ARTICLE IN PRESS

Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

MECOM: Live migration of virtual machines by adaptively compressing memory pages☆

Hai Jin [a], Li Deng [a,b], Song Wu [a,*], Xuanhua Shi [a], Hanhua Chen [a], Xiaodong Pan [c]

[a] Cluster and Grid Computing Lab, Services Computing Technology and System Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China

[b] School of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan, 430065, China

[c] Baofeng Network Technology Co., Ltd., Beijing, 100191, China

HIGHLIGHTS

- To the best of our knowledge, we are the first to exploit memory compression technique to minimize the downtime and total migration time of live migration.
- By analyzing memory page characteristics, we implement an adaptive memory compression approach to tune migration performance of virtual machines. A zero-aware characteristics-based compression (CBC) algorithm is designed for live VM migration.
- Memory compression is adaptive to network bandwidth, then the performance of live VM migration still keeps good in diverse network environments with different available network bandwidth.
- Experiment results demonstrate that compared with Xen, our approach can significantly reduce downtime, total migration time, and total transferred data by 27.1%, 32% and 68.8% respectively.

ARTICLE INFO

Article history:
Received 4 July 2012
Received in revised form
25 February 2013
Accepted 26 September 2013
Available online xxxx

Keywords:
Virtual machine
Live migration
Compression algorithm
Pre-copy

ABSTRACT

Live migration of virtual machines has been a powerful tool to facilitate system maintenance, load balancing, fault tolerance, and power-saving, especially in clusters or data centers. Although pre-copy is extensively used to migrate memory data of virtual machines, it cannot provide quick migration with low network overhead but leads to large performance degradation of virtual machine services due to the great amount of transferred data during migration. To solve the problem, this paper presents the design and implementation of a novel memory-compression-based VM migration approach (MECOM for short) that uses memory compression to provide fast, stable virtual machine migration, while guaranteeing the virtual machine services to be slightly affected. Based on memory page characteristics, we design an adaptive zero-aware compression algorithm for balancing the performance and the cost of virtual machine migration. Using the proposed scheme pages are rapidly compressed in batches on the source and exactly recovered on the target. Experimental results demonstrate that compared with Xen, our system can significantly reduce downtime, total migration time, and total transferred data by 27.1%, 32%, and 68.8% respectively.

1. Introduction

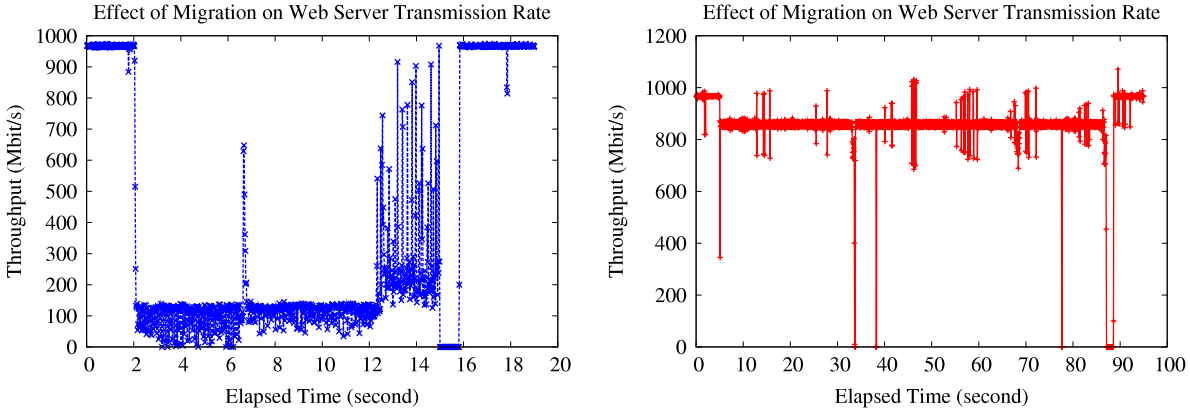
Virtual machines (VMs) [1] not only provide efficient and secure computing resource containers, but also can be effectively deployed based on applications' demands [2]. VMs can be migrated smoothly among multiple physical machines. Live migration of VMs [3–6] has been a strong management tool in virtualized environments. It facilitates system maintenance [7], load balancing [8], fault tolerance [9], and power management [10–13].

In a VM-based cluster, multiple virtual machines share a physical resource pool. Due to dynamical workloads, some nodes are under-utilized, while others become over-loaded. Virtual machine migration strategies [8,14] have been proposed to balance VM loads among physical nodes. In addition, a proactive fault tolerance scheme [9] migrates VMs from unhealthy nodes to healthy ones. With the advent of multi-core or many-core processors, power management has been a critical and necessary component of computing systems. Live migration of virtual machines provides flexible approaches to power-efficiency [10–12]. In various application

☆ The paper is based on a preliminary work published in Proceedings of Cluster'09.

* Corresponding author. Tel.: +86 27 87543529.
E-mail address: wusong@hust.edu.cn (S. Wu).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.09.031



(a) In non-adaptive rate limiting mode. (b) In adaptive rate limiting mode.

Fig. 1. Influence of pre-copy in Xen on web service.

scenarios, VM migration is expected to be fast and VM service degradation is also expected to be low during migration.

Live migration is a key feature of system virtualization technologies. It is fast and transparently transports a running VM from one host machine to another to keep the service continuous. Transferred data includes memory information and disk part. In this paper, we focus on VM migration within a LAN environment where a network-accessible storage system (such as SAN or NAS) is generally employed [3]. Only memory and CPU status needs to be transferred from the source node to the target one. Existing Live migration techniques mainly use a pre-copy approach [3,4], which first transfers all memory pages and then copies pages just modified during the last round iteratively. VM service downtime is expected to be minimized by iterative copy operations. When applications' writable working set becomes small, the virtual machine is suspended and only CPU state and dirty pages in the last round are sent out to the destination. Moreover, the maximum number of iterations must be set because not all applications' dirty pages are ensured to converge to a small writable working set over multiple rounds [3].

However, great service degradation would happen in pre-copy phase because migration daemon continually consumes network bandwidth to transfer dirty pages in each round. A VM is configured with 4 VCPUs and 1024 MB RAM to provide Apache service. When we migrate the VM lively between two nodes, *ab* (Apache benchmark) is used on another node as a client to access 512 kB static web pages. The number of concurrent accesses is set as 100. The total number of requests is 50 million. The web throughput is recorded in Fig. 1(a). From the Fig. 1(a), we can observe that, the whole migration time of VM is about 14 s. During the migration process, web throughput rate drops down from about 1000 to 100 Mb/s. And the throughput comes back to 1000 Mb/s as soon as the migration ends. The reason why great service degradation happens is that the migration of VM would consume much network bandwidth. In the above test, 1050.352 MB data have been transferred during the whole migration process. An adaptive rate limiting approach [3] is employed to limit the largest network bandwidth available for VM migration. We can see the effects of the adaptive rate limiting approach on VM migration in Fig. 1(b). From the Fig. 1(b), we can observe that although there is slight service degradation in VM, total migration time is prolonged nearly by nine times.

In fact, the above issues in pre-copy approach are caused by the significant amount of transferred data during the whole migration process. A checkpointing/recovery and trace/replay approach (CR/TR-Motion) is proposed [15] to provide fast VM migration by transferring execution trace file in iterations rather than dirty pages. Apparently, the total size of all log files is much less than that of dirty pages. So, total migration time and downtime of migration are reduced. However, CR/TR-Motion is valid only when the log replay rate is larger than the log growth rate. The inequality of computing ability between the source and the target nodes limits application scope of live VM migration in clusters. Another post-copy strategy was also introduced into live migration of virtual machines [16]. In this approach, all memory pages are transferred only once during the whole migration process and the baseline total migration time is achieved. However, the downtime is much higher than that of the pre-copy scheme due to the extra latency of fetching pages from the source node before VM can be resumed on the target.

With the advent of multi-core or many-core processors, there are abundant CPU resources available. Even if several VMs reside on the same multi-core machine, spare CPU resource is still available because physical CPUs are frequently amenable to multiplexing [17]. We can exploit these copious CPU resources to compress page frames, and thus the amount of transferred data can be effectively reduced. Moreover, memory compression algorithms typically have little memory overhead, while decompression requires no memory [18].

Based on the above observation, this paper presents a novel approach to optimize live virtual machine migration based on pre-copy technique. We use memory compression to provide fast VM migration. Virtual machine migration performance is greatly improved by cutting down the amount of transferred data. Furthermore, we design a zero-aware *characteristics-based compression* (CBC) algorithm for live migration. To further reduce the overhead introduced by multiple rounds compression operations, we use multi-threading techniques to parallelize compression tasks. We implement a prototype system MECOM based on Xen 3.1.0.

The main contributions of the paper are two-fold:

(1) To the best of our knowledge, we are the first to exploit a memory compression technique to minimize the downtime and total migration time of live migration.
(2) By analyzing memory page characteristics, we implement an adaptive memory compression approach to tune the migration performance of virtual machines.

The rest of the paper is organized as follows. We describe the background of live virtual machine migration and pre-copy approach in Section 2. In Section 3 we present the design of a characteristics-based compression algorithm for live migration. Our adaptive page compression method is depicted in Section 4. Section 5 describes the implementation of system MECOM in detail. In Section 6 we describe our evaluation methodology and present the experimental results. We discuss related work in VM migration and memory compression in Section 7. Finally, we summarize our contributions and outline our future work in Section 8.

Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031

2. Live VM migration and pre-copy approach

Because the virtualization layer decouples the inter-dependence between virtual machines and the underlying hardware, the virtual machine that encapsulates the whole application environment can be migrated freely and smoothly among different physical nodes, while the virtual machine service remains uninterrupted.

When a virtual machine is migrated from one host machine to another, the mapping between real hardware and virtual hardware changes. The change should be correctly reflected in the virtual machine's state before the virtual machine resumes on the destination host. There are three main kinds of state that need to be dealt with when migrating a VM: (1) the VM's physical memory; (2) the virtual device state including the state of the CPU, the motherboard, networking and storage adapters, and etc.; (3) external connections with devices such as networking, SCSI storage devices, USB devices. Since live migration of virtual machine mostly happens in the environment with shared network-accessible storage system, VM's physical memory is usually the largest part and the most mutable during the migration among the three kinds of state [3]. The transferring of physical memory data dominates the performance of live VM migration.

The performance of a virtual machine migration can be measured by two metrics: *total migration time*—the period from the beginning of pre-migration work to the end of all migration procedure and *downtime* during which virtual machine services are entirely unavailable.

Currently, a pre-copy scheme which is first proposed to migrate the address space of processes [19] is widely used to transfer physical memory during live migration of virtual machines. Fig. 2 depicts the pre-copy process. To shorten downtime, while virtual machine still runs on the source node, pre-copy is executed as an initial copy of the whole memory image followed by repeated copies of the pages modified during the previous copy from the source host to the destination machine until the number of modified pages is relatively small or until there is no significant reduction in the number of modified pages. The remaining modified pages are recopied after the virtual machine is suspended.

In the implementation, the correctness and good performance of pre-copy approach should be ensured. Page-level protection hardware is employed to guarantee that a consistent snapshot is transferred. The pre-copy operation is executed at a high priority to avoid the interference from other programs. To balance the downtime of virtual machine migration and the impact of migration traffic on running services, a rate-adaptive algorithm [3] is used. A low network bandwidth limit for migration traffic avoids impacting the performance of running services. And the gradually increased bandwidth limit helps to reduce the downtime.

Based on the working-set analysis of typical workloads [3], some set of pages are updated extremely frequently. These pages are usually modified rapidly in adjacent iterations of pre-copy phase. Transferring the pages just wastes network bandwidth. So, only the pages that are dirtied in the previous round and not modified again at the time to be transferred are the candidates to copy in each round. The working-set analysis also demonstrates that some virtual machines are not amenable to pre-copy migration because of large numbers of dirty pages in each round. Then, the maximum round of pre-copying is set.

3. Algorithm design

In this section, we present the design and implementation of our compression algorithm. First, we describe the objective of this design in the context of a previous pre-copy scheme. Then, we analyze the conditions under which a compression algorithm brings benefits to live VM migration. Then, based on the analysis of memory characteristics in VMs respectively with typical workloads, we depict our CBC algorithm for VM migration. Finally, we analyze the benefits from live VM migration using page compression.

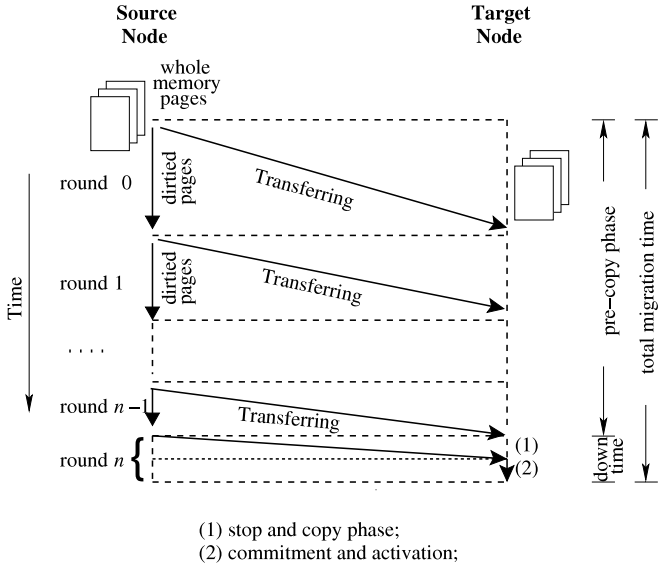


Fig. 2. Pre-copy phase.

3.1. Design objective

Compression technique can be used to significantly improve the performance of live migration. On one hand, compression indirectly augments network bandwidth for migration. Compressed data take less time to fly on the network. On the other hand, the number of dirty pages becomes much smaller due to a shortened elapsed time in each round of the *pre-copy* phase.

The compression algorithm is first lossless because the compressed data need to be exactly reconstructed. Second, the introduction of a compression algorithm should improve the performance of live VM migration. That is, the total migration time and downtime of VM migration should be shortened by using compression algorithm.

Although reduced data needs less time to fly on the network, compression operations themselves take time to construct the concise representation of data and decompression operations also take time to reconstruct the original data from a compressed format. The time benefit of shortened data transferring should be longer than that of the additional time cost brought by compression and decompression operations. The higher the compression ratio (the percentage of freed data) of an algorithm, the smaller the amount of data to be transferred, the shorter the duration and the fewer dirtied pages in each round of the pre-copy phase. However, an algorithm with a high compression ratio usually takes a relatively long time to find the regularities of data and construct a more concise representation, which leads to relatively slow compression. That is, an algorithm with a high compression ratio often takes a long time to execute compression and decompression operations. So, when we design a compression algorithm for live VM migration, the tradeoff between compression ratio and compression rate of an algorithm should be considered.

The following parts discuss the relationship of factors of the compression algorithm with formalized characterization in the context of live VM migration. The analysis would further direct the design of compression algorithm. What kind of compression algorithms would improve the performance of live VM migration? What characteristics do these algorithms have?

Fig. 3 depicts the process of our approach MECOM for live VM migration. In each round of pre-copy phase, dirty pages are sent out only after they are compressed. Also, some notations are defined in Table 1.

Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031

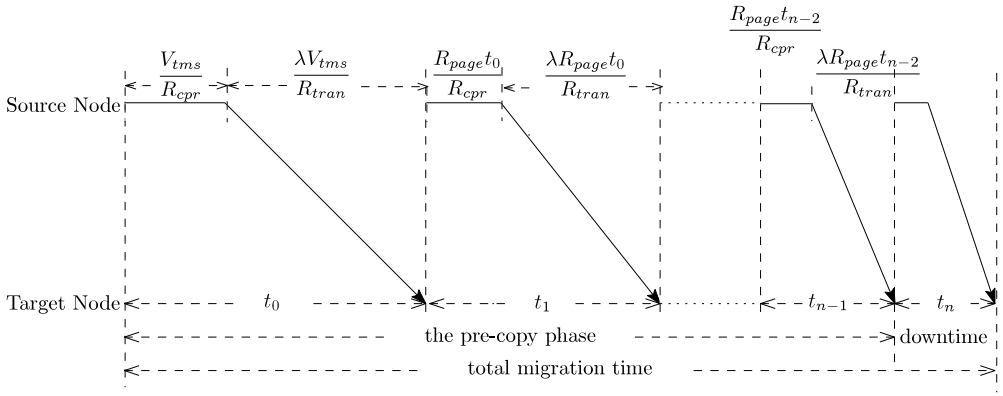


Fig. 3. Migration process with page compression.

Table 1
Symbols and definitions.

Symbol	Definition
V_{tms}	Total memory size of the migrated VM
V_{thd}	Threshold of dirty pages where pre-copy process is stopped
R_{page}	Average growth rate of dirty pages in the migrated VM
R_{tran}	Average page transfer rate—available network bandwidth for transferring pages
R_{cpr}	Average memory compression rate
ρ	Average compression ratio, the percentage of freed data

For our approach MECOM using page compression for VM migration, transferred data size and the elapsed time in all rounds are respectively represented by vector $V = \langle v_0, v_1, \ldots, v_{n-1}, v_n\rangle$ and vector $T = \langle t_0, t_1, \ldots, t_{n-1}, t_n\rangle$. For Xen, the corresponding sequences of transferred data size and the elapsed time are defined as vector $V' = \langle v'_0, v'_1, \ldots, v'_{m-1}, v'_m\rangle$, vector $T' = \langle t'_0, t'_1, \ldots, t'_{m-1}, t'_m\rangle$.

Let $\lambda = 1 - \rho$, then

$$t_0 = \frac{V_{\text{tms}}\lambda}{R_{\text{tran}}} + \frac{V_{\text{tms}}}{R_{\text{cpr}}}, \qquad t'_0 = \frac{V_{\text{tms}}}{R_{\text{tran}}}$$

$$t_1 = \frac{R_{\text{page}}\lambda t_0}{R_{\text{tran}}} + \frac{R_{\text{page}}t_0}{R_{\text{cpr}}}, \qquad t'_1 = \frac{R_{\text{page}}t'_0}{R_{\text{tran}}}$$

$$\ldots$$

$$t_n = \frac{R_{\text{page}}\lambda t_{n-1}}{R_{\text{tran}}} + \frac{R_{\text{page}}t_{n-1}}{R_{\text{cpr}}}, \qquad t'_m = \frac{R_{\text{page}}t'_{m-1}}{R_{\text{tran}}}. \tag{1}$$

And,

$$v_0 = \lambda V_{\text{tms}}, \qquad v'_0 = V_{\text{tms}}$$
$$v_1 = R_{\text{page}}\lambda t_0, \qquad v'_1 = R_{\text{page}}t'_0$$
$$\ldots$$
$$v_n = R_{\text{page}}\lambda t_{n-1}, \qquad v'_m = R_{\text{page}}t'_{m-1}.$$

We can get

$$t_n = V_{\text{tms}}R^n_{\text{page}}\left(\frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}}\right)^{n+1} = V_{\text{tms}}\left(\frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}}\right)\left(\frac{\lambda R_{\text{page}}}{R_{\text{tran}}} + \frac{R_{\text{page}}}{R_{\text{cpr}}}\right)^n, \quad n = 0, 1, 2, \ldots$$

$$v_n = \lambda V_{\text{tms}}(R_{\text{page}})^n\left(\frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}}\right)^n, \quad n = 0, 1, 2, \ldots$$

$$t'_m = V_{\text{tms}}\frac{R^m_{\text{page}}}{R^{m+1}_{\text{tran}}}, \quad m = 0, 1, 2, \ldots$$

$$v'_m = V_{\text{tms}}\left(\frac{R_{\text{page}}}{R_{\text{tran}}}\right)^m, \quad m = 0, 1, 2, \ldots. \tag{2}$$

After a compression algorithm is introduced, the performance of VM migration should be improved. Total migration time is shortened. Given the same amount M of data, MECOM should use less time to transfer them from source node to target node. Suppose that Xen spends time s' to transfer data M and MECOM spends time s, we have the following formulae:

$$s = \frac{M}{R_{\text{cpr}}} + \frac{(1-\rho)M}{R_{\text{tran}}}$$
$$s\prime = \frac{M}{R_{\text{tran}}}.$$

Because $s < s'$, we get

$$\rho > \frac{R_{\text{tran}}}{R_{\text{cpr}}}.$$

Let $\gamma = R_{\text{tran}}/R_{\text{cpr}}$, then

$$\rho > \gamma. \tag{3}$$

The formula (3) indicates that, if the compression ratio of an algorithm is larger than the ratio of network bandwidth available for migration to the compression rate of the algorithm, live VM migration can show better performance.

Let $\Delta s = s' - s$, then

$$\Delta s = \frac{M}{R_{\text{tran}}} - \left(\frac{M}{R_{\text{cpr}}} + \frac{(1-\rho)M}{R_{\text{tran}}}\right) = \frac{M}{R_{\text{tran}}}(\rho - \gamma).$$

So, the bigger the value $\tau = \rho - \gamma$, the larger Δs. That is, given the amount of transferred data M and a certain network bandwidth, a compression algorithm with bigger τ would bring more gains for live VM migration.

3.2. Memory data characteristic analysis

Data compression algorithms are designed typically based on the regularities of data. These algorithms exploit regularities in data to compress. For data to be compressed, compression algorithms whose expectations about the kinds of regularities meet the real regularities of data being compressed would work better.

Compression consists of two phases, which are typically interleaved in practice: modeling and encoding [20]. Modeling is the process of finding regularities that allow a more concise representation of information. Encoding is the construction of that more concise representation.

To a great extent, the performance of compression algorithms depends on the data being compressed. They must exploit data representation characteristics to achieve any compression. Paul R. Wilson deems that memory data representations have certain

Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031

Table 2
Memory page characteristics analysis.

WSR (%) zone	VM respectively with the following workloads						
	Null	gcc	Sort	Eclipse	Apache	MUMmer	dbench
0–25	0.448156	0.268028	0.450763	0.381282	0.409707	0.625361	0.446836
25–75	0.212847	0.192677	0.225925	0.400119	0.225957	0.215628	0.191420
75–100	0.338174	0.538479	0.321932	0.217200	0.362381	0.158034	0.361128

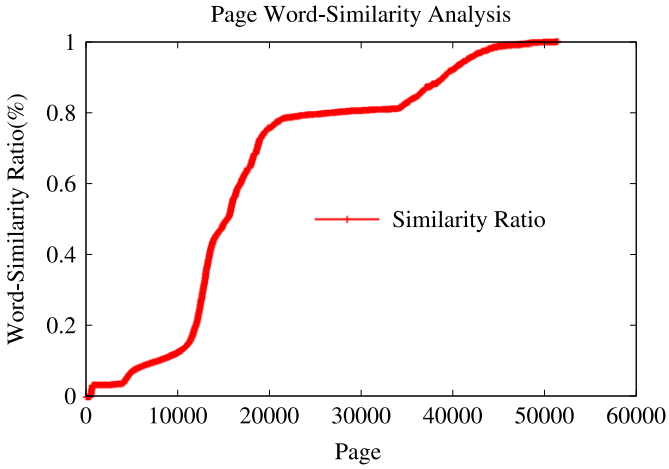


Fig. 4. Memory page word-similarity ratio distribution of a 256 MB-RAM virtual machine providing Apache service (pages are sorted by increasing word-similarity ratio).

strong regularities, although diverse applications might be loaded into the memory [21]. Moreover, there are a great number of zero bytes in the memory pages [22]. We first analyze the memory data representation of VMs.

A VM is typically configured with 256 MB RAM, in which an Apache 2.0.63 web server is running. Red Hat Enterprise Linux 5.0 is used as the guest operating system (OS). We analyze the word-similarity of each memory page with a 16-word dictionary. The dictionary keeps 16 words recently seen in a page. We count words which fully or partially match one word in the dictionary. The number of matched words in a page is called the word-similarity of the page. The word-similarity ratio is the percentage of word-similarity in a page. In order to find out the distribution characteristics of each page, we remove all zero pages and sort the remaining pages by increasing word-similarity ratio. The result is listed in Fig. 4.

From Fig. 4, we observe that most of memory pages polarize into completely opposite status: high word-similarity and low word-similarity. Then, we extend this test to VMs with other representative workloads from [23] and the test result is listed in Table 2. In Table 2, WSR means Word-Similarity Ratio (WSR) and the first workload *null* denotes a VM without any workload.

As Table 2 shows, a majority of memory pages have more than 75% similarity or less than 25% similarity. For high word-similarity pages, we can exploit a simple but very fast compression algorithm based on strong data regularities, which achieves win–win effects.

In addition, we observe that memory data contain a large proportion of zeros. For pages containing large numbers of zero-bytes, we can also design a simple but pretty fast algorithm to achieve high compression performance.

3.3. Characteristic-based compression (CBC) algorithm for live VM migration

The potential benefits of memory compression in live VM migration depend on the relationship between compression ratio and compression rate. Because an algorithm with a high compression ratio typically takes a long time to operate, long compression time subsequently prolongs migration time. If the compression ratio is not high enough to decrease the duration of each round, or if the compression time exceeds the saved data transfer time of each round, compression shows no benefit. Therefore, when we design a compression algorithm for VM migration, we should make trade-offs among the above factors.

If an algorithm itself embodies expectations about the kinds of regularities in the memory footprint, it must be very fast and effective. As we analyze in the above part, memory data in VMs with various workloads do not always show the same regularity. Then, a single compression method for all memory data finds it difficult to achieve the win–win status that we expect. Therefore, we should provide several compression approaches to pages with different kinds of regularities. For pages with strong regularities, we exploit simple but extremely fast algorithms.

Based on the analysis of memory data characteristics in the part 3.2, we first classify pages into the following kinds: (1) pages composed of a great many of zero bytes and sporadic nonzero bytes; (2) pages with high word-similarity; (3) pages with low word-similarity.

For the first kind of pages, we can scan the whole page and just record the information about the offset and value of nonzero bytes. For the second kind of pages, we can use methods that embody strong similarities, such as WKdm [21]. WKdm is a unique combination of dictionary and statistical techniques specifically designed to quickly and efficiently compress memory data. The last kind of pages has weak regularities, and then a universal approach with a high compression ratio is appropriate. LZO [18], a modern implementation of Lempel–Ziv compression, is an option.

Our compression algorithm CBC primarily includes two parts:

(1) Classifying the kind of pages being compressed. The intention of the step is to pick out pages with strong regularities, which mean high word-similarity or extremely high frequency of zero bytes. With the help of a dictionary, we count the similar words (complete or partial matching with the dictionary) and zero-bytes. The dictionary always keeps recently-seen words. In order to decrease the matching overhead, our algorithm manages this dictionary as a direct mapped cache and LRU (Least Recently Used) is used as the replacement algorithm for the dictionary.
(2) Making choice of appropriate concrete compression methods. A flag indicating the kind of a page is also added into the compressed output for future decompression.

The pseudo-code of our CBC algorithm is listed on the next page.

3.4. Benefits from CBC algorithm

In this part, we discuss the performance improvement of VM migration after the CBC algorithm is introduced.

Theorem 1. *For Xen, only if* $\varphi = \frac{R_{page}}{R_{tran}} < 1$, *the pre-copy approach would make the applications' WWS (Writable Work Set) monotonously decreasing.*

Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031

Algorithm 1 The pseudo-code of CBC algorithm

```
1:  Initialize a 16-word dictionary;
2:  for each page do
3:    zeroByte = 0;
4:    wordSimilarity = 0;
5:    for each word in page do
7:      for each byte in word do
8:        if byte is zero-byte then
9:          zeroByte++;
10:       end if
11:     end for
12:     if word is zero-word or word matches one in dictionary then
13:       wordSimilarity++;
14:     end if
15:   end for
16:   if zeroByte >= threOfZeroByte * totalNumOfByte then
17:     push 0 into compressedData;
18:     for each byte in page do
19:       if byte is not zero-byte then
20:         push (offset, value) into compressedData;
21:       end if
22:     end do
24:   else if wordSimilarity >= threshold * totalNumOfWord then
25:     push 1 into compressedData;
26:     compress page with compression method for high word-
          Similarity and push the result into compressedData;
27:   else
28:     push 2 into compressedData;
29:     compress page with compression method for low word-
          Similarity and push the result into compressedData;
30:   end if
31: end for
32: return compressedData;
```

Proof. Since

$$v'_m = V_{\text{tms}} \left(\frac{R_{\text{page}}}{R_{\text{tran}}} \right)^m, \quad m = 0, 1, 2, \ldots$$

v'_m $(m = 0, 1, 2, \ldots)$ is a geometric progression with a ratio of $R_{\text{page}}/R_{\text{tran}}$. When $R_{\text{page}}/R_{\text{tran}} < 1$, v'_m would be smaller and smaller with the increasing of m.

So, when $\varphi = \frac{R_{\text{page}}}{R_{\text{tran}}} < 1$, the pre-copy approach would make the applications' WWS (Writable Work Set) smaller and smaller. Thus proved. □

Theorem 2. *If $\rho R_{\text{cpr}} > R_{\text{tran}}$, then the following statements are true:*

$$t_i < t'_i, \qquad v_i < v'_i, \quad i = 0, 1, 2, \ldots.$$

Proof. We use mathematical induction.

Basis: Show that the statements hold for $i = 0$.

If $i = 0$,

$$t_0 = \frac{V_{\text{tms}}\lambda}{R_{\text{tran}}} + \frac{V_{\text{tms}}}{R_{\text{cpr}}}, \qquad t'_0 = \frac{V_{\text{tms}}}{R_{\text{tran}}}.$$

Because $\rho R_{\text{cpr}} > R_{\text{tran}}$, apparently we can get

$$t_0 < t'_0.$$

Since $\lambda < 1$, then we can also get

$$v_0 = \lambda V_{\text{tms}} < v'_0 = V_{\text{tms}}.$$

So the statements are true for $i = 0$.

Inductive step: Show that if the statements are true for $i = n$, then also the statements are true for $i = n + 1$.

We have:

$$t_n < t'_n, \qquad v_n < v'_n.$$

That is,

$$V_{\text{tms}} R^n_{\text{page}} \left(\frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}} \right)^{n+1} < V_{\text{tms}} \frac{R^n_{\text{page}}}{R^{n+1}_{\text{tran}}} \tag{4}$$

$$\lambda V_{\text{tms}} (R_{\text{page}})^n \left(\frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}} \right)^n < V_{\text{tms}} \left(\frac{R_{\text{page}}}{R_{\text{tran}}} \right)^n. \tag{5}$$

Since $\rho R_{\text{cpr}} > R_{\text{tran}}$, that is

$$0 < \frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}} < \frac{1}{R_{\text{tran}}}. \tag{6}$$

From formula (4) and (6), we can get

$$V_{\text{tms}} R^n_{\text{page}} \left(\frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}} \right)^{n+2} < V_{\text{tms}} \frac{R^n_{\text{page}}}{R^{n+2}_{\text{tran}}}$$

$$V_{\text{tms}} R^{n+1}_{\text{page}} \left(\frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}} \right)^{n+2} < V_{\text{tms}} \frac{R^{n+1}_{\text{page}}}{R^{n+2}_{\text{tran}}}$$

$$t_{n+1} < t'_{n+1}.$$

From formula (5) and (6), we can also get

$$\lambda V_{\text{tms}} (R_{\text{page}})^{n+1} \left(\frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}} \right)^{n+1} < V_{\text{tms}} \left(\frac{R_{\text{page}}}{R_{\text{tran}}} \right)^{n+1}$$

$$v_{n+1} < v'_{n+1}.$$

Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031

Thus, the statements are true for $n+1$.

Since both the *basis* and the *inductive step* have been proved, it has now been proved by mathematical induction that the statements hold for all natural n. Thus proved. □

Let $\varphi = \frac{R_{\text{page}}}{R_{\text{tran}}}$, there are two cases listed below:

(1) $\varphi < 1$, which means that the dirty pages rate is less than the network transfer rate. According to Theorem 1, applications' WWS decreases monotonously in the pre-copy phase. In this case the pre-copy process would be terminated when the amount of dirty page data in a certain iteration is equal to or less than the threshold V_{thd}.

As shown in formulas (2) of part 3.1,

$$t_n = V_{\text{tms}} R_{\text{page}}^n \left(\frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}}\right)^{n+1} \tag{7}$$

$$t'_m = \frac{V_{\text{tms}} R_{\text{page}}^m}{R_{\text{tran}}^{m+1}}. \tag{8}$$

If our approach MECOM using compression runs n rounds to reach the threshold and Xen runs m rounds in the pre-copy phase, we can get the equation: $t_n = t'_m$. From Eqs. (7) and (8), we can deduce the equation

$$R_{\text{page}}^n \left(\frac{\lambda}{R_{\text{tran}}} + \frac{1}{R_{\text{cpr}}}\right)^{n+1} = \frac{R_{\text{page}}^m}{R_{\text{tran}}^{m+1}}. \tag{9}$$

From formula (3) in part 3.1 and Eq. (9), we get $\varphi^m < \varphi^n$. Because $\varphi < 1$, we can deduce $m > n$. The inequation implies that when the dirty page rate is less than the available network bandwidth for migration, our approach MECOM has a faster convergence rate to reach the close point of pre-copy iterations than Xen.

According to Theorem 2, for each round i,

$$t_i < t'_i, \qquad v_i < v'_i.$$

Downtime, total migration time, and total transferred data in our approach would be less than that of Xen respectively.

(2) $\varphi \geq 1$. In this case, writable working sets of virtual machines would not get smaller with the time elapsed even when available network bandwidth for migration reaches the maximum. So, the pre-copy process is terminated as soon as the maximum transfer rate is available, which ensures that the downtime is not prolonged.

Provided that transfer rate R_{tran} is constant R_{tran0}, the pre-copy algorithm becomes ineffective when the dirty page rate R_{page} equals R_{tran0} in Xen. But after memory compression is introduced, the network transfer rate is indirectly augmented almost by a factor $1/(1-\rho)$ $(0 < \rho < 1)$. Only when the actual dirty page rate reaches about $R_{\text{tran0}}/(1-\rho)$, the pre-copy process is terminated in the system MECOM. That is, MECOM allows a higher dirty page rate than Xen. Furthermore, even if the real dirty page rate exceeds $R_{\text{tran0}}/(1-\rho)$ in our approach, downtime is greatly shortened due to the shrunken amount of transferred data.

4. Adaptive page compression

As shown in Section 3.1, for a certain application, when the relationship among transfer rate, compression rate and compression ratio meet formula (3), page compression would improve the performance of live VM migration. Because the live migration of VM may happen in diverse network environments with different available network bandwidth, we should dynamically adjust the compression rate and the compression ratio of the compression algorithm to meet formula (3), which assures that the performance of live VM migration is improved.

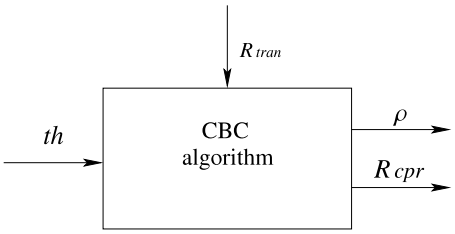


Fig. 5. An input–output model for adaptive compression mode.

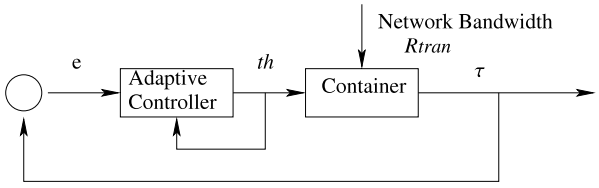


Fig. 6. Adaptive feedback-driven compression performance controller.

To adjust the performance of the compression algorithm, we tune the threshold between high word-similarity ratio (WSR) and low word-similarity ratio in the CBC algorithm. In a low network environment where bandwidth resource is limited, we make more pages compressed by the slow compression method with relatively high compression ratio. On the contrary, when available network bandwidth is high, more pages are compressed by a relatively fast compression method. Less compression time is hoped for.

We employ the control theory to address the problem of dynamically controlling the performance of the page compression algorithm during live migration of VMs. Generally, the object to be controlled is represented as an input–output model in control theory. The inputs are the control knobs and the outputs are the metrics being controlled [14].

Fig. 5 depicts an input–output representation of the object to be controlled. The input to the system is the threshold *th* between high word-similarity ratio and low word-similarity ratio. The outputs include average compression ratio ρ and average compression rate R_{cpr} of the CBC algorithm. The available network bandwidth for live VM migration R_{tran} is a disturbance factor to the model. It affects the outputs but is not directly under control. When R_{tran} varies, the input–output relationship changes accordingly, which increases the difficulty to control the objects.

To control the compression performance of the CBC algorithm in real time under various environments, we design a feedback-driven control approach which varies the inputs in the operating region and observes the corresponding outputs. The controller tunes the parameters of the CBC algorithm and measures the corresponding compression performance in a certain period of time. The controller is designed to adaptively adjust to a varying network status so that the performance of live VM migration stays good with the help of the page compression technique.

The block diagram for the controller is shown in Fig. 6. At the beginning of the control interval k, the controller takes the measured value during the previous intervals $(\tau(k-2), \tau(k-1))$ as inputs. The controller computes the tracking error (e) as $e(k-1) = \tau(k-1) - \tau(k-2)$, and decides the threshold *th* between high WSR and low WSR of CBC algorithm for the next interval. The new threshold *th* is calculated using the following control law:

$$th(k) = \begin{cases} 2 * th(k-1) - th(k-2) & \text{if } e(k-1) \geq 0, \\ th(k-2) & \text{if } e(k-1) < 0. \end{cases}$$

Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031

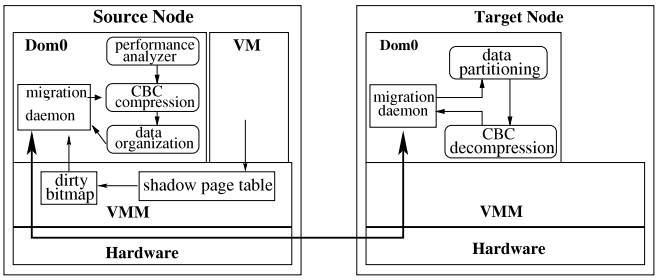


Fig. 7. The MECOM structure.

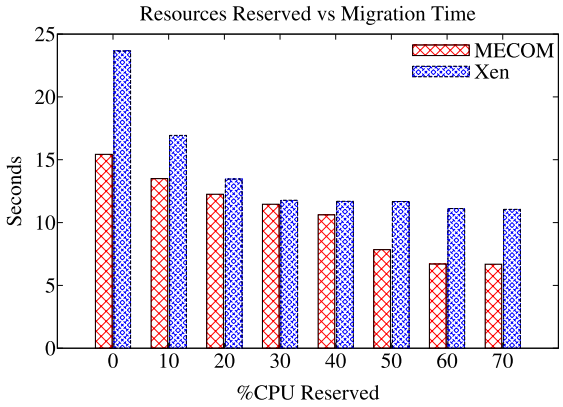


Fig. 8. Effect of CPU reservation on total migration time of Xen and MECOM.

5. System implementation

To demonstrate the utility of our CBC algorithm, we extend Xen 3.1.0 to introduce a memory compression technique in live virtual machine migration.

The whole migration procedure of Xen includes the following stages in turn: *pre-migration*, *reservation*, *iterative pre-copy*, *stop and copy*, *commitment and activation* [3]. The stages *stop and copy* and *commitment* contribute to the downtime of VM migration, while total migration time includes downtime and the duration of the stage *iterative pre-copy*. Memory compression/decompression operations take place in the stages *iterative pre-copy* and *stop and copy*. The source node runs the compression algorithm and the destination node executes the decompression algorithm.

The system structure of MECOM is presented in Fig. 7. Migration daemons running in the management VMs are responsible for performing migration. Shadow page tables in the VMM layer trace modifications to memory page in migrated virtual machines during the pre-copy phase. Corresponding flags are set in a dirty bitmap. At the start of each pre-copying round the bitmap is sent to the migration daemon. Then, the bitmap is cleared and the shadow page tables are destroyed and recreated in the next round.

Our system resides in the management virtual machine of Xen. Memory pages denoted by bitmap are extracted and compressed before being sent to the destination. The compressed data are decompressed on the target. Based on the performance of transferring data collected in the previous rounds of pre-copy process, the performance analyzer periodically adjusts the parameter of the compression algorithm to adapt to the available network bandwidth. The data organization module merges the compressed data of several adjacent pages into the buffer before they are sent to the target node. In the target node, the compressed data of each page are first partitioned by the data partitioning module and then decompressed by the corresponding algorithm.

One big challenge we face after the compression technique is used in the live VM migration process is how to make the compression time as short as possible.

Usually, quick algorithms are simple and have a very short compression time, but their compression ratios are limited. So, there are limits to shorten the compression time just by adjusting the algorithms themselves.

In MECOM, we introduce multi-threaded techniques to parallelize compression tasks, while keeping good compression effects. Memory pages being compressed are independent with each other and compression tasks are appropriate to be parallelized.

Multi-threading techniques minimize overall latency by overlapping the processing time of multiple threads and easily distribute the compression tasks on multiprocessors. Multi-threaded systems efficiently improve the utilization of system resources.

In addition, because dirty pages are produced and sent out in rounds, compression tasks are discontinuous. We exploit a thread pool to reduce thread creation and destruction overheads.

However, because more threads would occupy more system resources and adversely affect migration performance, the number of threads should be set correctly. In our experiments, the number of compression threads is set to 4. For the target host, only one decompression thread is needed due to the extremely fast decompression rate and negligible decompression overhead [18].

6. Performance evaluation

Our implementation has been tested on a wide variety of workloads. In this section we describe the experiments conducted to evaluate our optimized VM migration scheme described above. We primarily care about three performance metrics: migration downtime, total migration time, and the whole amount of data transferred during live migration of VM.

Our results demonstrate that in different application scenarios, our scheme only causes slight service degradation of the migrated VM, outperforming the native pre-copy approach implemented in Xen. Additionally, our experiments show that the system allows much higher dirty memory page rate than Xen during live migration of VM, while keeping a low downtime of millionths of a second.

6.1. Experimental setup

We conduct our experiments on several identical server-class machines, each with 2-way quad-core Xeon E5405 2 GHz CPUs and 8 GB DDR RAM. The machines have an Intel Corporation 80003ES2LAN gigabit network interface card (NIC) and are connected via switched gigabit Ethernet. Storage is accessed via an iSCSI protocol from a NetApp F840 network attached storage server. We use Red Hat Enterprise Linux 5.0 as the guest OS and the privileged domain OS (domain 0). The host kernel is the modified version of Xen 3.1.0. The guest OS is configured to use four VCPUs and 1024 MB of RAM except where noted otherwise.

For each experiment, we take five runs and report the average.

6.2. CPU resource reservation test

To evaluate the migration overhead of the MECOM system, the source physical machine is loaded with the equivalent ten CPU-bound virtual machines. An idle virtual machine with 1 GB memory is migrated under different CPU resource reservations. From Fig. 8 we observe that reserving about 20% of a CPU time for migration without memory compression minimizes the pre-copy time, while our MECOM system needs about 50% of a CPU to achieve minimal downtime due to additional compression and decompression operations. The results show that compression

Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031

Table 3
Effect of CPU reservation on downtime of MECOM and Xen.

CPU reservation (%)		0	10	20	30	40	50	60	70
Downtime (ms)	Xen	10	11	11	10	10	10	11	10
	MECOM	10	11	11	11	10	10	9	10

and decompression altogether introduce about 30% CPU overhead. Moreover, Table 3 shows that downtime of both scheme remains unchanged regardless of the amount of reserved CPU. Little CPU resource is used during the downtime phase.

6.3. Writing memory stress test

We discuss the effects of memory writing rate on the migration of Xen and MECOM. When the memory writing rate becomes larger than the network transferring rate, the writable working set of applications would not converge to a small one. Pre-copy becomes invalid and the duration of downtime increases sharply. From the results of our experiments, we observe that our scheme allows for a larger memory writing rate than Xen, while keeping very short downtime. Furthermore, when the memory writing rate is quick enough to make big writable working sets in both approaches, the downtime of our scheme is much less than that of Xen.

We run a test program written in C language in the migrated VM. This test program continuously writes 512 MB memory in predefined speed. We vary the memory writing speed from 500 to 2400 Mbps and test the downtimes of migration under different speeds. 512 MB memory is initialized with the content of 512 MB VM providing Apache service. Then, random numbers are written to the location that other random numbers point to.

Fig. 9 shows that, for Xen, when memory writing rate is larger than 800 Mbps, the duration of downtime increases sharply. But for MECOM, the rate is about 1800 Mbps, more than twice of that of Xen. When the migrated VM is written in a speed that is more than 1800 Mbps, the downtime of Xen is almost three times that of our approach.

The reason that our approach shows better migration performance is that memory compression indirectly augments network transfer rate during live migration. The amount of dirty pages in each round depends on the following two factors: memory writing rate and the elapsed time of the previous round. Memory compression does not change the speed, but shortens the time spent on flying data in the network due to smaller compressed data sent out. Memory compression makes dirty pages transferred to the destination faster.

6.4. Adaptive page compression

To test the adaptive page compression effects of MECOM, we let live VM migration happen in different network environments. We limit network bandwidth available for VM migration. An idle virtual machine with 1 GB memory is used in our tests. The experimental results are illustrated in Fig. 10.

Fig. 10 lists the total migration time of Xen and MECOM in different network environments. We can observe that MECOM has much shorter migration time than Xen regardless of network bandwidth. The reason is that our CBC algorithm can adjust to a different network bandwidth and greatly improves the performance of live VM migration. The results in Fig. 10 also show that MECOM has a smaller "stable point" of network bandwidth than Xen. The "stable point" means the lowest network bandwidth at which the total migration time of VM becomes stable. That is because shrunken transferred data in MECOM consume much less network resource.

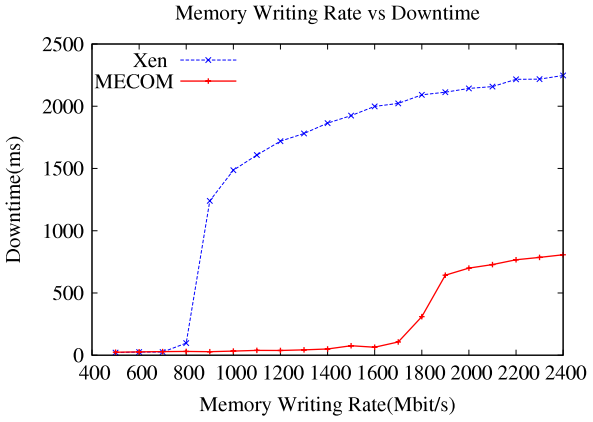


Fig. 9. Effect of memory writing rate on downtime of Xen and MECOM.

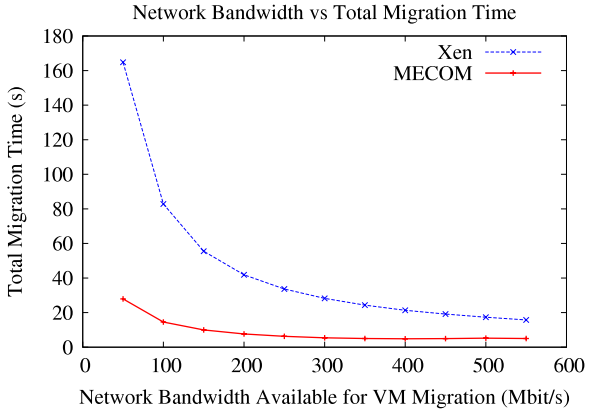


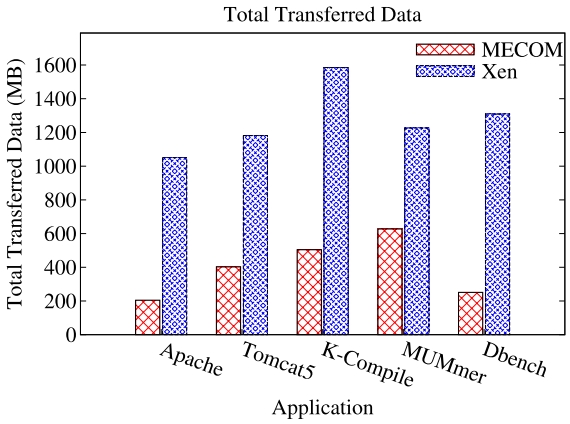
Fig. 10. The performance comparison of Xen and MECOM with different network bandwidth.

6.5. Application scenarios

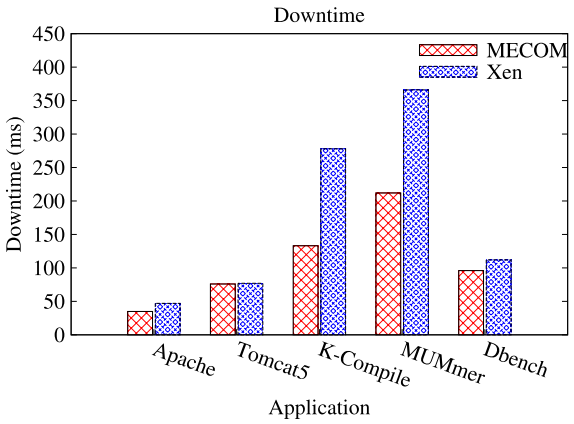
In this part we select several representative applications as the migrated virtual machine's workloads to evaluate the performance of MECOM, compared with Xen. The evaluation primarily includes downtime, total migration time, and total transferred data during live virtual machine migration. These applications are listed below:

1. *Static web server*: We migrate an Apache 2.0.63 web server which serves static content at a high rate. One client is configured with 100 concurrent connections and each connection continuously requests a 512 kB file.
2. *Dynamic web workload*: A Tomcat 5-5.5.23 web server acts as the workload of migrated virtual machine. A complex mix of page requests is handled: 30% of the requests are for dynamic content generation, 16% are HTTP POST operations, and 0.5% executes a CGI script. A number of client machines are used to generate the load for the server and each machine simulates a collection of users concurrently accessing the web site.
3. *Kernel-compile* (*K-compile* for short): The complete Linux 2.6.18 kernel compilation is a system-call intensive workload, which is costly to virtualization. This is a balanced workload that tests CPU, memory, and disk performance.

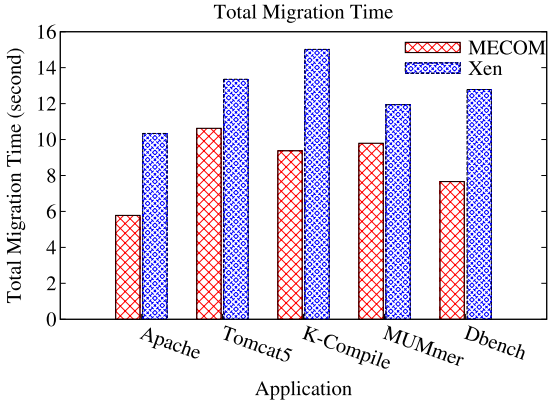
Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031



(a) Total transferred data during live migration for diverse workloads.



(b) The downtime for diverse workloads.



(c) Total migration time for diverse workloads.

Fig. 11. The performance comparison of Xen and MECOM with diverse workloads.

4. *MUMmer*: MUMmer 3.21 [24] is a scientific application to rapidly align genomes that has a very intensive memory usage. Almost all of its memory usage is composed of pages backed by swap.
5. *dbench*: dbench 4.0 [25] is an open source benchmark producing the file system load.

The migrated virtual machine as the only guest one resides on the source node and there are no guest virtual machines on the target machine.

Total data transferred. Fig. 11(a) shows that our MECOM scheme dramatically reduces the total data transferred during the whole migration process, compared with Xen. Although MECOM and Xen have the same dirty pages in the first round, much smaller amounts of data have been sent to the target by MECOM due to compression operations, resulting in fewer dirty pages in the next round. Plus, compression further reduces the amount of data really sent out in each round. Therefore, MECOM reaches the close point of pre-copy algorithm in higher speed than Xen. Experimental results show that MECOM reduces network traffic of VM migration by 48.8% at least (MUMmer) and by 80.8% at most (dbench), by 68.8% on average. Low-bandwidth wide area networks (WAN) would benefit greatly from MECOM.

Total time and downtime. Total migration time and downtime are two key performance metrics that clients of VM service care about the most, because they are concerned about service degradation and the duration that service is completely unavailable.

Fig. 11(b) shows that our scheme has a shorter downtime than Xen. This is because dirty pages in the last round are fast compressed before being sent out. For the above applications, downtime is reduced by 25.5%, 1.3%, 52.2%, 42.1%, and 14.3%, respectively. The performance improvement of Apache workload is little primarily because the size of memory pages is so small that the compression ratio is very low.

The results in Fig. 11(c) show that, compared with Xen, MECOM has greater performance improvement on total migration time than on downtime. This should be attributed to smaller number of rounds and less data transferred in each round. Our system reduces total migration time by 32% in average. In clusters or data centers, less total migration time of VMs would be more attractive to administrators.

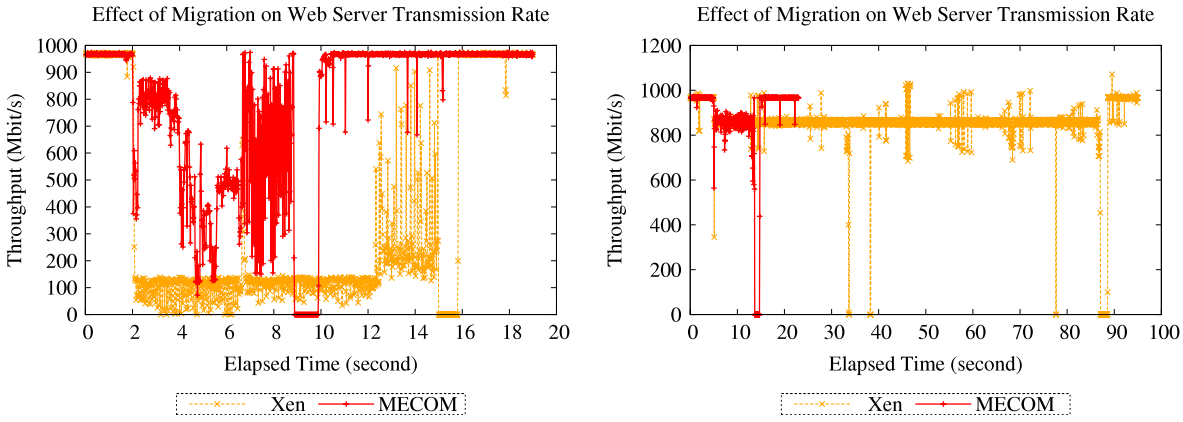
Network throughput. Fig. 12(a) illustrates the effect of migration on Apache web server transmission rate using Xen and MECOM. Compared with Xen, MECOM not only significantly improves transmission rate, but also shortens total migration time. The reason for the above differences is that MECOM uses memory compression to shrink total transferred data during migration.

Fig. 12(b) shows that with adaptive rate limiting, high web server performance has been kept both in two approaches. But total migration time of MECOM is only about one-eighth of that of Xen and downtime is nearly one-fourth. The result reveals that the large amount of transferred data is the key performance bottleneck of migration and MECOM gives a good solution to the problem.

7. Related work

Existing virtual machine migration techniques include two modes: live VM migration and non-live VM migration [26].

Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031



(a) Without adaptive rate limiting.

(b) With adaptive rate limiting.

Fig. 12. Results of migrating a running web server VM using Xen and MECOM with adaptive rate limiting or not.

Table 4
Virtual machine migration.

Categories		Characteristics	Typical systems
Live VM migration	Pre-copy	Transfer memory data in pre-copy phase	VMware, Xen, KVM, openVZ
	Trace and replay	Transfer checkpoint and execution trace files in pre-copy phase	CR/TR-Motion
	Post-copy	Memory transfer is deferred after CPU status transfer phase	Michael R. Hines, VEE2009
Non-live VM migration		VM is suspended during whole migration process	Zap, Collective, Denali, etc.

Pre-copy is the prevailing live migration technique to perform live migration of VMs. These include hypervisor-based approaches such as VMware [4], Xen [27], and KVM [28], operating-system level approaches such as OpenVZ system [29], as well as migration over a wide-area network [30,31]. Most of the existing systems use a pre-copy algorithm for live VM migration in a memory-to-memory style. An exception is CR/TR-Motion [15] which transfers checkpoint and execution trace files rather than memory pages in pre-copy phase. Additionally, D.K. Panda et al. [32] use remote direct memory access (RDMA) to transfer VM migration traffic.

Post-copy technique, previously studied in the context of process migration literature, has been introduced to migrate VMs [16]. The memory "copy" phase of live migration is deferred until the VM's CPU state has been migrated to the target node.

In non-live VM migration, the execution of the VM is suspended during the whole migration process. Several non-live VM migration approaches have been proposed. Zap virtualization [33] is integrated with a checkpoint-restart mechanism that enables process groups, called pods, to be migrated freely. Collective [26] addresses user mobility and system administration by encapsulating the state of the computing environment as capsules that can be transferred between distinct physical hosts. The μDenali [34] is built to be an extensible and programmable virtual machine monitor. It addressed migration of check-pointed virtual machines across a network incurring longer migration downtime. The Internet Suspend/Resume project [35] focuses on the capability to save and restore computing state on anonymous hardware.

Virtual machine migration techniques are summarized in Table 4.

For decades, in-memory compression is used to bridge the huge performance gap between normal RAM and disk. There are two main categories of data compression techniques: one based on statistical models and the other using a dictionary. Huffman coding [36] is one of the most common statistical compression techniques, while Lempel–Ziv (LZ) coding is a typical representative of compression based on dictionary. LZO [18] is a modern implementation. The WK compression algorithm [21] is a combination of dictionary and statistical techniques specifically designed to quickly and efficiently compress program data.

8. Conclusion and future work

In this paper we have presented the design, implementation and evaluation of MECOM, which first introduces memory compression technique into live VM migration. Based on memory page characteristics, we design a particular memory compression algorithm for live migration of VMs. Because the smaller amount of data is transferred and only very low compression overhead is introduced, the total migration time and downtime are both decreased significantly. Service degradation is also decreased greatly. Experimental results show that our system can get better average performance than Xen: up to 27.1% on virtual machine downtime, up to 32% on total migration time, and up to 68.8% data cut down that must be transferred.

We implement live migration of para-virtualized VM with memory compression. In the future, we will extend the technique to full-virtualized VM. We also plan to apply our scheme on physical machines with single CPU or one-way two-cores. It is a significant challenge to further diminish memory compression overheads. Furthermore, we intend to use our approach to implement live wide-area migration of virtual machines including local persistent state. Because Wide-Area Networks (WANs) typically have low available bandwidth, it is still a challenge for MECOM to fast migrate virtual machines especially both with memory data and huge disk data.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant 61073024 and 61232008. It is also supported by the Outstanding Youth Foundation of Hubei Province under Grant 2011CDA086, National 863 Hi-Tech Research and Development Program under Grant 2013AA01A213 and 2013AA01A208, and Research Fund for the Doctoral Program of MOE under Grant 20110142130005.

References

[1] R. Goldberg, Survey of virtual machine research, IEEE Computer (1974) 34–45.
[2] W. Gao, H. Jin, S. Wu, X. Shi, J. Yuan, Effectively deploying service on virtualization infrastructure, Frontiers of Computer Science 6 (4) (2012) 398–408.
[3] 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 Second Symposium on Networked Systems Design and Implementation, NSDI'05, 2005, pp. 273–286.
[4] M. Nelson, B. Lim, G. Hutchines, Fast transparent migration for virtual machines, in: Proceedings of the USENIX Annual Technical Conference, USENIX'05, 2005, pp. 391–394.
[5] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath, P.Y. Wang, Seamless live migration of virtual machines over the MAN/WAN, Future Generation Computer Systems 22 (2006) 901–907.
[6] H. Jin, W. Gao, S. Wu, X. Shi, X. Wu, F. Zhou, Optimizing the live migration of virtual machine by CPU scheduling, Journal of Network and Computer Applications 34 (4) (2011) 1088–1096.
[7] H. Chen, R. Chen, F. Zhang, B. Zang, P.-C. Yew, Live updating operating systems using virtualization, in: Proceedings of the Second ACM/USENIX Conference on Virtual Execution Environments, VEE'06, 2006, pp. 35–44.
[8] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, Black-box and gray-box strategies for virtual machine migration, in: Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation, NSDI'07, 2007, pp. 229–242.
[9] A.B. Nagarajan, F. Mueller, C. Engelmann, S.L. Scott, Proactive fault tolerance for HPC with Xen virtualization, in: Proceedings of 21st ACM International Conference on Supercomputing, ICS'07, 2007, pp. 23–32.
[10] R. Nathuji, K. Schwan, Virtual power: coordinated power management in virtualized enterprise systems, in: Proceedings of the 21st ACM Symposium on Operating Systems Principles, SOSP'07, 2007, pp. 265–278.
[11] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, J. Lawall, Entropy: a consolidation manager for clusters, in: Proceedings of the ACM/USENIX International Conference on Virtual Execution Environments, VEE'09, 2009, pp. 41–50.
[12] L. Hu, H. Jin, X. Liao, X. Xiong, H. Liu, Magnet: a novel scheduling policy for power reduction in cluster with virtual machines, in: Proceedings of the IEEE International Conference on Cluster Computing, Cluster'08, 2008, pp. 13–22.
[13] X. Liao, H. Jin, H. Liu, Towards a green cluster through dynamic remapping of virtual machines, Future Generation Computer Systems 28 (2) (2012) 469–477.
[14] P. Padala, K.G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, 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.
[15] H. Liu, H. Jin, X. Liao, L. Hu, C. Yu, Live migration of virtual machine based on full system trace and replay, in: Proceedings of the 18th International Symposium on High Performance Distributed Computing, HPDC'09, 2009, pp. 101–110.
[16] M.R. Hines, K. Gopalan, Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning, in: Proceedings of the ACM/USENIX International Conference on Virtual Execution Environments, VEE'09, 2009, pp. 51–60.
[17] D. Gupta, S. Lee, M. Vrable, S. Savage, A.C. Snoeren, G. Varghese, G.M. Voelker, A. Vahdat, Difference engine: harnessing memory redundancy in virtual machines, in: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, OSDI'08, 2008, pp. 309–322.
[18] LZO real-time data compression library (online), 2013. http://www.oberhumer.com/opensource/lzo/.
[19] M.M. Theimer, K.A. Lantz, D.R. Cheriton, Preemptable remote execution facilities for the v-system, in: Proceedings of the 10th ACM Symposium on Operating System Principles, SOSP'85, 1985, pp. 2–12.
[20] M. Nelson, J.-L. Gailly, The Data Compression Book, second ed., M & T Books, 1995.
[21] P.R. Wilson, S.F. Kaplan, Y. Smaragdakis, The case for compressed caching in virtual memory systems, in: Proceedings of the USENIX Annual Technical Conference, USENIX'99, 1999, pp. 101–116.
[22] M. Ekman, P. Stenstrom, A robust main-memory compression scheme, in: Proceedings of the International Symposium on Computer Architecture, ISCA'05, 2005, pp. 74–85.
[23] Compressed caching: performance and compression statistics (online), 2013. http://linuxcompressed.sourceforge.net/linux24-cc/statistics/.
[24] A system for aligning entire genomes (online), 2013. http://mummer.sourceforge.net/.
[25] Open source benchmark producing the file system load (online), 2013. http://samba.org/ftp/tridge/dbench.
[26] C.P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M.S. Lam, M. Rosenblum, Optimizing the migration of virtual computers, in: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, OSDI'02, 2002, pp. 377–390.
[27] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Proceedings of the 19th ACM symposium on Operating Systems Principles, SOSP'03, 2003, pp. 164–177.
[28] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, KVM: the Linux virtual machine monitor, in: Proceeding of the 2007 Ottawa Linux Symposium, 2007, pp. 225–230.
[29] Container-based virtualization for Linux (online), 2013. http://openvz.org/Main_Page/.
[30] R. Bradford, E. Kotsovinos, A. Feldmann, H. Schioberg, Live wide-area migration of virtual machines including local persistent state, in: Proceedings of the 3rd International Conference on Virtual Execution Environment, VEE'07, 2007, pp. 169–179.
[31] Y. Luo, B. Zhang, X. Wang, Z. Wang, Y. Sun, H. Chen, Live and incremental whole-system migration of virtual machines using block-bitmap, in: Proceedings of the IEEE International Conference on Cluster Computing, Cluster'08, 2008, pp. 99–106.
[32] W. Huang, Q. Gao, J. Liu, D.K. Panda, High performance virtual machine migration with RDMA over modern interconnects, in: Proceedings of the IEEE International Conference on Cluster Computing, Cluster'07, 2007, pp. 11–20.
[33] S. Osman, D. Subhraveti, G. Su, J. Nieh, The design and implementation of Zap: a system for migrating computing environments, in: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, OSDI'02, 2002, pp. 361–376.
[34] A. Whitaker, R.S. Cox, M. Shaw, S.D. Gribble, Constructing services with interposable virtual hardware, in: Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation, NSDI'04, 2004, pp. 169–182.
[35] M. Satyanarayanan, B. Gilbert, M. Toups, N. Tolia, A. Surie, D.R.O. Hallaron, A. Wolbach, J. Harkes, A. Perrig, D.J. Farber, M.A. Kozuch, C.J. Helfrich, P. Nath, H. Lagar-Cavilla, Pervasive personal computing in an Internet suspend/resume system, IEEE Internet Computing 11 (2007) 16–25.
[36] D.A. Huffman, A method for the construction of minimum-redundancy codes, Proceedings of the Institute of Radio Engineers (1952) 1098–1101.

Hai Jin received his Ph.D. in Computer Science from Huazhong University of Science and Technology (HUST), China, in 1994. He is now Dean of the School of Computer Science and Technology at HUST. Jin is a senior member of the IEEE and a member of the ACM. His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security.

Li Deng received her Ph.D. in Computer Science from Huazhong University of Science and Technology (HUST), China, in 2013. She is currently working in the School of Computer Science and Technology at Wuhan University of Science and Technology, China. Her main research interests include virtualization technology for computing systems, parallel and distributed computing.

Song Wu received his Ph.D. in Computer Science from Huazhong University of Science and Technology (HUST), China, in 2002. He is now a Professor in the School of Computer Science and Technology at HUST. His research interests include parallel and distributed computing and virtualization technology for computing systems.

Xuanhua Shi received his Ph.D. in Computer Science from Huazhong University of Science and Technology (HUST), China, in 2005. From 2006, he worked as an INRIA Post-Doc in PARIS team in Rennes for one year. Currently he is an Associate Professor in the Service Computing Technology and System Lab (SCTS) and Cluster and Grid Computing Lab (CGCL) at HUST. His research interests include cluster and grid computing, fault-tolerance, web services, network security and grid security and virtualization technology.

Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031

Hanhua Chen received his Ph.D. in Computer Science from Huazhong University of Science and Technology (HUST), China, in 2010. He is now an Associate Professor in the School of Computer Science and Technology at HUST. He worked at the Hong Kong University of Science and Technology as a Post-Doctoral Research Associate between 2009 and 2010, and as a visiting scholar between 2007 and 2009. His research interests include distributed computing, wireless sensor networks, and peer-to-peer computing. He is a member of the IEEE.

Xiaodong Pan received his M.S. in Computer Science from Huazhong University of Science and Technology (HUST), China, in 2009. He worked as a Service Operation Engineer at Baidu Online Network Technology Co., Ltd. for one year since 2009, and then joined Baofeng Network Technology Co., Ltd. as a Senior Software Engineer until now. His research interests include virtualization technology for computing systems, load balancing and fault tolerance.

Please cite this article in press as: H. Jin, et al., MECOM: Live migration of virtual machines by adaptively compressing memory pages, Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.09.031