# Split Transfer Omitting Redundant Dirty Pages to Accelerate a Virtual Machine Migration

**Jae-Geun Cha[†], Chi-Hoon Shin[††], and Hag-Young Kim[††]**

[†]Computer Software, Science University of Science and Technology, Daejeon, Republic of Korea
[††]Cloud Computing Department, Electronics and Telecommunication Research Institute, Daejeon, Republic of Korea

**Summary**

The pre-copy method for live migration lead to redundant transmission of data in the memory which has already been transmitted. In addition, the migration efficiency decreases in the memory intensive environment where the memory is frequently modified because the rate of dirty page increases. Recently, these problem appeared in memory-intensive environment because Big-data and growing complexity of computing environment. In order to address these problems, we propose a method which splits the memory and omits redundant transmission. We simulated our approach base on the pre-copy method to evaluate the performance. In particular, we used static bandwidth for migration and generated evenly dirty pages during migration to show the effect of transferring the split memory. Simulation result showed that the proposed method saved the total migration time from 6% to 65% compared to pre-copy method. Also, when the environment has dirty page rate of 90%, our approach took less total migration time than MECOM which is the improvised version of pre-copy method. Therefore, the proposed method is able to alleviate system performance degradation because our approach takes less migration time than previous method in Big-data or multi-core environment.

***Key words:***
*Virtual machine, Live migration, Pre-copy, Split memory data.*

## 1. Introduction

The migration of a Virtual Machine (VM) between two systems is to stop the VM, to move the VM from one system (source) to another (destination) without any modification, and to resume the VM [1]. The migration is able to provide continuous VM service to a user no matter when and where the user is or even when errors occur. Figure 1 (a) shows a simple procedure of a VM migration. First, the VM running on a source is stopped. Then, the copy of the VM is transferred to a destination. The time taken in this step is referred as downtime. The downtime could be a few seconds depending on the network bandwidth and the size of data to transfer. Because the out-of-service for the several seconds could interrupt real time services, many studies [2] to minimize downtime have conducted; these studies have been categorized as live migration.

The live migration for a real-time service moves a running VM without halting. During the live migration from a source to a destination, some pages of memory on the source are modified because the VM keeps running on the source. Therefore, the live migration has a synchronization step to match the modified pages on the source and transferred data on the destination. The modified memory page on the source is referred to as dirty page(s).

According to a synchronization strategy, live migrations are grouped into two categories - pre-copy and post-copy. The pre-copy executes the synchronization stage – State Migration in Figure 1- before the stop and copy; the post-copy conducts that after the stop and copy. The Figure 1 (b) and (c) shows these procedures respectively [1],[3].

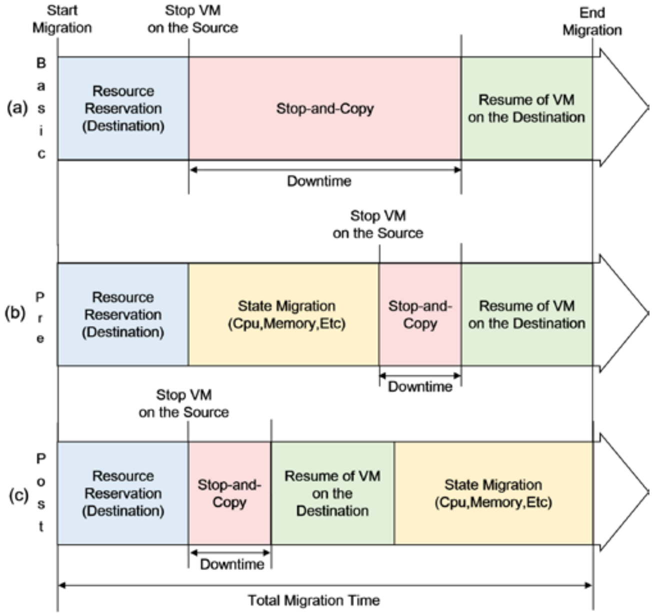

Start Migration
Stop VM on the Source
End Migration
(a)
B a s i c
Resource Reservation (Destination)
Stop-and-Copy
Resume of VM on the Destination
Downtime
Stop VM on the Source
(b)
P r e
Resource Reservation (Destination)
State Migration (Cpu,Memory,Etc)
Stop-and-Copy
Resume of VM on the Destination
Downtime
Stop VM on the Source
(c)
P o s t
Resource Reservation (Destination)
Stop-and-Copy
Resume of VM on the Destination
State Migration (Cpu,Memory,Etc)
Downtime
Total Migration Time

Figure 1 migration timeline

The pre-copy transfers the full memory to destination at the start of a migration. Thereafter, the dirty pages, which occur during migration, are repeatedly transferred. This process is called as iterative pre-copy. Generally, when the dirty page size is less than the minimum size, which is set by the administrator, the pre-copy stops the state migration stage and starts the stop and copy stage. In the stop and copy stage, the remains of dirty page are transferred to

Manuscript received November 5, 2014
Manuscript revised November 20, 2014

destination. After that, the virtual machine resumes on the destination [1].

The pre-copy can minimize the downtime by reducing the size of the dirty pages over Iterative pre-copy. But the memory data that has already been transferred should be retransferred if it is modified. Additionally, if the VM produces more dirty pages than the capacity of data transmission, the iterative pre-copy takes too long time or reaches deadlock [2],[3]. Lately, because of increasing the complex and a large-scale computing environment [4],[5], these problems hamper the performance severely so we need a novel way to improve it.

Recently, a scale of the data grows larger in science, healthcare, social field and so on [4]. The computing environment is continually advanced to handle these data. For example, number of CPU cores increases, and the data processing rate of memory is improved 55% per year [5]. This trend affects the performance of the migration. Z.Ibrahim [6] showed that the migration time gets longer if CPU cores assigned to the VM increase. Generally, the more cores share a memory, the more memory access occurs [7]. In this circumstance, the production rate of the dirty page will grow higher.

Previous works to improve pre-copy focused on reducing the migration time and downtime. Ma [8], Hu [9] proposed the way to transfer a dirty page which is frequently modified in the last iteration of Iterative pre-copy. Jin [10] proposed the data compression method on transmission data. However, these methods could take excessive time for a migration with large-scale of data and the high production rate of dirty page (i.e. memory intensive environment).

In this paper, we propose improved pre-copy method by splitting the transmission data and omitting redundant dirty page in the memory intensive environment. As previously mentioned, the dirty page is the modified memory data during each iteration time. Therefore, we consider saving the time of each iteration to reduce the production rate of dirty page. For this purpose, we conduct splitting the transmission data to save the time of transmission. It will avoid degradation of migration performance in the memory intensive environments during live migration.

This paper is organized as follows. In section 3, we explain the proposed methods in detail. Also, we describe the design of our improved pre-copy approach. Then, we present the experimental results in section 4. Finally, we conclude and give our future work in section 5.

## 2. Split transfer Omitting Redundant Dirty pages method

### 2.1 Split transfer Omitting Redundant Dirty pages method

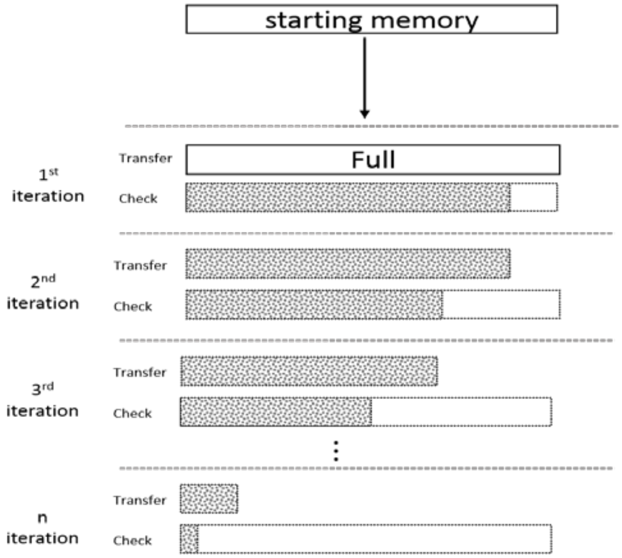


Figure 2 Iterative pre-copy procedure

In this section, we explain our approach, the Split Transfer Omitting Redundant Dirty-pages (STORD), designed to improve the typical pre-copy migration. The conventional pre-copy greedily sends dirty pages until the size of the pages grows small enough to download. The Figure 2 shows the typical pre-copy executes iterative procedures. The 1st iteration consists of two steps. In the first step, a source VM transfers the full memory, which is a copy of the starting memory, to a destination VM. Simultaneously, the source VM accessing to the main memory generates dirty pages. In the second step, the source VM checks the dirty pages, and prepares to move them to the destination VM [1].

The *2nd iteration* has the same steps as the *1st iteration*: *transfer* and *check*. In the *transfer* of the *2nd iteration*, the source VM transfers the dirty pages, which are checked in the *1st iteration*, instead of the full memory. In the *check* step of the 2nd iteration, the source VM identifies recent dirty pages during transferring the old dirty pages. This two-step iteration can repeat *n* times. The mechanism of an *nth iteration* is identical to the iterations abovementioned. The VM in the *nth iteration* moves the dirty pages checked in the previous iteration, i.e. (n-1)th. Then the source VM checks recent dirty pages [1].

The iterations can continue until the size of dirty pages becomes smaller than the constraint defined by the migration administer. In other words, the size of dirty page decides when to stops the iterative procedures and starts the down downtime procedure [1].

Also, the dirty pages is direct proportion in memory data transfer time in previous iteration if the workload of virtual machine is static. Therefore, if a memory data decreases or a network bandwidth increases, the dirty pages rate decreases. And it reduces total migration time and downtime.

In this paper, we propose the STORD that reduces a reducing transfer memory data to reduce the dirty page rate during each iteration. The details of the STORD are illustrated in the Figure 3.

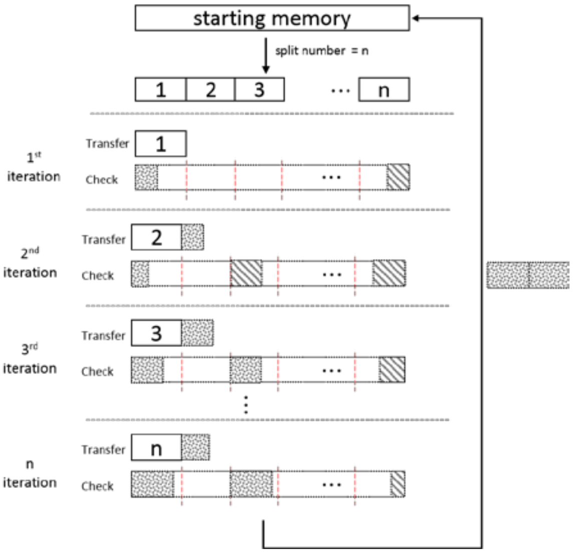


Figure 3 proposed method procedure

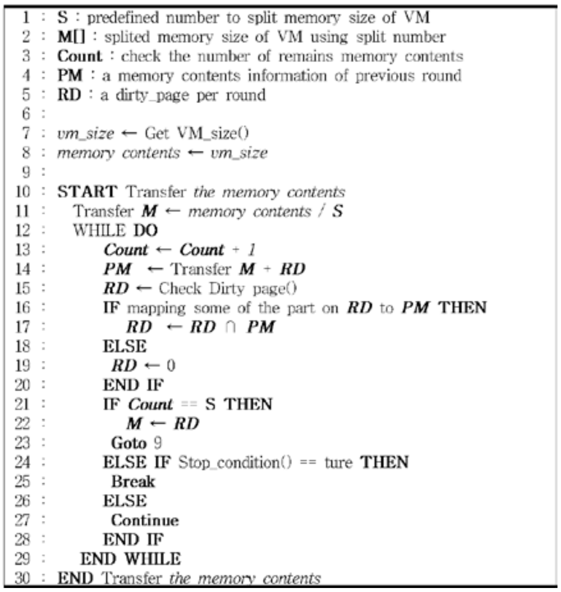
Before moving in the iterative flow, the STORD splits the starting memory into *k* pieces; the *k* is predefined split number (e.g. if the *k* is two, the memory will be split into two pieces). During the first iteration, the STORD transfers one split piece of memory to destination node. At the same time, it checks dirty pages during transferring the piece of memory.

In the *2nd iteration*, the STORD transfers next piece of split memory and dirty pages to destination node. The dirty pages is just related transferred memory data in previous iteration. It is to omit redundant transfer. These procedure such as transfer, check and transfer continues as long as existing a remaining memory data which is split before the first iteration. If the STORD transfers last remaining memory data to destination node, the STORD considers a dirty page which is checked at the same time to split memory data using split number. And then, it repeats the procedure. A condition to stop the repeat procedure is same as pre-copy. The repeat procedure stop depending on a dirty page size.

## 2.2 Pseudo Code

Table 1 shows that pseudo code on the proposed method. It conducts a same way as Figure 3. The proposed method implements the line 7-8 one time to get the virtual machine memory size to be migrated. And then, it split the transfer memory data according to predefined memory split number. A count on the line 13 is used to check a remaining memory data. If the count value is same as split number, the proposed method consider that the remaining memory data is not exist and the previous procedure repeats from line 10 And a omitting redundant dirty page is implemented in line 17.

Table 1: Pseudo Code of STORD



```
1  : S : predefined number to split memory size of VM
2  : M[] : splited memory size of VM using split number
3  : Count : check the number of remains memory contents
4  : PM : a memory contents information of previous round
5  : RD : a dirty_page per round
6  :
7  : vm_size ← Get VM_size()
8  : memory contents ← vm_size
9  :
10 : START Transfer the memory contents
11 :   Transfer M ← memory contents / S
12 :   WHILE DO
13 :         Count ← Count + 1
14 :         PM  ← Transfer M + RD
15 :         RD ← Check Dirty page()
16 :         IF mapping some of the part on RD to PM THEN
17 :            RD  ← RD ∩ PM
18 :         ELSE
19 :          RD ← 0
20 :         END IF
21 :         IF Count == S THEN
22 :            M ← RD
23 :          Goto 9
24 :         ELSE IF Stop_condition() == ture THEN
25 :          Break
26 :         ELSE
27 :          Continue
28 :         END IF
29 :     END WHILE
30 : END Transfer the memory contents
```

## 2.3 Relationship of pre-copy and the STORD

Next, we discuss the relationship of pre-copy and the STORD. The analysis would further direct the design of our approach.

First, some notations are defined as following:

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

$R_{dirty}$: the growth rate of dirty pages in the migration virtual machine.

$N_{split}$ : the split number to split transfer memory data.

$F_{tms}$ : the transfer memory data and dirty page size in first iteration of pre-copy.

$S_{tms}$ : the total transfer memory data size during pre-copy.

$F'_{tms}$ : the transfer memory data and dirty page size in first n times iteration of the STORD.

$S'_{tms}$ : the total transfer memory data size during STORD.

If network bandwidth is static, total migration time is direct proportion in total transfer memory data size by migration procedure. Therefore, we compare the total transfer memory data size with pre-copy and the STORD.

The total transfer memory data size with pre-copy is (1). As shown (1), the total transfer memory data size is sum of geometric sequence, first term is $F_{tms}$ and common ratio is

$R_{dirty}$. And if a number of iteration is infinite, we can get (2).

$$F_{tms} = V_{tms}\,, \qquad S_{tms} = \frac{F_{tms}\left(1-\left(R_{dirty}\right)^{n}\right)}{1-R_{dirty}} \tag{1}$$

$$\lim_{n\to\infty} S_{tms} = \frac{F_{tms}}{1-R_{dirty}} \tag{2}$$

In case of the STORD, a transfer memory data and dirty page size in first iteration of pre-copy are respectively represented by vector $F'_{tms} =< f'_2, f'_3, f'_4, \dots, f'_k >$. Also a growth dirty page rate are respectively represented by vector $R'_{dirty} =< r'_2, r'_3, r'_4, \dots, r'_k >$. And $k$ is split number to split transfer memory data.

Similarly the pre-copy methods, the total transfer memory size of the STORD is sum of geometric sequence. But this case, first term and common ratio is different because our approach splits a transfer memory data size. So we can get (3) and (4) as the split number is $k$. Also if a number of $k$ is infinite, $F'_{tms}$ is inverse proportion by split number(3). As noted earlier, the total transfer memory size is sum of geometric sequence so it is represented by (5). It indicates that the total transfer memory size is direct proportion in a split number.

$$f'_k = \frac{\frac{V_{tms}}{N_{split}}\left(1-\left(\frac{R_{dirty}}{N_{split}}\right)^{k}\right)}{1-\frac{R_{dirty}}{N_{split}}}, \quad \lim_{k\to\infty} F'_{tms} = \frac{\frac{V_{tms}}{N_{split}}}{1-\frac{R_{dirty}}{N_{split}}} \tag{3}$$

$$r'_k = \frac{R_{dirty}}{N_{split}} + \left(\frac{R_{dirty}}{N_{split}}\right)^{2}\left(N_{split}-1\right) + \cdots$$
$$+\left(\frac{R_{dirty}}{N_{split}}\right)^{k-1}\left(N_{split}-1\right)\left(N_{split}-2\right)\times\cdots \qquad \times\left(N_{split}-\left(N_{split}-2\right)\right)$$

$$S'_{tms} = \frac{F'_{tms}\left(1-\left(R'_{dirty}\right)^{n}\right)}{1-R'_{dirty}}, \quad \lim_{n\to\infty} S'_{tms} = \frac{F'_{tms}}{1-R'_{dirty}} \tag{5}$$

From (4), we observes that width of decrease of a dirty page rate is the largest when the memory split number is 2. And the dirty page rate increases even though width of decrease decreases when the memory split number.

In the relationship of pre-copy and our approach, we can consider $R'_{tms} < R_{tms}\,,\ F'_{tms} < F_{tms}$. Therefore if a network bandwidth to migrate virtual machine, the total transfer memory size of the STORD is smaller than the pre-copy, it means that the STORD improve the total migration time of pre-copy(6).

$$F'_{tms} < F_{tms}\,, \quad R'_{tms} < R_{tms} \quad \rightarrow \quad S'_{tms} < S_{tms} \tag{6}$$

## 3. Evaluation

To demonstrate the validity of the STORD, we have designed and implemented a modeling based on iterative pre-copy procedure of the conventional method. And we considered a static network bandwidth and dirty page rate to evaluate the proposed method without other factors which impacts on migration performance.

Our test environment consist of a virtual machine with 4GB memory and a network bandwidth 120MB/S on the Ethernet. In particular, the dirty page rates range from 20% to 90%. It used to appraise the STORD in the memory intensive environment.

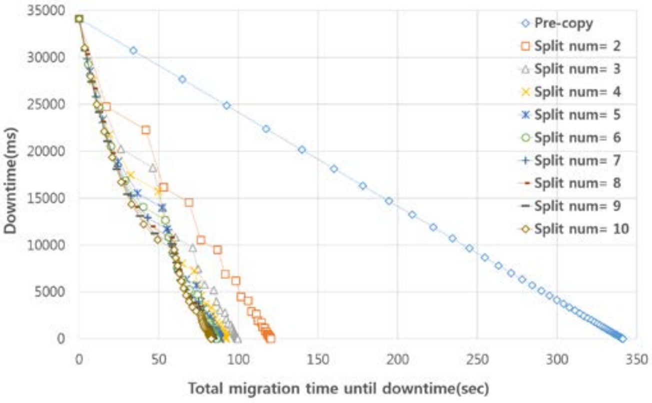


Fig.4 Downtime and total migration time for various split number

The Figure 4 displays downtime and the total migration time for various number in 90% rate of the dirty page. From this data, we observe that the pre-copy takes about 350 seconds with the total migration time. But the STORD takes at most 120 seconds, it saves the total migration time up to minimum 65% rather than the pre-copy.

In case that predefined number to split memory size is two, the downtime according to total migration time shows two type of gradient. If the procedure of split transfer is halted, our method regards remaining a split memory which has not transferred to destination as dirty page. So a falling rate of downtime depends on whether or not a split memory remains.

Also, we observe that the total migration time has little difference even between two and ten split memory. As we mentioned in equation (4) in Section 2, width of decrease of a dirty page rate is the largest when the memory split number is 2. After that, the rate of dirty page decrease. Therefore, additional split number does not dramatically help save the total migration time after two of split number.

The Figure 5 illustrates the rate of total migration time in different dirty page rates. The STORD saves the total migration time up to 65% compare to the pre-copy in 90% rate of the dirty page. And in case of 50%, 20%, it reduces respectively 20%, 10%. It indicates that the STORD improves migration performance in the memory intensive environment rather than another. Because the size of dirty page during each iteration is bigger in the memory intensive environment.

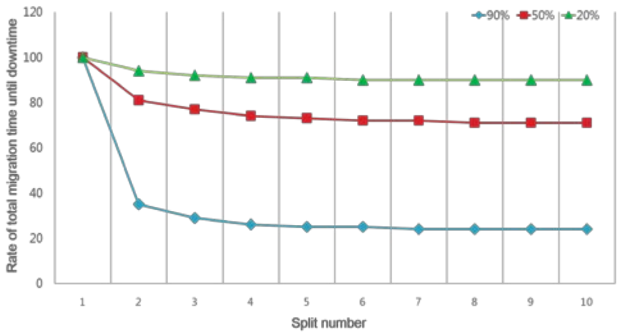


Fig.5 Rate of total migration time for various number of memory partition

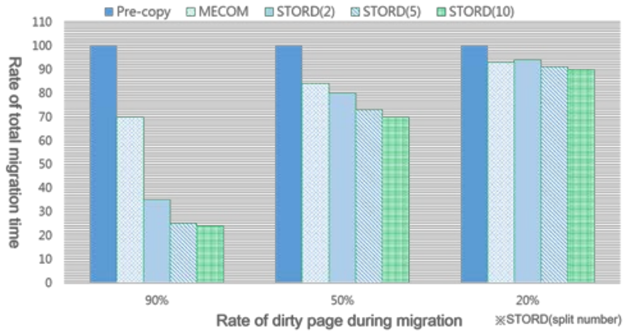


Fig.6 Rate of total migration time for different dirty page rate during migration

The Figure 6 shows the result of the STORD compare to a previous improved method of pre-copy which is memory compression (MECOM) by Jin [9]. From the data, if the rate of dirty page decreases, the performance gap between our approach and MECOM decreases. Even though MECOM's shows 2% performance better than two split memory number in 20% rate of dirty page. Because MECOM has compression and decompression in every iteration and that time increase according to a transmission size of memory data.

## 4. Conclusion

This paper describes the improved method of live migration to acceleration a virtual machine migration in the memory intensive environment. Base on dirty page characteristics, we designed splitting memory data for live migration VM. Because the dirty page occur depending on transmission data size when the network bandwidth is static. The study indicated that our approach saves the total migration time by at least 65% and at most 75% in the memory intensive environment. This is in contrast to previous work in which, in the memory intensive environment, migration takes a long time or reaches a deadlock [6],[8],[9],[10]. In same environment, our approach can reduce more the total migration time compared to MECOM as much as 40%. Because MECOM has a overhead such as compression or decompression and the overhead increase in the memory intensive environment. In the future, we will extend the method to real-migration environment. Furthermore, we plan to apply our approach to migration on wide area network(WNAs). Because migration in WAN should provide a way to keep the network connection [11]. Several method has been studied such as IP tunneling, Virtual Private Network(VPNs) [11]. Also, WAN has low available bandwidth so this is a challenge of migration in the memory intensive environment.

## References

[1] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt and A. Warfield, "Live Migration of Virtual Machine," 2nd Symposium on NSDI, pp.273-286, 2005.

[2] V. Medina and J. Manuel, "A Survey of Migration Mechanisms of Virtual Machine," ACM Computing Surveys(CSUR), vol.46, 2014.

[3] M. Hines and K. Gopalan, "Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning," 2009 ACM SIGPLAN/SIGOPS conf. on VEE 09, pp.51-60, 2009.

[4] K. Kambatla, G. Kollias, V. Kumar and A. Grama, "Trend in Big Data Analytics," Journal of Parallel and Distributed Computing, vol.74, pp.2561-2573, 2014.

[5] S. Oliveira, K. Furlinger and D. Kranzlmiiller, "Trend in Computation, Communication and Strage and the Consequences for Data-Intensive Science," 2014 IEEE 14th International Conference on HPPC-ICESS, pp.572-579, 2012.

[6] K. Z.Ibrahim, S. Hofmeyr, C. Iancu and E. Roman, "Optimized Pre-copy Live Migration for Memory Intensive Applications," 2011 International Conference for HPC, No.40, 2014.

[7] A. Alhammad and R. Pellizzoni, "Schedulability Analysis of Global Memory-Predictable Scheduling," the 14th International Conference on Embedded Software, pp.12-17, 2014.

[8] F. Ma, F. Liu and Z. Liu, "Live Virtual Machine Migration based on Improved Pre-copy Approach," 2010 IEEE ICSESS, pp.230-233, 2010.

[9] B. Hu, Z. Lei, Y. Lei, D. Xu and J. Li, "A Time-Series Based Precopy Approach for Live Migration of Virtual Machines," the 2011 IEE 17th ICPDS, pp.947-952, 2011.

[10] H. Jin, L. Deng, S. Wu, X. Shi and X. Pan, "Live Virtual Machine Migration with Adaptive Memory Compression," CLUSTER 09 IEEE ICCC, pp.1-10, 2009.

[11] R. Bradford, E. Kotsovinos, A. Feldmann, H. Schioberg, "Live Wide-Area Migration of Virtual Machines including Local Persistent State," the 3rd ICVEE, pp.169-179, 2007.