Live Virtual Machine Migration with Adaptive Memory Compression

Hai Jin, Li Deng, Song Wu, Xuanhua Shi, Xiaodong Pan

Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China
hjin@hust.edu.cn

***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 a predominantly used approach in the state of the art, it is difficult to provide quick migration with low network overhead, due to a great amount of transferred data during migration, leading to large performance degradation of virtual machine services. This paper presents the design and implementation of a novel memory-compression-based VM migration approach (MECOM) that first 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. Pages are quickly compressed in batches on the source and exactly recovered on the target. Experiment demonstrates that compared with Xen, our system can significantly reduce 27.1% of downtime, 32% of total migration time and 68.8% of total transferred data on average.**

I. Introduction

Virtual machines (VMs) [1] not only provide efficient and secure computing resource containers, but also can be migrated smoothly among multiple physical machines. Live migration of virtual machines has been a strong management tool in multiple-VM-based environment. It facilitates system maintenance, load balancing, fault tolerance, and power management.

In a VM-based cluster, multiple virtual machines share a physical resource pool. Due to dynamically varying workloads, some nodes are often under-utilized, while others may become heavily-loaded. Some virtual machine migration strategies [2][3] are proposed to balance VM loads among physical nodes. Based on periodically collected resource usage status, some virtual machines are migrated from overloaded machines to light-loaded ones. In addition, by monitoring health status of nodes, such as temperatures or disk error logs, node failures can be anticipated. A proactive scheme for fault tolerance [4] migrates VMs from unhealthy nodes to healthy ones using live migration of virtual machines. With the advent of multi-core or many-core, power management has been a critical and necessary component of computing systems. Live migration of virtual machines provides a flexible way to implement power-budget, power reservation and power-saving [5][6][7]. By consolidating VMs on several light-loaded physical machines, some idle nodes can be powered off. In various application 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. In this paper, we focus on VM migration within a cluster environment where a network-accessible storage system (such as SAN or NAS) is employed. Only memory and CPU status needs to be transferred from the source node to the target one. Live migration techniques in the state of the art mainly use pre-copy approach [8][9], which first transfers all memory pages and then copies pages just modified during the last round iteratively. VM service downtime is expected to be minimal 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. In pre-copy phase, although VM service is still available, great service degradation would happen because migration daemon continually consumes network bandwidth to transfer dirty pages in each round. An adaptive rate limiting approach is employed [8] to mitigate the issue, but total migration time is prolonged nearly by ten times. 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.

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 [10] to provide fast VM migration. The approach transfers execution trace file in iterations rather than dirty pages, which is logged by a trace daemon. 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 drastically reduced. However, CR/TR-Motion is valid only when the log replay rate is larger than the log growth rate. The inequality between source and target nodes limits application scope of live VM migration in clusters. Another novel strategy post-copy is also introduced into live migration of virtual machines [11]. In this approach, all memory pages are transferred only once during the whole migration process and the baseline total migration time is achieved. But the downtime is much higher than that of pre-copy due to the latency of fetching pages from the source node

978-1-4244-5012-1/09/$25.00 ©2009 IEEE

before VM can be resumed on the target.

With the advent of multi-core or many-core, there are abundant CPU resources available. Even if several VMs reside on a same multi-core machine, CPU resource is still rich because physical CPUs are frequently amenable to multiplexing [12]. We can exploit these copious CPU resources to compress page frames and the amount of transferred data can be significantly reduced. Memory compression algorithms typically have little memory overhead. Decompression is simple and very fast and requires no memory for decompression.

This paper presents a novel approach to optimize live virtual machine migration based on pre-copy algorithm. We first use memory compression to provide fast VM migration. Virtual machine migration performance is greatly improved by cutting down the amount of transferred data. We also design a zero-aware *characteristics-based compression* (CBC) algorithm for live migration. In the source node, data being transferred in each round are first compressed by our algorithm. When arriving on the target, compressed data are then decompressed. Compression time is performance bottleneck of additional overhead introduced by compression operations. We use multi-threading techniques to parallelize compression tasks and the number of threads is tuned to provide best performance. We implement our prototype system MECOM based on Xen 3.1.0.

The main contributions of the paper are listed as follows: 1) We design and implement a novel approach that first exploit memory compression technique to minimize the downtime and total migration time of live migration. 2) By analysing memory page characteristics, we implement an adaptive memory compression approach to tune migration performance of virtual machines.

The rest of the paper is organized as follows. In section II we present the design of characteristics-based compression algorithm for live migration. Section III describes the implementation of system MECOM in detail. In section IV we describe our evaluation methodology and present the experimental results. We discuss related work in VM migration and memory compression in section V. Finally, we summarize our contributions and outline our future work in section VI.

II. Algorithm Design

A. 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 dirty pages take shorter time to fly on the network. On the other hand, network traffic of migration drops dramatically when much less data are transferred between source and target nodes.

The compression algorithm is first lossless because the compressed data need to be exactly reconstructed. Second, the overhead of memory compression should be as small as possible. If the overhead outweighs the advantage of memory compression, live VM migration would not get any benefit from it. Compression overhead primarily denotes to CPU overhead because memory compression algorithms typically occupy negligible memory region. CPU overhead is often reflected in compression time. However, there are some contradictions between overheads and compression effects. A compression algorithm with low overhead is ordinarily simple but difficult to achieve high compression ratio. So, how to balancing the relationship between overheads and compression effects is crucial to design a good algorithm for live VM migration.

Next, we discuss the relationship of overheads and compression effects with formalized characterization. The analysis would further direct the design of compression algorithm.

First, some notations are defined as following:

R_{page}: the average growth rate of dirty pages in the migrated virtual machine.

R_{tran}: the average page transfer rate, which denotes available network bandwidth for transferring pages from the source node to the target.

R_{cpr}: the average memory compression rate.

ρ_{cpr}: the average compression ratio, the percentage of freed data.

V_{thd}: the threshold of dirty pages where pre-copy process is stopped.

V_{tms}: the total memory size of virtual machine.

For our MECOM approach, transferred data size and the elapsed time in all rounds are respectively represented by vector $V=<v_0, v_1, \ldots, v_{n-1}>$ and vector $T=<t_0, t_1, \ldots, t_{n-1}>$. For Xen, the corresponding sequences of transferred data size and the elapsed time are defined as vector $V'=<v'_0, v'_1, \ldots, v'_{n-1}>$, vector $T'=<t'_0, t'_1, \ldots, t'_{n-1}>$.

Let λ=1-ρ_{cpr}, then

$$
\begin{aligned}
t_0 &= \frac{V_{tms}\lambda}{R_{tran}} + \frac{V_{tms}}{R_{cpr}}, & t'_0 &= \frac{V_{tms}}{R_{tran}} \\
t_1 &= \frac{R_{page}\lambda t_0}{R_{tran}} + \frac{R_{page}t_0}{R_{cpr}}, & t'_1 &= \frac{R_{page}t'_0}{R_{tran}} \\
&\ldots \\
t_n &= \frac{R_{page}\lambda t_{n-1}}{R_{tran}} + \frac{R_{page}t_{n-1}}{R_{cpr}}, & t'_m &= \frac{R_{page}t'_{m-1}}{R_{tran}}
\end{aligned} \quad (1)
$$

And,

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

Only when t_0 is less than t'_0, the performance of migration can be improved. From $t_0<t'_0$, we can get

$$R_{cpr}\rho_{cpr} > R_{tran} \quad (2)$$

The formula (2) indicates that the product of compression time and compression ratio should be larger than network

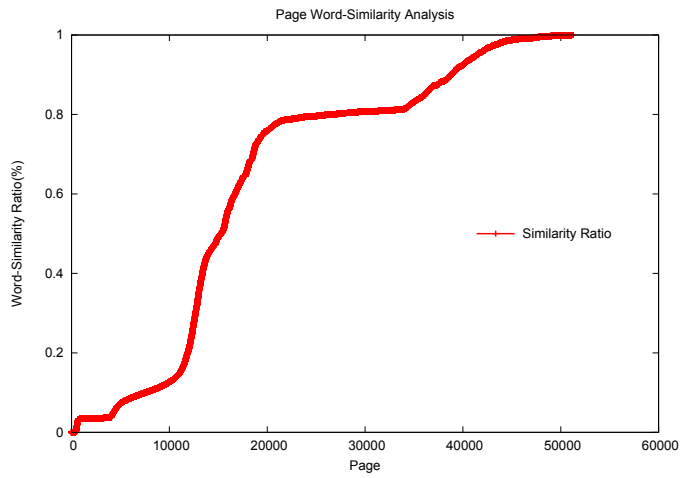

Fig. 1. Memory page word-similarity ratio distribution of a 256MB-RAM virtual machine providing Apache service

bandwidth available for migration so that live VM migration can show better performance.

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 two phases, which are typically interleaved in practice: modeling and encoding [13]. Modeling is the process of finding regularities that allow a more concise representation of the information. Encoding is the construction of that more concise representation.

B. Memory Data Characteristic Analysis

To a great extent, the performance of compression algorithms depends on data being compressed. They must exploit data representation characteristics to achieve any compression. Paul R. Wilson [14] deems that memory data representations have certain strong regularities, although diverse applications might be loaded into the memory. Moreover, there are a great number of zero bytes in the memory pages [15]. We first analyse memory data representation of VMs.

A VM is configured with 256MB RAM, in which an Apache 2.0.63 web server is running. Redhat Enterprise Linux 5.0 is used as the guest operating system (OS). We analyse 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 word-similarity of the page. 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 word-similarity ratio. The result is listed in Figure 1.

From Figure 1, 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. Table I shows the similarity ratio distribution of pages in the memory footprint for test suite from [16]. The test suite is especially for compression and performance statistics. The first workload *null* in Table I denotes an idle VM.

As Table I 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.

C. Characteristic-Based Compression (CBC) Algorithm for Live VM Migration

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

If an algorithm itself embodies expectations about the kinds of regularities in the memory footprint, it must be very fast and effective. As we analyse in the above part, memory data in VMs with various workloads do not always show the same regularity. Then, a single compression algorithm for all memory data is difficult to achieve the win-win status that we expect. Therefore, we should provide several compression algorithms 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 II-B, we first classify pages into the following kinds: 1) pages composed of a great many of zero bytes and sporadic non-zero bytes; 2) pages with high similarity; 3) pages with low similarity.

For the first kind of pages, we can scan the whole page and just record the information about the offset and value of non-bytes. For the second kind of pages, we can use algorithms that embody strong similarities, such as WKdm [14]. 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 algorithm with high compression ratio is appropriate. LZO [17], a modern implementation of Lempel-Ziv compression, is an option.

Our compression algorithm CBC primarily includes two parts:

TABLE I
MEMORY PAGE CHARACTERISTICS ANALYSIS

Word-Similarity Ratio(%) 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	0.19142
75-100	0.338174	0.538479	0.321932	0.2172	0.362381	0.158034	0.361128

```
Input: pages
Output: compressedData

Initialize a 16−word dictionary;
For each page{
   zeroByte = 0;
   wordSimilarity = 0;
   For each word in page{
      For each byte in word{
         if(byte is zero−byte){
            zeroByte++;
         }
      }
      if(word is zero−word or word matches one in dictionary){
         wordSimilarity++;
      }
   }
   if(zeroByte >= threOfZeroByte * totalNumOfByte){
      push 0 into compressedData;
      For each byte in page{
         if(byte is not zero−byte){
            push (offset,value) into compressedData;
         }
      }
   }else if(wordSimilarity >= threshold * totalNumOfWord){
      push 1 into compressedData;
      compress page with algorithm for high wordSimilarity and
            push the result into compressedData;
   }else{
      push 2 into compressedData;
      compress page with algorithm for low wordSimilarity and
            push the result into compressedData;
   }
}
return compressedData;
```

Fig. 2. Pseudo-code of the CBC compression algorithm

(1) Determining the kind of pages being compressed. The intention of the step is to pick out pages with strong regularities, which mean high 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 algorithm. 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 in Figure 2.

D. Benefits from CBC Algorithm

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

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

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

From equation sets (1) listed in part II-A, we get the following equations:

$$t_n = V_{tms} R_{page}^n (\frac{\lambda}{R_{tran}} + \frac{1}{R_{cpr}})^{n+1} \tag{3}$$

$$t'_m = \frac{V_{tms} R_{page}^m}{R_{tran}^{m+1}} \tag{4}$$

If our approach MECOM using compression runs n rounds to reach the threshold and Xen runs m rounds in pre-copy phase, we can get the equation: t_n=t'_m. From equation (3) and (4), we can deduce the equation

$$R_{page}^n (\frac{\lambda}{R_{tran}} + \frac{1}{R_{cpr}})^{n+1} = \frac{R_{page}^m}{R_{tran}^{m+1}} \tag{5}$$

From inequation (2) in part II-A and equation (5), we get φ^m<φ^n. Because φ<1, we can deduce

$$m > n$$

The above inequation implies that when dirty page rate is less than available network bandwidth for migration, our approach MECOM has faster convergence rate to reach the close point of pre-copy iterations than Xen. Obviously, for each round i,

$$t_i < t'_i, v_i < v'_i$$

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

(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, pre-copy process is terminated as soon as maximum transfer rate is available, which ensures that the downtime is not prolonged.

Provided that transfer rate R_{tran} is constant R_{tran0}, pre-copy algorithm becomes ineffective when dirty page rate R_{page} equals to R_{tran0} in Xen. But after memory compression is

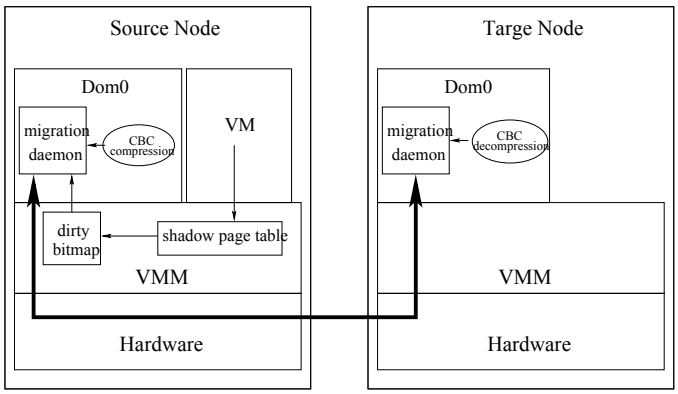

Fig. 3. The MECOM structure

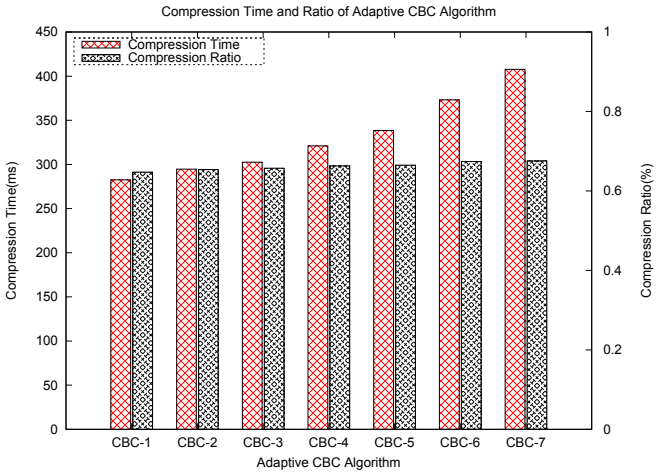

Fig. 4. Compression time and compression ratio of adaptive CBC algorithm

introduced, network transfer rate is augmented indirectly by factor 1/(1 - ρ_{cpr}). Only when actual dirty page rate reaches R_{tran0}/(1 - ρ_{cpr}), pre-copy process is terminated in the system MECOM. That is, MECOM allows higher dirty page rate than Xen. Furthermore, even if real dirty page rate exceeds R_{tran0}/(1-ρ_{cpr}) in our approach, downtime is greatly shortened due to the shrinked amount of transferred data.

III. System Implementation

To demonstrate the utility of our CBC algorithm, we expand Xen 3.1.0 to introduce memory compression technique in live virtual machine migration. The scheme is called MECOM for short.

The whole migration procedure of Xen includes the following stages in turn: *pre-migration*, *reservation*, *iterative pre-copy*, *stop and copy*, *commitment and activation* [8]. 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 compression algorithm and the destination node executes decompression algorithm.

The system structure of MECOM is presented in Figure 3. 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 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 sent to the destination. The compressed data are de-compressed on the target.

The following describes the challenges when the compression technique is introduced to improve live migration of virtual machines in Xen.

A. Adaptive Compression Mode

When dirty page rate is higher than network transfer rate, a great amount of dirty page data are produced by rapid memory writing operations. In the situation, what we care about the most is migration time, not network traffic. For the compression operations, faster algorithms are expected regardless of compression ratios.

We provide an adaptive compression mode to tackle the above problem. Our system detects the amount of dirty pages in real-time and automatically adjusts the threshold between low similarity ratio and high similarity one. Then compression time gets shorter. The amount of dirty pages subsequently becomes relatively small. Figure 4 shows different performance of our adaptive CBC algorithm on a 64MB-RAM VM with a Apache 2.0.63 server. CBC-1 has the lowest threshold of high similarity ratio, which makes more pages compressed using simple but quick algorithm. Although compression ratio is not high, compression time is short. With the number *n* increases, CBC-*n* has higher threshold and compression ratio, but longer compression time.

But for some real-time VM services such as online game or online video, low network traffic of migration is expected. We also provide interfaces for users to designate the threshold between low similarity ratio and high similarity one.

B. Multi-threaded Compression

Usually, quick algorithms are simple and have very short compression time, but their compression ratios are also low. So, there are limits to shorten compression time just by adjusting 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 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 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. After we repeat tests many times, the number of compression threads is set to 4. For the target host, only one decompression thread is needed due to extremely fast decompression rate and negligible decompression overhead.

IV. 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 low downtime in million-second. Furthermore, our results show that the system is able to successfully migrate a VM with CPU-intensive workloads, which implies that our system does not introduce significant CPU resource overhead due to compression and decompression operations.

A. Experimental Setup

We conduct our experiments on several identical server-class machines, each with 2-way quad-core Xeon E5405 2GHz CPUs and 8GB DDR RAM. The machines have Intel Corporation 80003ES2LAN gigabit network interface card (NIC) and are connected via switched gigabit Ethernet. Storage is accessed via iSCSI protocol from a NetApp F840 network attached storage server. We use Redhat 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 4 VCPUs and 1024MB of RAM except where noted otherwise.

Each experiment is repeated five times and every test result comes from the arithmatic average of five values.

B. CPU Resource Reservation Test

To evaluate the migration overhead of MECOM system, source physical machine is loaded with the equivalent of 10 CPU-bound virtual machines. An idle virtual machine with 1GB memory is migrated under different CPU resource reservations. From Figure 5 we observe that reserving about

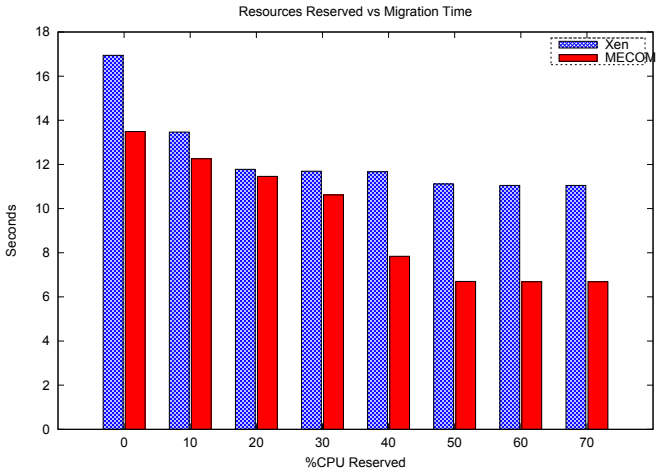

Fig. 5. Effect of CPU reservation on total migration time of Xen and MECOM

TABLE II
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

20% a CPU 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 and decompression altogether introduce about 30% CPU overhead. Moreover, Table II shows that downtime of both schemes remains unchanged regardless of the amount of reserved CPU. Little CPU resource is used during downtime phase.

C. Writing Memory Stress Testing

Next, we will discuss the effects of memory writing rate on migration of Xen and MECOM. When memory writing rate becomes larger than network transferring rate, the writable working set of applications would not converge to a small one. Precopy gets invalid and the duration of downtime increases sharply. From results of our experiments, we observe that our scheme allows larger memory writing rate than Xen, while keeping very low downtime. Furthermore, when memory writing rate is quick enough to make big writable working sets in both approaches, 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 512MB memory in pre-defined speed. We vary the memory writing speed from 500Mbit/s to 2400Mbit/s and test downtime of migration under different speeds. 512MB memory is initialized with the content of 512MB VM providing Apache service. Then, random numbers are written to the location that other random numbers point to.

Figure 6 shows that, for Xen, when memory writing rate is larger than 800Mbit/s, the duration of downtime increases

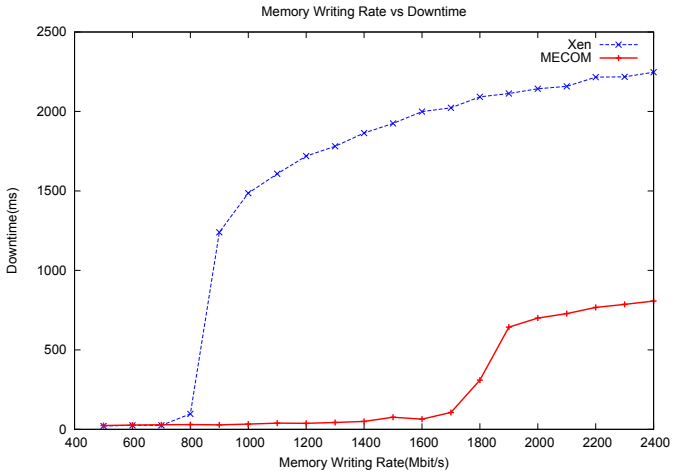

Fig. 6. Effect of memory writing rate on downtime of Xen and MECOM

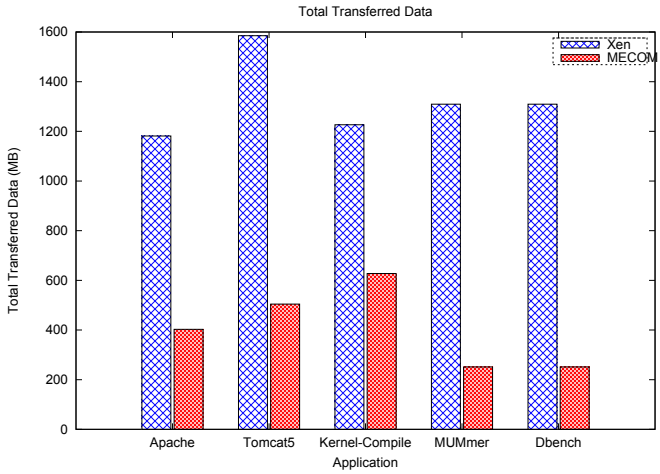

Fig. 7. Total transferred data of Xen and MECOM during live migration for diverse workloads

sharply. But for MECOM, the rate is about 1800Mbit/s, more than twice of that of Xen. When the migrated VM is written in speed that is more than 1800Mbit/s, downtime of Xen is almost three times of 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: one is memory writing rate and the other is the elapsed time of the previous round. Memory compression does not change the speed, but shorten 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 in faster speed.

D. Application Scenarios

In this part we select several representative applications as the migrated virtual machine's workloads to evaluate the performance of MECOM, which is 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 512KB files.

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% require 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: The complete Linux 2.6.18 kernel compilation is a system-call intensive workload, which is expensive to virtualization. This is a balanced workload that tests CPU, memory, and disk performance.

4. MUMmer: MUMmer 3.21 [18] 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 [19] is an open source benchmark producing the filesystem load.

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

Total Data Transferred. Figure 7 shows that our MECOM scheme dramatically reduces 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 amount of data have been sent to the target by MECOM due to compression operations, resulting that there are 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 at least 48.8% (MUMmer) and at most 80.8% (dbench), an average of 68.8%. 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.

Figure 8 shows that our scheme has shorter downtime than Xen, because dirty pages in the last round are also fast compressed before sent out. For the above applications, downtime is reduced by 25.5%, 1.3%, 52.2%, 42.1%, and 14.3%, respectively, an average of 27.1%. 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 test results in Figure 9 show that, compared with

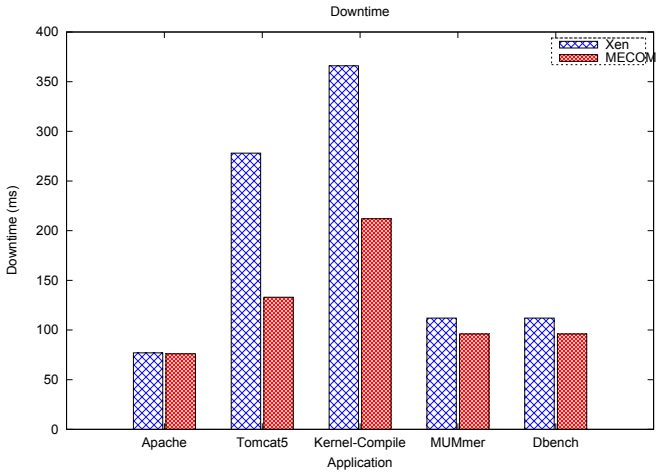

Fig. 8. The downtime of Xen and MECOM for diverse workloads

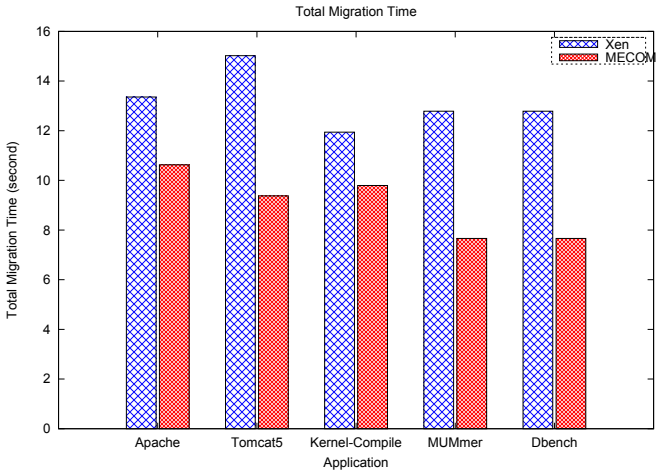

Fig. 9. Total migration time of Xen and MECOM for diverse workloads

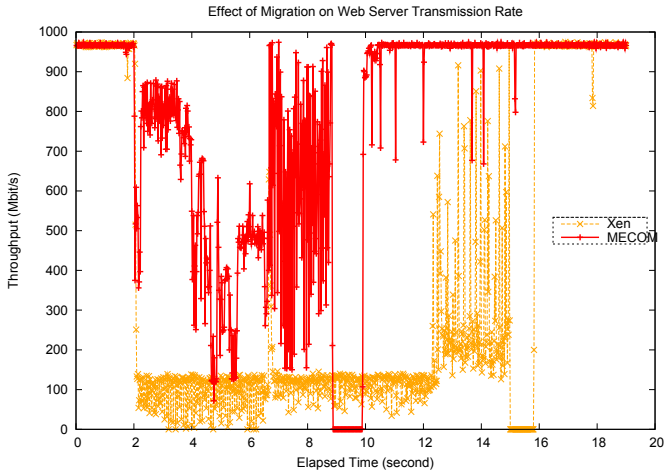

Fig. 10. Results of migrating a running web server VM using Xen and MECOM

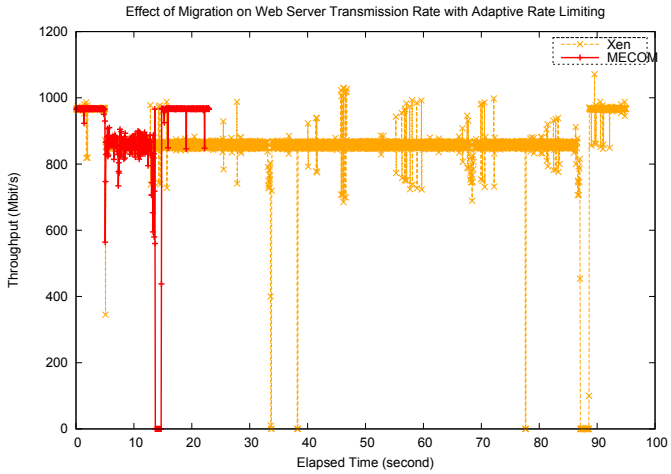

Fig. 11. Results of migrating a running web server VM with adaptive rate limiting using Xen and MECOM

Xen, MECOM has greater performance improvement on total migration time than on downtime. This should be attributed to smaller rounds and less data transferred in each round. Our system reduces total migration time by an average of 32%. In clusters or data centers, less total migration time of VMs would be more attractive to administrators.

Network Throughput. Figure 10 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 only reason for the above differences is that MECOM uses memory compression to shrink total transferred data during migration.

Figure 11 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. Again, our tests prove that the large amount of transferred data is the key performance bottleneck of migration and MECOM gives a good solution to the problem.

E. A Diabolical Workload

Finally, we consider the situation in which a virtual machine uses up all CPUs and continuously writes to memory in high speed. We test this diabolical case using a virtual machine with 8 VCPUs and 1GB memory. In the virtual machine, a memtester process [20] tests 512MB region of memory, while other 8 CPU-intensive processes simultaneously run. The result of the migration is shown in Figure 12.

In first round, half of the memory is compressed and then transferred, while the other half is immediately marked dirty by memtester. Because high dirty page rates are detected, our adaptive compression control approach introduced in section III-A attempts to adjust the threshold to use quicker compression algorithm on more pages. But the attempt seems to be failure because much more dirty pages occur subsequently. In the last iteration, the threshold drops to zero and all dirty pages are compressed with quicker algorithm regardless of page characteristics. However, virtual machines with such workloads seem to be rare in real application scenarios.

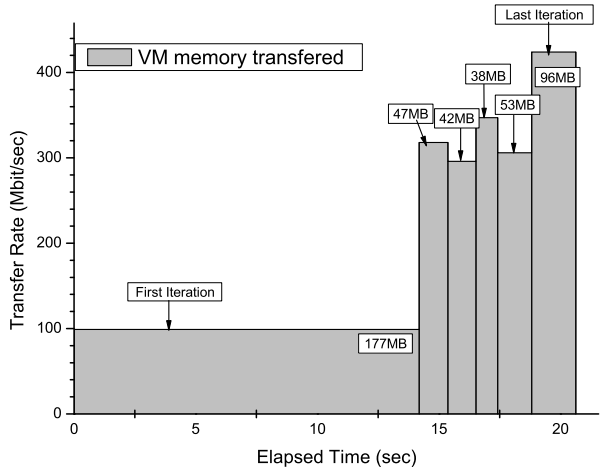

Fig. 12. Results of migrating a VM running a diabolical workload

V. Related Work

Virtual machine migration includes two modes: live VM migration and non-live VM migration.

Pre-copy is the prevailing live migration technique to perform live migration of VMs. These include hypervisor-based approaches such as VMware [9], Xen [21], and KVM [22], operating-system level approaches such as OpenVZ system [23], as well as migration over wide-area network [24][25]. All of the above systems use pre-copy algorithm for live VM migration in a memory-to-memory approach. A novel approach CR/TR-Motion [10] is proposed to transfer checkpoint and execution trace files rather than memory pages in pre-copy phase. Additionally, D.K. Panda et al [26] uses 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 [11]. 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 whole migration process. Several non-live VM migration approaches have been proposed. Zap virtualization [27] is integrated with a checkpoint-restart mechanism that enables process groups, called pods, to be migrated freely. Collective [28] 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 [29] is architected 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 [30] focuses on the capability to save and restore computing state on anonymous hardware.

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 [31] is one of the most common statistical compression techniques, while Lempel-Ziv (LZ) coding is a typical representative of compression based on dictionary. LZO [17] is a modern implementation. The WK compression algorithm [14] is a unique combination of dictionary and statistical techniques specifically designed to quickly and efficiently compress program data.

VI. 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.

Acknowledgment

This work is supported by the National 973 Basic Research Program of China under grant No. 2007CB310900. It is also supported by Information Technology Foundation of MOE and Intel under grant MOE-INTEL-09-03.

References

[1] R. Goldberg, "Survey of virtual machine research," *IEEE Computer*, pp. 34–45, 1974.

[2] 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.

[3] T. Wood, P. Shenoy, A. Venkataramani, and 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.

[4] A. B. Nagarajan, F. Mueller, C. Engelmann, and 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.

[5] R. Nathuji and 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.

[6] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and 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.

[7] L. Hu, H. Jin, X. Liao, X. Xiong, and 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.

[8] 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 Second Symposium on Networked Systems Design and Implementation (NSDI'05)*, 2005, pp. 273–286.

[9] M. Nelson, B. Lim, and G. Hutchines, "Fast transparent migration for virtual machines," in *Proceedings of the USENIX Annual Technical Conference (USENIX'05)*, 2005, pp. 391–394.

[10] 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 International Symposium on High Performance Distributed Computing (HPDC'09)*, 2009, pp. 101–110.

[11] 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/Usenix international conference on Virtual execution environments (VEE'09)*, 2009, pp. 51–60.

[12] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and 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.

[13] M. Nelson and J.-L. Gailly, *The Data Compression Book*, 2nd ed. M & T Books., 1995.

[14] P. R. Wilson, S. F. Kaplan, and 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.

[15] M. Ekman and PerStenstrom, "A robust main-memory compression scheme," in *Proceedings of the International Symposium on Computer Architecture (ISCA'05)*, 2005, pp. 74–85.

[16] (2009) Compressed caching: Performance and compression statistics. [Online]. Available: http://linuxcompressed.sourceforge.net/linux24-cc/statistics/

[17] (2009) LZO real-time data compression library. [Online]. Available: http://www.oberhumer.com/opensource/lzo/

[18] (2009) A system for aligning entire genomes. [Online]. Available: http://mummer.sourceforge.net/

[19] (2009) Open source benchmark producing the filesystem load. [Online]. Available: http://samba.org/ftp/tridge/dbench

[20] (2009) A userspace utility for testing the memory subsystem. [Online]. Available: http://pyropus.ca/software/

[21] 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)*, 2003, pp. 164–177.

[22] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceeding of the 2007 Ottawa Linux Symposium*, 2007, pp. 225–230.

[23] (2009) Container-based virtualization for linux. [Online]. Available: http://www.openvz.com/

[24] R. Bradford, E. Kotsovinos, A. Feldmann, and 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.

[25] Y. Luo, B. Zhang, X. Wang, Z. Wang, Y. Sun, and 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.

[26] W. Huang, Q. Gao, J. Liu, and 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.

[27] S. Osman, D. Subhraveti, G. Su, and 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.

[28] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and 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.

[29] A. Whitaker, R. S. Cox, M. Shaw, and 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.

[30] 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, and H. Lagar-Cavilla, "Pervasive personal computing in an internet suspend/resume system," *IEEE Internet Computing*, vol. 11, pp. 16–25, 2007.

[31] D. A. Huffman, "A method for the construction of minimum-redundancy codes," in *Proceedings of the Institute of Radio Engineers*, 1952, pp. 1098–1101.