section V details the design and implementation of BAC. Section VI shows the evaluation results. Section VII draws the conclusions and presents the future work.

II. Background

In this section, we introduce the background about live VM migration, including the key metrics for migration, the pre-copy migration algorithm, and existing techniques to improve migration performance.

There are three key metrics for live migration: total migration time (TMT), total data transferred (TDT), and downtime. TMT refers to the elapsed time from the beginning of migration to the moment when VM can run on the target host independently. It is critical to shorten the TMT [17]. The purpose of migration is to perform system maintenance or load balancing, etc., so longer TMT may lead to the misses of the better migration opportunities and make the system maintenance or load balancing less effective. Besides, during migration, the applications inside the migrated VM may suffer from performance degradation due to the overhead of migration thread for the CPU and network resources, and thus a shorter TMT alleviates performance degradation. Furthermore, TDT is the amount of data transferred from the source node to the target node for migration, and it indicates the network resource consumed by migration. Because the migration process and applications running inside the VMs often share the same network infrastructure, less TDT would leave more bandwidth to applications during migration. Moreover, downtime is the period from the time the migrated VM is suspended on the source node to the time it is resumed on the target node. This period of downtime should be short enough so that it is not perceived obviously by users.

Pre-copy [2], [13] is the prevailing approach for migration and has been widely used in many virtualization platforms, such as VMware [13], KVM [18], and Xen [1]. It works as follows. The bulks of the VM's memory pages are transferred to the target node while the VM is still running at the source node. The pages which are dirtied during the transfer are resent to the target node in the next round. This iterative process comes to the end when there are so little dirty pages remaining that the expected time to transfer them is shorter than the acceptable maximum downtime. Then the migration process steps into the *stop and copy* phase: the VM is suspended at the source node and the remaining dirty pages along with the VCPU state are transferred to the target node. After that, the VM continues to be executed at the target node. Pre-copy's overriding goal is to keep the downtime small. However, as pre-copy needs to transmit parts of the memory pages many times, it often causes large amount of data transferred on the network and long migration time. To address the problem of pre-copy, other live migration algorithms such as post-copy [19], [20] and CR/TR-Motion [21] are also proposed in previous works. However, they are seldom used in production environments because of their lack of reliability or practicality.

To reduce the data transferred during pre-copy migration and shorten the migration time, previous schemes use the techniques of compression [14], [15], [16], [22]. However, none of them takes network bandwidth into consideration when leveraging compression to improve the migration performance. Our analysis in the next section shows that, a suitable compression approach should be chosen according to the network bandwidth allocated to the migration process. Besides, while previous works compress the VM memory contents at the granularity of single page, we demonstrate that a larger compression window obtains more performance improvements for migration.

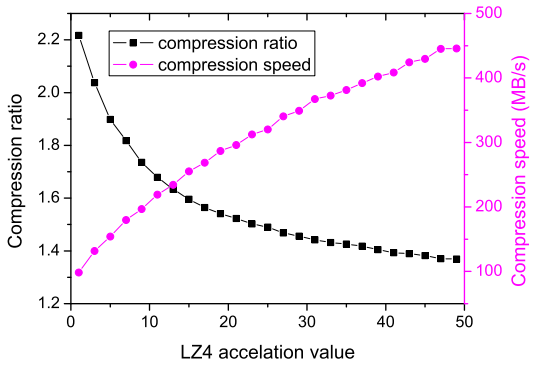


Fig. 1. The variation of compression ratios and speeds when the memory content of a VM is compressed using LZ4 with different acceleration values. The VM is allocated with 1 GB RAM and runs kernel compilation. The experiment is conducted on machines with quad-core Xeon X3220 2.4GHz CPU.

III. Motivation

A. Considering Network Bandwidth When Using Compression in VM Migration

Different compression approaches vary in compression ratios and speeds. Generally speaking, the approaches with better compression ratio may sacrifice compression speed, and vice versa. In addition, many compression algorithms offer trade-offs between the two metrics through some ways, e.g., setting parameters in the APIs (such as LZ4 [23]), or defining macros in the head files (such as QuickLZ [24]). Thus users can choose among different compression approaches according to their requests. In this paper, LZ4 is chosen for the compression in VM migration, because 1) it is quite fast, which is suitable for memory compression; 2) it provides abundant trade-offs between compression ratios and speeds.

LZ4 offers a parameter, called "acceleration", in its compression function. Figure 1 shows the variation of the compression ratios[1] and speeds when the memory content of a VM running kernel compilation is compressed using LZ4 with different acceleration values. The value is adjusted from 1 to 49 by the step of 2. It is shown that when the compression ratio is decreased from 2.22 to 1.37, the compression speed improves from 98MB/s to 446MB/s.

In the real-world data centers, the network situation varies over time. The administrators of data centers would also use the adaptive rate limiting [1] to adjust the bandwidth allocated to the migration process according to the requirements of applications running inside the VMs, because the migration process and the applications often compete for the same network resources. These factors lead to the changes of the network bandwidth available for VM migration.

[1]Compression ratio = size before compression / size after compression.

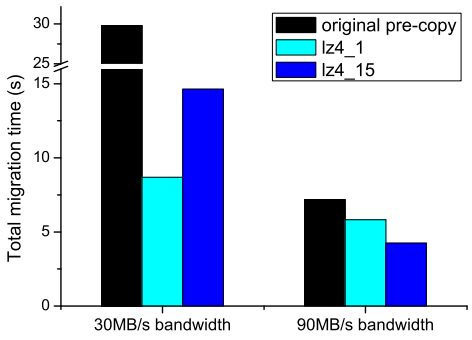


Fig. 2. Total migration time with different network bandwidths. The VM has 1GB memory and is running kernel compilation during migration.

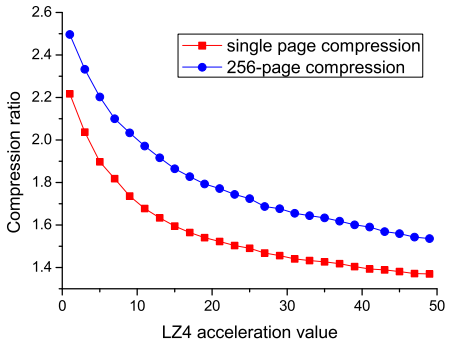


Fig. 3. The compression ratios when we compress the VM memory content at different granularities. Note that the compression speeds of these two different granularities have no significant difference.

TABLE I. SOME NOTATIONS WE DEFINED IN THIS PAPER

Notations	Description
S_t	the average network bandwidth available for migration, which is the transfer speed of the VM memory content.
S_{mgt}	the average migration speed of the VM memory content, which equals S_t without compression.
S_c	the average compression speed of the compression approach we used.
ρ	the average compression ratio of the compression approach we used, which equals the data size before compression divided by the size after compression.

When we consider the trade-off between compression and transmission, LZ4 approaches with different acceleration values should get different performance improvements for migration, and the improvements would vary with network bandwidth. We conduct the following experiment of VM migration to verify this consideration. We allocate the VM with 1GB memory, and a Linux kernel compilation workload is running during migration. We conduct the experiment with the original pre-copy migration and the migration using LZ4 with acceleration values 1 and 15 respectively. Besides, the experiment is repeated with 30MB/s and 90MB/s network bandwidths. In our gigabit network environment, 30MB/s bandwidth represents the situation when the network inside a data center is busy and the administrator allocates low bandwidth to the migration process, to guarantee the performance of applications inside the data center; on the other hand, 90MB/s bandwidth represents the situation when the network is idle and the administrator allocates high bandwidth to the migration process.

Figure 2 shows the result of the total migration time. It demonstrates that, although both LZ4 approaches outperform the original pre-copy, their improvements differ and vary with network bandwidth. When the bandwidth is low, LZ4_1[2] obtains more performance improvements than LZ4_15, and the contrary is the case with the high bandwidth.

The following analysis explains this phenomenon in details. We first define some notations in Table I. When compression is added into the process of migration, the size of the VM memory content needed to be transferred after compression is $1/\rho$ of the original size. This implies that S_t is augmented indirectly by a factor of ρ. The compression of the memory content and the transmission of the compressed content are parallel, because we compress the next package when the last compressed package is being transferred. So the migration speed with compression is the minimum of the compression speed and the augmented transfer speed, i.e.

$$S_{mgt} = min(S_c, S_t * \rho) \quad (1)$$

The bigger the S_{mgt} is, the shorter the total migration time (TMT) is. So given a specific value of S_t, the compression approach with the biggest S_{mgt} will lead to the shortest TMT. This is the basic idea that motivates us to design a bandwidth-aware compression scheme for the VM migration.

B. Multi-page Compression

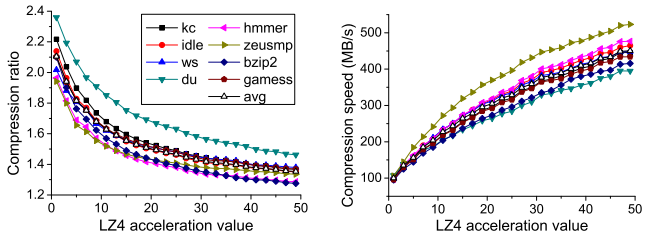
Most compression techniques identify redundant strings within a window of data. A larger window size offers opportunities to find more redundant strings, leading to better compression ratio, at the cost of more compression overheads [25]. In general, the implementation of LZ4 uses a 64 KB compression window, which is much larger than a single page size (4KB). Thus the single page compression approach, which has been widely used, misses the potential opportunities to find redundant strings in a larger area. Figure 3 verifies this consideration. We compress the memory content of a VM running kernel compilation with different granularities: single page (4KB) and 256 pages (1MB). While the resulting compression speeds have no significant difference, Figure 3 shows that the compression ratio of 256-page compression is improved compared with the single page compression. This motivates us to compress memory content at the granularity of multiple pages.

IV. ANALYSIS OF THE VM MEMORY

Formula (1) in the former section indicates that we need to first get the precise compression ratios (ρ) and speeds (S_c) of VM memory using different compression approaches, in order to figure out which one has the biggest S_{mgt}, thus leading to the shortest TMT. However, when different workloads are running inside the VM, the compression ratios and speeds would differ when the memory content is compressed. This is a big challenge for the implementation of our BAC scheme, because it needs us to use separate sets of compression ratios and speeds to compute the S_{mgt} for different workloads. However, the analysis of the VM memory in this section is helpful to address this challenge.

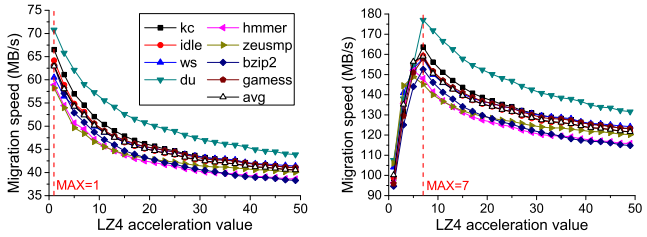
We select eight representative workloads in virtualization environments to get comprehensive understanding of the com-

[2]In this paper, LZ4_n denotes LZ4 using n as acceleration value.



(a) Compression ratio (b) Compression speed

Fig. 4. The results of compressing memory content of VMs running 8 different workloads using LZ4 with varying acceleration values. The experiment is conducted on machines with quad-core Xeon X3220 2.4GHz CPU.



(a) $S_t = 30MB/s$ (b) $S_t = 90MB/s$

Fig. 5. S_{mgt} of LZ4 using varying acceleration values with different network bandwidths.

pression ratios and speeds. Table II shows the details of these workloads. We also show the page dirtying speed (S_d) of these workloads in Table II, because it is an important factor to influence the performance of VM migration. Note that although S_d has an effect on the TMT and TDT, it has no relation with which compression approach we should choose. This is because as is shown in Formula (1), the migration speed (S_{mgt}) is not influenced by S_d. We expect to get the biggest S_{mgt} when we choose among different compression approaches, so we do not need to consider S_d.

To get the compression ratios and speeds of the VM memory during migration process, we modify the code of migration routine in QEMU [26]. We use LZ4 to compress memory pages before they are transferred to the target node. Besides, at the beginning of each iteration of pre-copy, we reset all the bits in the dirty bitmap of VM memory, to make all the pages to be compressed and transferred during every iteration. In different iterations, the acceleration value is adjusted in the LZ4 compression API from 1 to 49 by a step of 2. We accumulate the time and the compressed size of each compression operation, to compute the compression ratios and speeds of LZ4 with different acceleration values.

Figure 4 shows the results. For all the workloads, when the compression speed gets faster, the compression ratio becomes worse, and the results have a big gap between LZ4_1 and LZ4_49. Besides, the results differ a lot between different workloads, because their memory contents are quite different. The figures also show the average values for the compression ratios and speeds of the 8 workloads.

We further compute the S_{mgt} of LZ4 with varying acceleration values using formula (1) $S_{mgt} = min(S_c, S_t * \rho)$ for each pair of the compression ratios and speeds. S_t is set to be 30MB/s and 90MB/s respectively as the representations of the low and high network bandwidth situations. Figure 5 shows the results. Because different workloads vary in compression

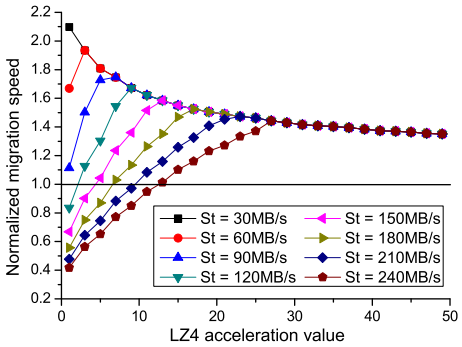


Fig. 6. Normalized S_{mgt} with varying S_t, computed by the average compression ratios and speeds of the eight workloads. Normalization means that S_{mgt} is divided by the network transfer speed S_t.

ratios and speeds, their values of S_{mgt} vary too. When $S_t = 30MB/s$, all of the workloads get the biggest S_{mgt} with LZ4_1, so does the average value. When $S_t = 90MB/s$, the average value gets the biggest S_{mgt} with LZ4_7. Meanwhile, each of the 8 workloads either gets the biggest S_{mgt} or is quite close to the biggest S_{mgt} with LZ4_7. Taking hmmer as an example, its S_{mgt} with LZ4_7 is 148MB/s, while it gets the biggest S_{mgt} of 152MB/s with LZ4_5, which are very close.

So we argue that, although various workloads differ in the compression ratios and speeds, and so differ in the S_{mgt}, all of their S_{mgt} get the maximum with almost the same LZ4 acceleration value. This makes it feasible to use the average values of the compression ratios and speeds, to compute the S_{mgt} for all of the VM workloads. Thanks to this significant finding, the implementation of our BAC scheme is simplified without storing separate sets of the compression ratios and speeds for different workloads.

Next, we continue to compute the S_{mgt} when the network bandwidth is further improved, using the average compression ratios and speeds. As shown in Figure 6, we present the result as normalized S_{mgt} (which is S_{mgt}/S_t) to indicate the factor by which S_{mgt} is augmented using compression. From the result we have three insights. First, when the acceleration value is not very big, the normalized S_{mgt} of the same acceleration value drops as the bandwidth improves. This implies that compression obtains more performance improvements for migration with lower network bandwidth. Second, the biggest S_{mgt} occurs at bigger acceleration value as the bandwidth improves. This implies that we should sacrifice the compression ratio to use a faster approach with higher network bandwidth. Third, some slow compression approaches with good compression ratio lead to a normalized S_{mgt} less than 1, when the bandwidth is high. This means that the compression would worsen the migration performance in such a situation.

From the results above, we conclude three cases in brief:

Case 1: When the network bandwidth is low, a slow compression approach with good compression ratio is suitable for migration;

Case 2: When the network bandwidth is high, a fast compression approach which sacrifices the compression ratio is suitable for migration;

Case 3: When the network bandwidth fluctuates during migration, the compression approach needs to be adjusted dynamically according to the bandwidth.

TABLE II. THE EIGHT REPRESENTATIVE WORKLOADS WE SELECT IN VIRTUALIZATION ENVIRONMENT.

Workloads	Description	S_d
Kernel Compilation	A linux-3.7.9 kernel is compiled, which is a balanced workload to test the performance of system virtualization.	Middle
Idleness	The virtual machine boots up without running any application.	Slow
Web Server	An apache web server which serves static content at a high rate is running. One client is configured with 100 concurrent connections while each connection continuously requests random 512KB files from the web server.	Middle
Daily Use	Some daily operations are carried out inside the VM, such as browsing websites using Firefox browser, playing a 720p high definition movie, and editing some documents with the LibreOffice.	Slow
CPU2006.Hmmer	This is an integer benchmark, which performs protein sequence analysis using profile hidden Markov models.	Slow
CPU2006.Bzip2	This is another integer benchmark, which performs Seward's bzip2 version 1.0.3, modified to do most work in memory.	Fast
CPU2006.Zeusmp	This is a floating point benchmark, which performs the simulation of astrophysical phenomena using computational fluid dynamics.	Fast
CPU2006.Gamess	This is another floating point benchmark, which implements a wide range of quantum chemical computations.	Slow

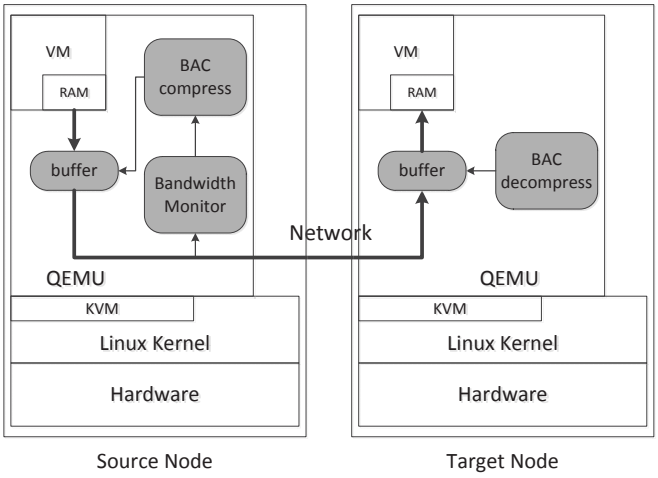


Fig. 7. The system architecture of BAC.

V. DESIGN AND IMPLEMENTATION

The analysis in the former section demonstrates that the acceleration value of LZ4 with which we will obtain the shortest TMT varies with network bandwidth. In the real-world environment, the network situation varies over time. The administrators of data centers would also use the adaptive rate limiting to adjust the bandwidth allocated to migration process according to the requirements of applications running inside data centers. So we design a Bandwidth-Aware Compression (BAC) scheme for live VM migration, which has been implemented in the KVM/QEMU platform. Figure 7 presents the system structure of our BAC scheme. Here we present the details of BAC as follows.

A. Bandwidth-Aware Compression

In our BAC scheme, the source node of the migration to perform the compression operations stores a *list of compression strategies*, along with the average *compression ratios* and *speeds* we have obtained in Section IV. When *strategy i* in the list is selected, it means compressing the VM memory content via LZ4 with acceleration value *i*. BAC monitors the network to get the bandwidth allocated to the migration process, and then selects the best compression strategy whose S_{mgt} is the biggest, which is computed based on formula (1). We could leverage the bandwidth monitor to get the real-time bandwidth periodically (e.g. once per second), so that the best compression strategy is adjusted dynamically according to the network bandwidth. The decompression at the target node in our BAC scheme does not need additional operations since the LZ4 decompression API is universal no matter what the acceleration value is.

As we have discussed in Section IV, we store the average values of the compression ratios and speeds of the eight workloads into the *list of compression strategies*, and they would apply to all workloads running inside the VM. Our implementation stores 16 pairs of compression ratios and speeds, from LZ4_1 to LZ4_31 by the step of 2. We do not store all of the 31 pairs because the S_{gmt} changes only slightly between two adjacent acceleration values. The pseudo-code to choose among *compression strategies* is presented in Algorithm 1. Note that in Algorithm 1, $strategies[i][0]$ refers to the compression ratio (ρ), and $strategies[i][1]$ refers to the compression speed (S_c).

Algorithm 1 Pseudo-code to choose among *compression strategies*

Input: current network bandwidth, St
Output: the LZ4 acceleration value which earns the most benefits for migration, $best$

```
Predefined values: the list of compression strategies, strategies[16][2]
i ← 0
max_Smgt ← 0
while i < 16 do
    if strategies[i][1] < St * strategies[i][0] then
        Smgt ← strategies[i][1]
    else
        Smgt ← St * strategies[i][0]
    end if
    if Smgt > max_Smgt then
        max_Smgt ← Smgt
        best ← 2 * i + 1
    end if
end while
return best
```

Besides, existing implementation of the pre-copy migration in QEMU employs a standard optimization to compress uniform pages, of which each byte is the same, such as zero pages. QEMU uses only one byte (with a flag to distinguish) to represent one entire 4KB uniform page when being transferred in migration. We keep this optimization in our implementation, and the uniform pages are not compressed by LZ4.

B. Multi-Page Compression

As we have stated in Section 3, a larger compression window contributes to better compression ratio for the VM memory, without obvious impact on compression speed. So in our BAC scheme, we maintain a buffer in the source node to store the pages being transferred temporarily. When the buffer is full, the memory pages in the buffer will be compressed together and then be transferred to the target node. The size of the buffer is adjustable, and a 1MB buffer has been proven to work well in practice. At the target node, there is also a buffer to receive the compressed multi-page content. After decompression, the pages are submitted to the RAM space of VM.

VI. Evaluation

This section presents the performance characteristics of the proposed techniques. We select three representative workloads (kernel compilation, zeusmp, and web server) from the eight introduced in Section 4 and measure the TMT, TDT, and downtime in various network scenarios. The results show that our BAC scheme shortens the TMT a lot, with comparable performance for the TDT and the downtime, compared with the conventional compression approaches.

A. Experimental Setup

Our experimental environment consists of four machines, each with quad-core Xeon X3220 2.4GHz CPU, 8GB RAM and two Intel PRO/1000 gigabit network interface card (NIC). Two of these machines act as the source and target nodes of live migration, and they share the storage from another machine through the NFS method. The last machine acts as the client of the web server workload. All of the four machines are connected via a gigabit ethernet switch. Besides, a separate gigabit ethernet exists between the source and target nodes and it's used only for the migration process. The OS of the host machines is Redhat Enterprise Linux 6.2, and the guest OS inside the VM is Ubuntu-12.04. The VM is configured with 1 VCPU and 1GB of RAM except where noted otherwise.

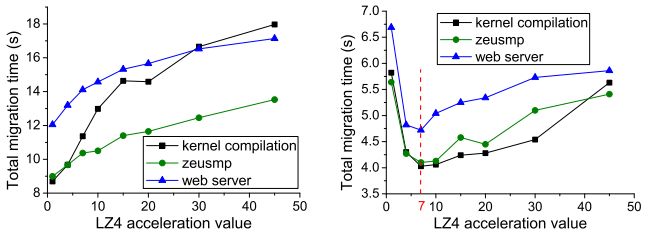
We repeat each migration experiment three times, and use the arithmetic average of the three values as the result. Besides, to ensure that the migration experiments are conducted in the same situation for each workload, we first boot up the VM and run the workload from the same starting point, and then begin migration after the workload has run for the same period of time.

To set the various network scenarios during VM migration, the *migrate_set_speed* command, which sets the bandwidth allocated to migration process, is used in QEMU monitor.

B. Effects of LZ4 Compression with Varying Acceleration Values

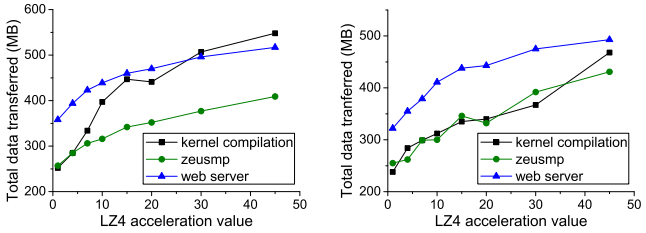
We first conduct the experiments to get the effects of LZ4 compression with varying acceleration values. Eight acceleration values (1, 4, 7, 10, 15, 20, 30, 45) are used to repeat the migration experiments. With these eight candidates, we get the varying trends of the TMT and the TDT when using various LZ4 approaches. The experiments are repeated with 30MB/s and 90MB/s network bandwidth corresponding to Figure 5.

Total Migration Time. Figure 8 shows the results of the TMT. With 30MB/s bandwidth, the TMT of all the three workloads becomes longer with the increment of the acceleration value. With 90MB/s bandwidth, the TMT first drops to the bottom from LZ4_1 to LZ4_7, and then climbs up all the way to LZ4_45. Compared with Figure 5, the varying trends of the TMT for these workloads conform to that of the S_{mgt}. When the S_{mgt} gets biggest (LZ4_1 and LZ4_7 respectively for the 30MB/s and 90MB/s bandwidth), the TMT gets shortest. So we conclude that our method to compute the S_{mgt} and to use it as an indicator of the performance improvements earned by the compression is efficient.



(a) 30MB/s bandwidth (b) 90MB/s bandwidth

Fig. 8. The total migration time of various LZ4 compression approaches with different network bandwidth.



(a) 30MB/s bandwidth (b) 90MB/s bandwidth

Fig. 9. The total data transferred of various LZ4 compression approaches with different network bandwidth.

Besides, the results of different LZ4 approaches differ a lot (but they are all better than that of the original pre-copy). Take the kernel compilation with 30MB/s network for instance, the TMT is 8.7s using LZ4_1 and 18.0s using LZ4_45, which has more than 2X difference. This indicates that it's significant to choose the proper compression approach for migration to obtain the maximum benefits.

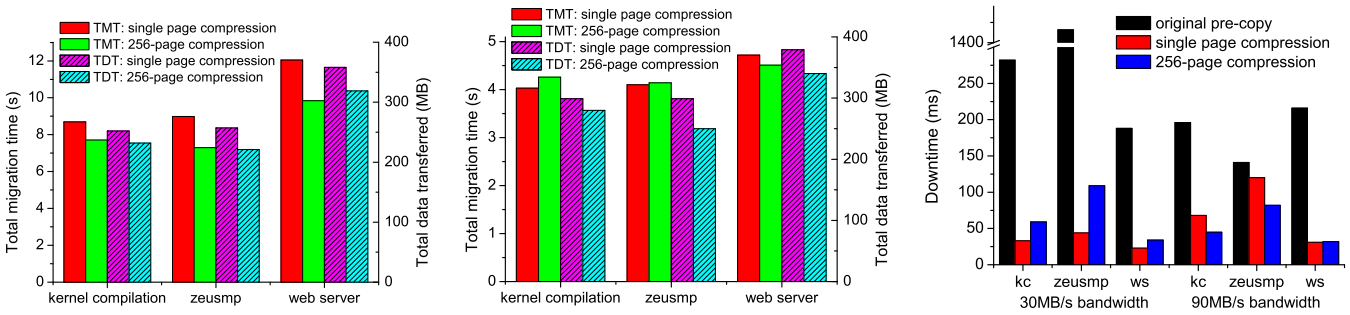
Total Data Transferred. Figure 9 shows the results of the TDT. With both network bandwidths, the TDT increases continuously from LZ4_1 to LZ4_45. The results suggest that the TDT is dominated by the compression ratio, and better compression ratio leads to less data transferred.

Comparing LZ4_1 with LZ4_7 in Figure 8(b) and Figure 9(b), we see that less data is transferred in more time with 90MB/s bandwidth. This is because although LZ4_1 leads to less data transferred due to its better compression ratio, its compression speed is much slower than LZ4_7. With 90MB/s bandwidth, it is the slow compression speed of LZ4_1, rather than the network bandwidth, that becomes the bottleneck of migration, so it leads to a longer migration time than LZ4_7.

C. Effects of Multi-Page Compression

Next, we conduct the experiments to verify the effect of multi-page compression. LZ4 approaches with the best acceleration values are used as baseline, which are LZ4_1 for 30MB/s bandwidth and LZ4_7 for 90MB/s bandwidth. A 1MB buffer is used to hold the memory pages to be compressed.

Figures 10(a) and 10(b) show the results of the TMT and the TDT. With 30MB/s bandwidth, compared with the single



(a) TMT and TDT with 30MB/s bandwidth (b) TMT and TDT with 90MB/s bandwidth (c) Downtime

Fig. 10. The effects of multi-page compression on the migration performance.

page compression, our multi-page compression method gets a speedup of 1.13, 1.23 and 1.22 respectively for the three workloads in terms of the TMT. TDT is reduced by 8%, 14%, and 11% respectively. With 90MB/s bandwidth, although the multi-page compression doesn't obtain obvious improvement for the TMT, it reduces the TDT by 6%, 16%, and 10% respectively. Figure 10(c) shows the downtime, and multi-page compression leads to comparable downtime compared with single page compression. Note that the zeusmp workload has such a fast memory dirtying rate that when the bandwidth is 30MB/s, the original pre-copy process is not able to converge to the *stop and copy* phase. To handle that, when the original pre-copy migration has continued for 60 seconds, we set the downtime value to be big enough through the QEMU monitor, to force the pre-copy process stepping into the *stop and copy* phase. This results in a huge downtime (1.46s).

In summary, the multi-page compression method improves both the TMT and the TDT with low bandwidth, while only improving the TDT with high bandwidth.

D. Migration with Fixed Network Bandwidth

In this and the next subsections, we compare the performance of BAC with four fixed LZ4 approaches (LZ4_1, LZ4_7, LZ4_15, LZ4_30) across various network scenarios. Besides, the performance of original pre-copy is also shown in the evaluation. First we conduct experiments with fixed network bandwidth, which means the bandwidth allocated to migration process stays constant during the whole migration process. As stated in Section III, 30MB/s and 90MB/s bandwidths are chosen to represent the busy network and idle network situations inside a data center.

Figure 11 shows the results with 30MB/s bandwidth. According to the *compression strategies*, BAC chooses LZ4_1 for migration. It is shown that BAC along with all the other compression approaches obtain remarkable improvements from the original pre-copy for all the three metrics. Furthermore, due to the use of bigger compression window (256 pages), BAC is better than the LZ4_1 approach. Besides, compared with the other LZ4 approaches, BAC gets significant improvements in terms of the TMT and the TDT. Note that when compared with the original pre-copy migration, almost all of the compression approaches reduce the downtime significantly from hundreds of milliseconds to tens of milliseconds. Tens of milliseconds is already in a micro scope, hence the little difference of downtimes among different compression approaches is not important for the performance of migration.

Figure 12 shows the results with 90MB/s bandwidth. In this case, BAC chooses LZ4_7 for migration. Again, all the compression approaches get remarkable improvements from the original pre-copy. Besides, BAC is the best in all these compression approaches considering both TMT and TDT. Compared with LZ4_1, BAC gets a speedup of 1.37X, 1.36X and 1.48X respectively in terms of the TMT, while has a comparable performance for the TDT.

In summary, in the network scenarios with fixed bandwidth, our BAC scheme gets significant improvement for the TMT, compared with the conventional compression approaches, while having comparable performance for the TDT and the downtime.

E. Migration with Fluctuating Network Bandwidth

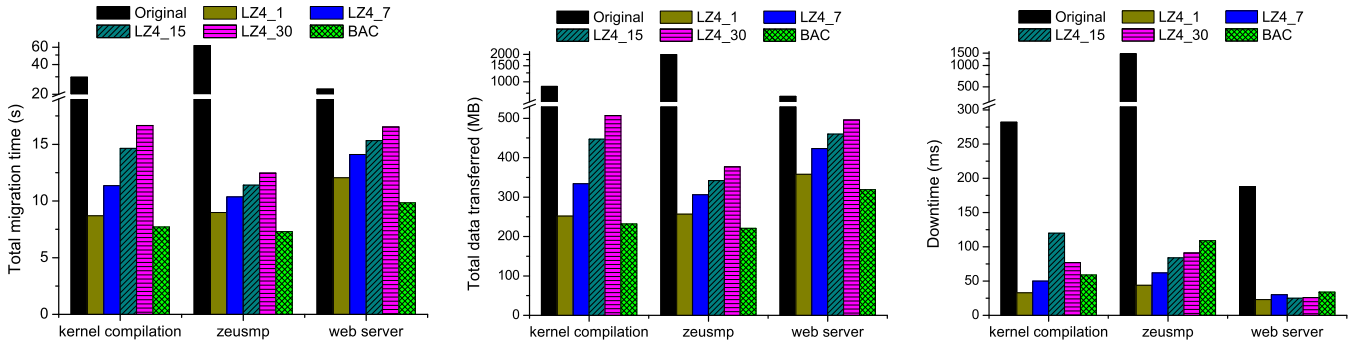
Then, we conduct experiments in a network with fluctuating bandwidth. The *migrate_set_speed* command is used in QEMU monitor to change the bandwidth allocated to migration every 1 second to emulate this network scenario. The bandwidth is alternately set as 30MB/s, 60MB/s, 90MB/s, 30MB/s, 60MB/s, 90MB/s... Besides, we migrate VMs allocated with 4GB RAM, to prolong the migration processes, so that the implication of the fluctuating bandwidth is better observed.

As shown in Figure 13, all the compression approaches get remarkable improvements from the original pre-copy, except for the downtimes of zeusmp. For zeusmp, the downtimes of several compression approaches are much longer than that of the original pre-copy. We explain this abnormality as follows. In the implementation of QEMU for the pre-copy process, at the end of each iteration, the code computes the data transfer rate of last iteration and gets the amount of remaining dirty pages. These two values are used to estimate the downtime, to decide if pre-copy should step into the *stop and copy* phase. However, the fluctuating bandwidth may lead to the result that the transfer rate computed by the code differs a lot from the real-time bandwidth. Besides, as mentioned before, zeusmp has a very fast memory dirtying rate. We consider these factors as the reasons for the abnormality of the downtimes for zeusmp.

Still, BAC obtains the best performance among all of the compression approaches considering both TMT and TDT. Compared with LZ4_1, in terms of the TMT, BAC gets a speedup of 1.26X, 1.23X and 1.31X respectively for the three workloads, while having comparable performance for the TDT and the downtime.

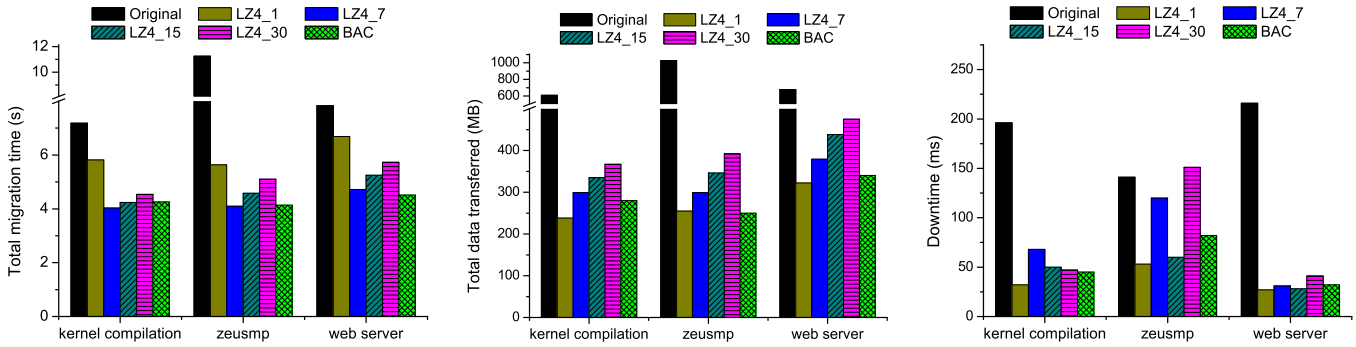
F. Overhead Analysis

Our BAC scheme has little memory overhead, just two buffers for multi-page compression and decompression (both



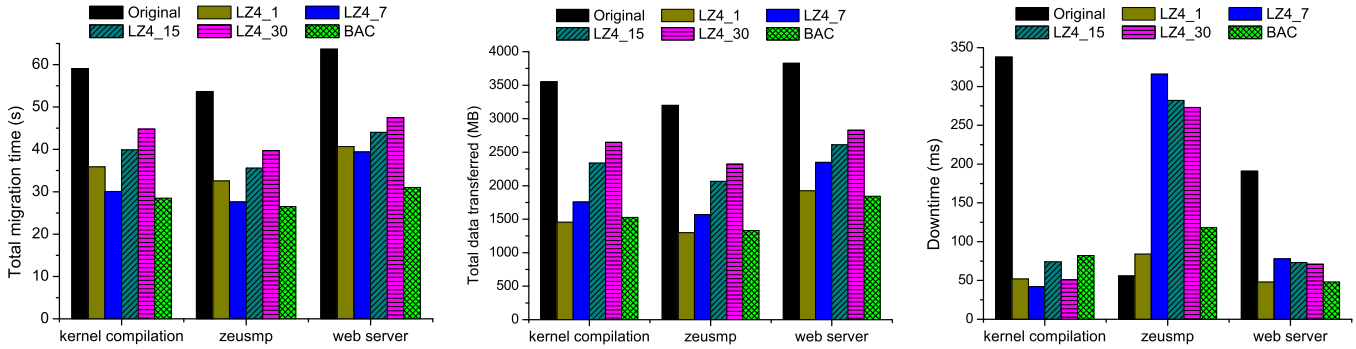
(a) Total migration time (b) Total data transferred (c) Downtime

Fig. 11. Migration performance with low network bandwidth (30MB/s).



(a) Total migration time (b) Total data transferred (c) Downtime

Fig. 12. Migration performance with high network bandwidth (90MB/s).



(a) Total migration time (b) Total data transferred (c) Downtime

Fig. 13. Migration performance with fluctuating network bandwidth. VMs are allocated with 4GB RAM.

are 1MB in our experiments) and the memory space used by LZ4 algorithm (typically about 16KB by its default setting). The main overhead is the CPU resources used for LZ4 compression, while decompression is very fast and uses little CPU resources. To evaluate the CPU overhead of LZ4 compression, we repeat the kernel compilation migration experiments with 30MB/s network bandwidth, using different LZ4 approaches. We get the precise CPU time spent on compression and the total migration time through the experiments. Then we compute the CPU overhead of LZ4 compression which is the compression time divided by the total migration time. Table III shows the results. Note that we use only one thread to execute the compression task, so the numbers represent the usage of a single CPU core. It is shown that from LZ4_1 to LZ4_19, the CPU overhead varies from 56.2% to 11.7% of a single CPU core. With the availability of multi-core or many-core, CPU resources tends to be abundant in current data centers, so we consider the CPU overhead of our BAC scheme as acceptable.

VII. Conclusion and Future Work

In this paper, we design and implement a Bandwidth-Aware Compression (BAC) scheme for live VM migration. BAC chooses the suitable compression approach according to the network bandwidth available for the migration process and employs multi-page compression. These features make BAC obtain more performance improvements for migration. To choose among various compression approaches, we get a *list of compression strategies* through the analysis of memory

TABLE III. CPU OVERHEAD OF A SINGLE CORE FOR DIFFERENT LZ4 APPROACHES

	LZ4_1	LZ4_3	LZ4_5	LZ4_7	LZ4_9	LZ4_11	LZ4_13	LZ4_15	LZ4_17	LZ4_19
CPU overhead	56.2%	43.7%	31.3%	22.6%	18.3%	15.2%	13.7%	12.8%	12.4%	11.7%

contents of VMs running various workloads. We demonstrate that this *list* applies to all the VM workloads, which makes our BAC scheme practical for implementation. Experiments with various network scenarios (low bandwidth, high bandwidth, and fluctuating bandwidth) demonstrate that compared with the conventional compression approaches, BAC shortens the TMT, while achieving the comparable performance for the TDT and the downtime.

For the future work, we expect to adopt more compression algorithms besides LZ4 into our BAC scheme. Besides, we plan to evaluate our BAC scheme in more network scenarios to represent the real-world situations inside data centers.

ACKNOWLEDGMENT

This work was partly supported by the National Key Research and Development Program of China under Grant 2016YFB1000202; State Key Laboratory of Computer Architecture, No.CARCH201505; NSFC No. 61502190 and No. 61502191; Fundamental Research Funds for the Central Universities, HUST, under Grant No. 2015MS07; Hubei Provincial Natural Science Foundation of China under Grant No. 2016CFB226. This work was also supported by Engineering Research Center of Data Storage Systems and Technology, Ministry of Education, China.

REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*. ACM, 2003, pp. 164–177.

[2] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (NSDI'05)*, 2005, pp. 273–286.

[3] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys'07)*, 2007, pp. 289–302.

[4] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07)*, 2007, pp. 119–128.

[5] Z. Zheng, M. Li, X. Xiao, and J. Wang, "Coordinated resource provisioning and maintenance scheduling in cloud data centers," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'13)*, 2013, pp. 345–349.

[6] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'10)*, 2010, pp. 1–9.

[7] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Almost optimal virtual machine placement for traffic intense data centers," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'13)*, 2013, pp. 355–359.

[8] M. Alicherry and T. Lakshman, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'13)*, 2013, pp. 647–655.

[9] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: High availability via asynchronous virtual machine replication," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*, 2008.

[10] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for hpc with xen virtualization," in *Proceedings of the 21st annual International Conference on Supercomputing (ICS'07)*. ACM, 2007, pp. 23–32.

[11] R. Nathuji and K. Schwan, "Virtualpower: coordinated power management in virtualized enterprise systems," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP'07)*, 2007.

[12] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Efficient datacenter resource utilization through cloud resource overcommitment," in *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2015, pp. 330–335.

[13] M. Nelson, B.-H. Lim, G. Hutchins *et al.*, "Fast transparent migration for virtual machines." in *Proceedings of USENIX Annual Technical Conference (USENIX'05), general track*, 2005, pp. 391–394.

[14] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive memory compression," in *IEEE International Conference on Cluster Computing and Workshops (CLUSTER'09)*. IEEE, 2009, pp. 1–10.

[15] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "Vmflock: virtual machine co-migration for the cloud," in *Proceedings of the 20th international symposium on High Performance Distributed Computing (HPDC'11)*, 2011, pp. 159–170.

[16] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," in *Proceedings of the ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments (VEE'11)*. ACM, 2011.

[17] J. Zhang, F. Ren, and C. Lin, "Delay guaranteed live migration of virtual machines," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'14)*, 2014, pp. 574–582.

[18] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux symposium*, vol. 1, 2007, pp. 225–230.

[19] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *Proceedings of the ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments (VEE'09)*. ACM, 2009.

[20] Y. Abe, R. Geambasu, K. Joshi, and M. Satyanarayanan, "Urgent virtual machine eviction with enlightened post-copy," in *Proceedings of the ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments (VEE'16)*. ACM, 2016.

[21] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, "Live migration of virtual machine based on full system trace and replay," in *Proceedings of the 18th ACM international symposium on High Performance Distributed Computing (HPDC'09)*. ACM, 2009, pp. 101–110.

[22] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines," in *Proceedings of the 20th international symposium on High Performance Distributed Computing (HPDC'11)*. ACM, 2011.

[23] "LZ4-Extremely fast compression," https://github.com/Cyan4973/lz4, accesed February 2016.

[24] "The QuickLZ compression library," http://www.quicklz.com/, accesed February 2016.

[25] X. Lin, G. Lu, F. Douglis, P. Shilane, and G. Wallace, "Migratory compression: Coarse-grained data reordering to improve compressibility," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST'14)*, 2014, pp. 256–273.

[26] F. Bellard, "Qemu, a fast and portable dynamic translator." in *Proceedings of USENIX Annual Technical Conference (USENIX'05)*, 2005.