Flexible Bandwidth Allocation Algorithm for Virtual Machine Live Migration

Shuang Kai and Chen Yan

***Abstract*—Virtual machine(VM) live migration is a core feature of virtualization which enables dynamic resource requirements to be matched with available physical resources, leading to better performance and reduced energy consumption. A group of pages have been mapped writable in response to the migration algorithm, however there lacks a algorithm to choose the matching algorithm dynamically based on the upper running features. This paper propose FBA algorithm which can first divide all the migrations into two types "sluggish dirty page" and "swifting dirty page" by monitoring the value of *skip_to*, moreover this paper present corresponding corrective strategies for migrating the two types of application. Finally via simulation and experiments with real system prototypes, the results show that when it belongs to "swifting dirty page", compared with existing pre-copy algorithm FBA algorithm improve the bandwidth utilization and reduce the total migration time; while for the "sluggish dirty page", the pre-copy algorithm will fail and FBA algorithm makes a beneficial supplement.**

***Index Terms*—VM live migration, FBA algorithm, swifting dirty page, sluggish dirty page.**

I. Introduction

Virtualization technology has become commonplace in modern data centers and cluster systems due to its capability of isolation, consolidation and migration workload [1]. In particular, the capability of virtual machine(VM) live migration which can transfer an active VM from one physical host to another without perceivable interruptions brings multiple benefits such as load balancing, improved manageability and fault tolerance [2].

However, the resource consumption and latency of VM live migration reduce these benefits to much less than their potential. For this reason, the optimization of VM live migration is highly desirable [3].

Existing live migration algorithm like pre-copy algorithm [4], its overriding goal is to keep downtime small by minimizing the amount of VM state that needs to be transferred during downtime. However downtime and application performance are likely to be affected in different ways for different applications due to varying memory usages and access patterns, so this algorithm not applicable to all scenarios and may fail or cause lower performance in some special scenarios [5].

Manuscript received December 26, 2015; revised May 27, 2016. This work was supported in part by National Key Basic Research Program of China (973 Program) (2011CB302506). And National Natural Science Foundation under grant 61170274.

The authors are with State Key Laboratory of Networking & Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China (e-mail: shuangk@bupt.edu.cn, 15650713360@163.com).

Aiming at addressing this need, this paper propose FBA (Flexible Bandwidth Allocation) algorithm. This algorithm first put forward Application Recognition Strategy which can divide all the migrations into two types "sluggish dirty page" and "swifting dirty page" by monitoring the value of $skip\_to$, then for the "swifting dirty page", pre-copy algorithm take effect, but the low bandwidth utilization and redundant data transmission will reduce the performance of migration. So this paper present Bandwidth Adaptive Allocation strategy and Page Judgment strategy which can optimize the pre-copy algorithm by improving the bandwidth utilization and delay the transmission of the dirty pages which turn dirty frequently. For the "sluggish dirty page", pre-copy algorithm will fail, while this paper come up with corresponding strategy for this type of migrating, and make a beneficial supplement for the pre-copy algorithm.

Finally, one prototype system is implemented based on Xen virtualization platform according to FBA algorithm [6]. The algorithm has been verified through experiments on various types of load, and results show that compared with pre-copy algorithm, the FBA can matching different strategies according to different upper applications, moreover the corresponding strategies can reduce total amount of data, iterative rounds and total migration time during the transmission effectively.

II. Related Work

The algorithm for VM live migration can be divided into two types pre-copy and copy [7]. The copy can ensure the duplication of each memory page at most once which can avoid occupation of the bandwidth by the redundant data fundamentally, but when the VM runs on the destination host and access to the page which is not synchronized, this algorithm will result in lower performance and longer delay [8]. So the pre-copy algorithm is a predominantly used approach by most existing virtualization platform [9].

The optimization of pre-copy algorithm attracts significant attention from the research community. Sohan [10] propose a Balloon drive mechanism during the preparation phase, it can reduce the idle memory to shorten the time of the first round of iteration, but it will cause the waste of some bandwidth. Gao [11] proposed the page slicing algorithm to overcome the repeatedly copy of the same dirty page during the iteration, but it will cause large resources. Jin [5] proposed data compression algorithm to send every round of pages compressed, but it will cause more time due to the compression and decompression process additional. Sun [12] using Markov prediction model to optimize the selection of working set in pre-copy mechanism, in order to reduce the

transmission of redundant data, but this method limited to the high rate of dirty pages only. Riteau [13] proposed a mechanism called Shrinker, through examination of memory pages to avoid repeatedly sending copies of the same content, thus reducing the amount of data and time during the migration, but this method can not change bandwidth allocation dynamically according to business characteristics.

Based on the work above, this paper propose FBA (Flexibility Bandwidth allocation) algorithm which can divide the migrations into two types according to a key value $skip\_to$. One type called "sluggish dirty page", the other called "swifting dirty pages". For the two different types this paper also give different strategies. Finally, the paper builds the prototype system based on Xen source code, and the results show that compared with existing pre-copy algorithm, FBA algorithm have better performance and make a useful supplement in some scenarios where the pre-copy algorithm may fail.

III. FBA Algorithm Design

FBA(Flexible Bandwidth allocation) algorithm focus on the optimization of VM live migration, this paper first propose Application Recognition strategy which can divide all the migrations into two types "swifting dirty page" and "sluggish dirty page". Then this paper comes up with corresponding strategies for each of the types.

A. Application Recognition Strategy

to_skip collects the identification of dirty pages from the end of the last iteration to the start of a new iteration, which is related to the upper application type. By observing to_skip bitmap marked 1 (refers to the page turn dirty), it can divide the live migration of VM into two types.

The first type called "swifting dirty page", the decline trend of bitmap marked 1 is stable and obvious so the number of second round iteration is less than the first round. The second type called "sluggish dirty page", the bitmap marked 1 keep a very slow downward trend as whole, in some scenarios it will produce even larger fluctuations. So the number of second round is more than the first round. Therefore through comparison between the number of to_skip bitmap marked 1 in the first and second round, this paper can judge the type of upper application.

Steps are as follows:

1) Collecting the to_skip bitmap marked 1 at two key moments:
   - Before the first round of copy called $skip\_first\_iter$;
   - The beginning of the second round called $skip\_second\_iter$;
2) Analyze the changing trend of to_skip bitmap marked 1.

The critical formula is:

$$skip\_second\_iter = skip\_first\_iter \times \sqrt[N-1]{\mu/M} \quad (1)$$

If

$$skip\_second\_iter < skip\_first\_iter \times \sqrt[N-1]{\mu/M},$$

the migration is the first type called "swifting dirty pages".

If

$$skip\_second\_iter \geq skip\_first\_iter \times \sqrt[N-1]{\mu/M},$$

the migration is the second type called "sluggish dirty page".

The following two are the corresponding strategies for each of the types.

B. Strategy for "Swifting Dirty Page"

"Swifting dirty page", as the analysis above, the rate of dirty pages is not very high and produce a stable decline gradually, in this situation the existing pre-copy algorithm take effect.

There are two phase in the pre-copy algorithm, "Push" phase and "Stop-and-copy" phase. The migration process is: the VM transmit the memory pages from the source host to the destination host in "Push" phase until the following four rules are all satisfied then enter into "Stop-and-copy" phase [5].

1) Non-proliferation: the dirty pages transmitted this round ($sent\_this\_iter$) exceeds the last round ($sent\_last\_iter$) and the transmission rate exceeds the maximum value of MAX_MBIT_RATE.
2) Limited cycle: no more than the maximum number of iterations MAX_ITER.
3) Focus: the dirty pages transmitted of current round less than P_{max}.
4) Non redundant: the total transmission pages is no more than 3 times of the amount of memory for the client operating system [14].

When entering into "Stop-and-copy" phase, the VM stop running and transmit all the pages left behind to the destination host at one time. Later start the VM in the new VM and release the resources in the old VM after running normally. This is the whole process of the migration [5].

The pre-copy algorithm is: every iteration is adjustment granularity, the bandwidth of a new iteration is the previous dirty rate plus 50Mbit/sec. The dirty rate refers to the total dirty pages produced in the transmission process divided by the duration of this iteration copy. When reaching the default upper limit, the bandwidth is no longer growth. Therefore, in the "Stop-and-copy" phase the bandwidth is maximum while the first iteration is minimum. The core formula is [10]:

$$B_i = DirtyPages_{i-1} \div T_{i-1} + 50 \quad (2)$$

The existing problems are as follows:

(1) The bandwidth is monotonically increasing, it can not change according to the non-monotonic variation of dirty rate, so the bandwidth utilization is low.

(2) This algorithm can not distinguish the pages which turn dirty frequently, so it transmit this pages many times resulting in a larger amount of data transmission.

(3) The amount of data transmitted in the first iteration

accounts for a large proportion, but the bandwidth is minimum. While in the "Stop-and-copy" phase the amount of data is small but the bandwidth is maximum. Therefore, the allocation of the bandwidth is not reasonable, result in longer transmission time.

In this section, this paper proposes two strategies to address these problems.

1) Bandwidth adaptive allocation

The constant values used in the VM live migration include: M, The memory size allocated by the client operating system; T_{max}, Maximum number of migration times; μ, when the number of dirty pages of one round below μ, the phase of "Push" will stop; I_{max}, Maximum number of iteration times.

The variables values used in the VM live migration include: $dirtyPages_i$, No *i* the dirty pages produced in the No *i* iteration; B_i, the bandwidth of migration in the No *i* iteration; t_i, the time cost by the No *i* iteration; P_i, the rate of dirty pages in the No *i* iteration, reference formula:

$$P_i = dirtyPages_i \div t_i \tag{3}$$

Thus, the time of the No *i* iteration is:

$$t_i = dirtyPages_{i-1} \div B_i = \frac{p_i \times t_{i-1}}{B_i} \tag{4}$$

Because the number of memory in the first round is almost *M*, so:

$$t_1 = M \div B_1 = \frac{M}{B_{\min}} \tag{5}$$

After successive approximation, the expression of t_i can be obtained:

$$t_i = \frac{M}{B_i} \times \prod_{k=1}^{i-1} \frac{p_k}{B_k} \tag{6}$$

The whole process is made up of "Push" phase and "Stop-and-copy" phase, so the time is:

$$T_{pre-copy} = T_{push-phase} + T_{stop-phase} \tag{7}$$

$$T_{push-phase} = \sum_{i=1}^{n} t_i = M \times \sum_{i=1}^{n} \frac{\prod_{k=1}^{i-1} p_k}{\prod_{k=1}^{i} B_k} \tag{8}$$

$$T_{stop-phase} = dirtyPages_n \div B_n = \frac{dirtyPages_n}{B_{\max}} \tag{9}$$

From the formula (6) and (8), the ratio of the dirty page rate and the bandwidth is the key factor that affects the performance of the "Push" phase. For the "Stop-and-copy" phase, the downtime is proportional to the number of dirty pages in the last iteration. While the dirty pages produced this iteration affected by the ratio also. So P/B is the key factor that affect the performance of the migration.

Due to the Non-proliferation and Focus rule, the number of dirty pages should be descended to μ before NO I_{max} round at latest. So the critical formula is:

$$P_{I_{\max}-1} \times t_{I_{\max}-1} \le \mu \tag{10}$$

Substitute the formula (4) to $t_{I_{max}}$, this inequality turn to:

$$M \times \prod_{k=1}^{I_{\max}-1} \frac{p_k}{B_k} \le \mu \tag{11}$$

Therefore controlling Bi can get the best fit relationship between P/B:

$$P/B \le \sqrt[N-1]{\mu/M} \tag{12}$$

According to the formula (12), adjust the bandwidth for each round of the iteration, make P/B less than $\sqrt[N-1]{\mu/M}$, So the bandwidth resource cost by per-round is dynamically changed according to the change of the dirty rate .

$$B = p \div \sqrt[N-1]{\mu/M} = \frac{dirtyPages_0}{\Delta t} \div \sqrt[N-1]{\mu/M} \tag{13}$$

At initial phase, the bandwidth of the first round can be calculated according to the formula (13), avoid the data transmitted of the first round is far greater than that of the following, but the bandwidth allocated is less than the successive rounds. This strategy reduce the difference between the amount of data in the front and rear rounds, as well as the total migration time.

2) Page judgment

The "pre-copy" algorithms is unable to distinguish the pages which turn dirty frequently, which result in the transmission of redundant data and longer migration time. While this paper put forwards page judging, the core idea is distinguish those pages and delay its transmission in order to reduce the total amount of data and the time of migration.

The principle of pages judging such as shown in Fig. 1, according to three bitmap to_skip (the dirty pages produced in the current iteration which can be skip over without transmission), to_send (the dirty pages produced last iteration), and to_fix (the pages which will not be transmitted until the "Stop-and-copy" phase) to decide which page should be transmitted in this round.

At beginning, set to_send to 1 that is to say that all the pages should be transmitted and clear the bitmap of the shadow page .When it is time for the first round of iteration, firstly collect the new dirty page bitmap to to_skip and then judging the pages, filtering the pages which turn dirty in a short time. Because this pages update frequently so it may become dirty in the next round of iteration ,and it may also turn dirty in this short period and need to be transmitted next round of iteration.

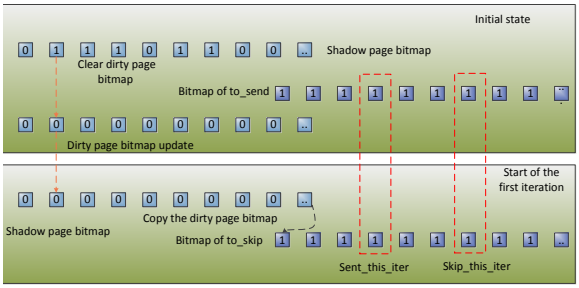

Fig. 1. Page judgment.

From the second round, the steps are as follows:

Obtain the value of to_send at the end of No $i-1$ round of iteration, regard it as the complete works of the target pages which will be transmitted, namely the to_send bitmap marked 1. After that, clear the bitmap of the shadow page.

(1) Obtain the value of to_skip at the beginning of the No i round, identifying the pages become dirty once again from the end of the No $i-1$ round to the start of No i round. Because this pages may turn dirty frequently so can not be transmitted this round of iteration.

(2) For the pages marked 1 in to_send bitmap, if it is marked 1 also in to_skip bitmap, that is to say it turns dirty in a short time, so it will not be transmitted in the current round. Then record it in $skip\_this\_iter$. If to_skip is 0, then transmit it this round and record it in $sent\_this\_iter$.

From the analysis above, these two strategies can allocate the bandwidth more reasonable and avoid redundant data transmission in order to reduce the total migration time and the amount of data transmission. To some extend it solve the problems exist in the pre-copy algorithm.

C. Strategy for "Sluggish Dirty Page"

Most migrations belong to the "swifting dirty page", but when the migration belongs to "sluggish dirty page" the rate of dirty page may be very high, even if the bandwidth is maximum, it is unable to catch up with the speed of dirty pages' changing so the pre-copy algorithm will fail. Therefore in this scenario this paper come up with another strategy to solve this problem, there are two methods can be used as follows:

(1) Adjust bandwidth, accelerate the end of "Push" phase

In the second round of iteration, adjust the bandwidth directly to maximum in order to minimize the dirty pages produced, then enter the "Stop-and-copy" phase and transmit the rest dirty pages in memory.

(2) Adjust weight of CPU, reduce the read/write speed of memory.

Jin [5] proposed that through adjusting the allocation strategy of virtual CPU appropriately, it can reduce the memory read/write speed of the VM.

Virtual CPU scheduling refers to the determination made by the virtual monitor that which client operating system can use physical CPU. Virtual CPU scheduling requires making full use of the CPU resources and allocate CPU accurately at the same time.

By establishing a model, this paper puts forward:

$$w_0 = \frac{e}{1-e} \times \sum_{i=1}^{n} w_i \tag{14}$$

where w_0 is the weight of the VM, e is the percentage of the time of virtual CPU assigned to the VM, for the sake of avoiding affecting the upper application, this paper pointed out that the e cannot be less than 20%. $w_1,...,w_n$ are the weights of the other virtual machines.

According to the formula above, adapting the weight of virtual CPUs can reduce the read/write speed of the VM to some extent. Then the pre-copy mechanism take effect, combine the Bandwidth Adaptive Allocation and Page Judgment strategies with pre-copy algorithm, better results can be achieved.

Therefore, this strategy can be used when the pre-copy algorithm fails in some special scenarios. So it make a beneficial supplement.

IV. System Implementation

A. Architecture of Prototype System

The monitor layer of Xen realizes the algorithm of the migration. This paper adds Application Recognition module, Bandwidth Adaptive Allocation module and Page Judgment module to distinguish two types of migration and give strategies for different type of migration. The data acquisition and analysis module are added to support the strategy. Shown as Fig. 2 [15].

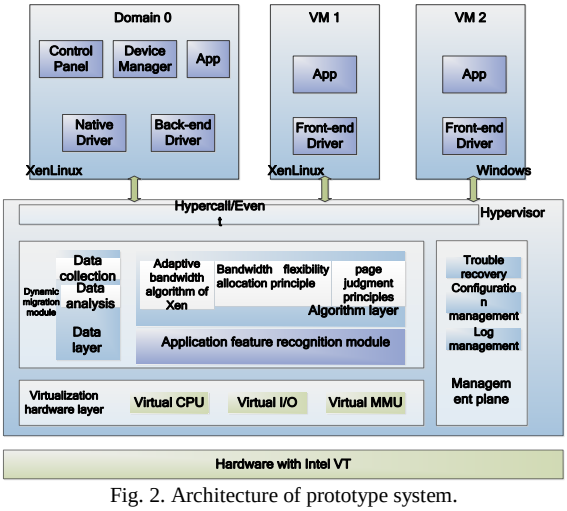

Fig. 2. Architecture of prototype system.

B. Testing Environment

Eight Dell PowerEdge r410 server are deployed in the test platform, four of which work as test hosts, one as a shared storage server, one as a monitor server and the rest two work as SpecWeb installation server. The configuration is shown in Table I.

Test environment is shown in Fig. 3, two LAN switches connect with an convergence layer switch, constitute two independent LAN sub-net, one of the switch connecting several experimental virtual servers, constitute a small experimental data center, another switch connect testing service hosts in Benchmark system.

TABLE I: TESTING ENVIRONMENT

Type	Equipment type		
Hardware	Version	Memory	Network
	DELL R410	2G*4	Gigabit interface *4
Software	Host OS	Version	Virtualization Platform
	Cent OS 2.6.18	Xen-4.1.2	Prototype System
Two layer switch	Version	Bandwidth	Interface
	WS-C2960S-24TS-S	Gigabit	24

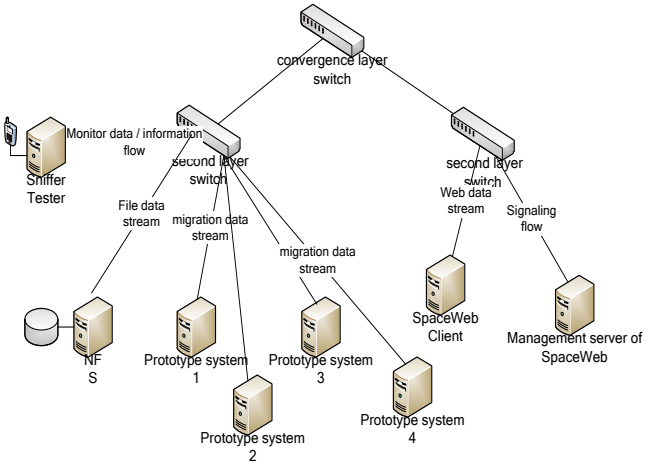

Fig. 3. Test environment.

C. Performance Statistical Tools

The monitoring of the testing platform includes two aspects, on the one hand it needs to monitor the occupation of the underlying resources during the migration, such as computing resources, memory and disk storage resources, and bandwidth consumption etc.; on the other hand monitoring the use of LAN network resources, and the resources allocation of the exchange equipment and the distribution of traffic to analyze the affects made by the VM migration to the network [16].

Here select two standard analysis tools, one is a component of the Linux performance monitoring tool, the other is a sniffer flow detection software, shown as Tables II, III.

TABLE II: PERFORMANCE MONITORING TOOL OF LINUX SYSTEM

Tool	Function	Performance index
mpsts	Multi processor state statistics mpstat [-P {|ALL}] [internal [count]]:	➢ occupation time of user state ➢ Hard disk IO waiting time ➢ Hard interrupt time ➢ Soft interrupt time ➢ Idle time of CPU ➢ Number of interrupt per second received by CPU
vmsts	Status of physical memory and virtual memory	Details of memory,swap
iosts	Use of external storage devices	➢ TPS and Throughput information ➢ Equipment usage, IO response time

TABLE III: PERFORMANCE MONITORING TOOL OF LINUX SYSTEM

Function	Description	Index
Dashboard	Ratio of maximum bandwidth for transmission	Network utilization
	Transmission speed	Data packets per second
	Error rate	Per second error
Protocol Distribution	The occupancy percentage of each protocol packet	The use of network protocols
Host Table	The number of incoming and outgoing packets of the host	Traffic details of each host
Matrix	The information of each host and the data size transmitted between two addresses	Network connection

V. PERFORMANCE EVALUATION

A. Evaluation of "Swifting Dirty Page"

When the upper application belongs to "swifting dirty page", FBA algorithm can identify it and apply pre-copy mechanism in the first two rounds.

1) Test Plan

Create VM on the virtualization platform, the number of vCPU is 2 ~4, Run the CPU intensive application on the VM. Shown as Table IV.

TABLE IV: TEST PLAN FOR "SWIFTING DIRTY PAGE"

No	VM		Configuration		Migration algorithm
	System	Image	vCPU	Memory	/
01	Linux CentOS	10G	2	1024M	FBA/Pre-copy
02	Linux CentOS	10G	2,2 threads run simultaneously, CPU occupation reached 200%	1024M	FBA/Pre-copy
03	Linux CentOS	10G	4,4 threads run simultaneously, CPU occupation reached 400%	1024M	FBA/Pre-copy

2) Test result

a) The relationship between bandwidth and the rate of dirty pages

For the pre-copy algorithm the bandwidth is monotonically increasing refers to the formula (1), and it can not change according to the non-monotonic variation of the rate of dirty pages. So the bandwidth utilization is low. While for the FBA algorithm, the Bandwidth Adaptive Allocation strategy put by this paper solve this problem appropriately. The bandwidth is calculated by applying the formula (13) in FBA realize the changing of bandwidth together with the dirty pages, so the utilization of the bandwidth can be increased .

Fig. 4 and Fig. 5 are the relationship between the bandwidth and the rate of dirty pages of FBA algorithm and pre-copy algorithm. In the figure the abscissa represents the number of iterations, the vertical coordinates represents the number of pages. As shown in Fig. 4, pre-copy algorithm, the rate of dirty page of the fifth round is 50, the sixth round is 185, while the bandwidth is 100 unchanged; the 9th round

is 402, the 10th round is 278, the bandwidth is 207,459 and it did not decrease with the reduce of the dirty pages rate, but increased, reducing the bandwidth utilization. As shown in Fig. 5, FBA algorithm, the rate of dirty page of the eighth round is 404, the 9th round is 289, the 10th round is 911 and the bandwidth allocation is 500,374,500, obviously the bandwidth can change together with the rate of dirty pages, reducing the waste of bandwidth resources and improve the bandwidth utilization.

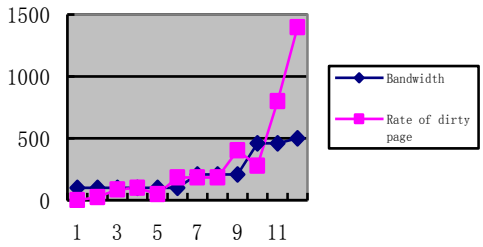

Fig. 4. Relationship between bandwidth and dirty page rate of pre-copy.

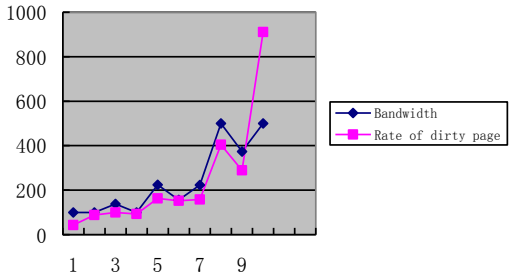

Fig. 5. Relationship between bandwidth and dirty page rate of FBA.

b) The proportion of pages in each transmission

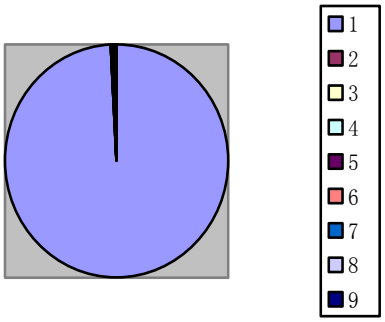

Fig. 6. The proportion of pages for each transmission of Pre-copy.

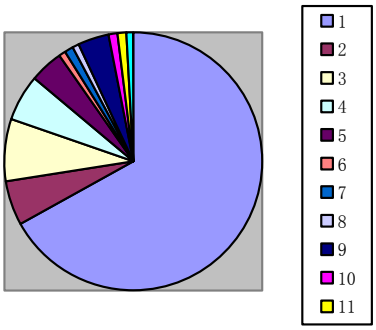

Fig. 7. The proportion of pages for each transmission of FBA.

The pre-copy algorithm as shown in Fig. 6, the first round of iteration accounted for 98%. However, the first round uses the minimum bandwidth, in subsequent iterations increase the bandwidth gradually to transmit fewer pages, finally the "Stop-and-copy" stage uses a maximum bandwidth to transmit a small number of pages, so the total migration time is long.

In this paper, the initial bandwidth of the first round is obtained by the formula (13) in FBA algorithm, which reduce the first round proportion only accounts for 67%. As shown in Fig, 7, compared to the pre-copy, FBA can reduce the total migration time.

c) Total transmission data

The pre-copy algorithm can not identify the dirty pages changed frequently, caused transmission of the redundant data. While the strategy Page Judgment in the FBA algorithm can delay the transmission of these pages, reduce the amount of data ,and solve this problem appropriate.

Fig. 8 shows the total transmission data of pre-copy algorithm and FBA algorithm. The abscissa represent the round of iteration, the vertical coordinates represent the number of pages. For the pre-copy algorithm the total number of pages is 480848 while the FBA algorithm is 340097, thus the FBA algorithm can effectively reduce the redundancy of data transmission, reducing the total data transmission.

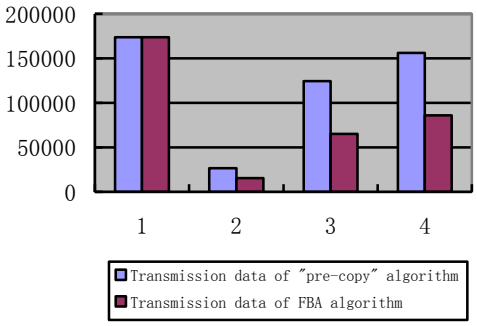

Fig. 8. Total transmission data.

d) The migration time

As shown in Table V, the migration time of the two algorithm is tens of seconds of magnitude, FBA is about 15 seconds less than pre-cop; downtime is in 10 milliseconds magnitude, FBA is 3 milliseconds less than pre-copy; the iterative rounds of FBA is about 1 or 2 rounds less than pre-copy.

TABLE V: THE MIGRATION TIME OF "SWIFTING DIRTY PAGE"

Migration Algorithm	/	migration time	Iteration time	Downtime	Rounds
Pre-copy	01	83001ms	82989ms	11ms	7
	02	82259ms	82099ms	11ms	7
	03	82393ms	82383ms	7ms	11
FBA	01	67765ms	67756ms	10ms	6
	02	67583ms	67575ms	8ms	5
	03	65390ms	65379ms	3ms	10

B. Evaluation of "sluggish dirty page"

When the upper application belongs to the "sluggish dirty pages" type, the pre-copy mechanism failed, the FBA algorithm can identify it and apply the strategy put forward before.

1) Test Plan

Create VM on the virtualization platform, the memory is 1024MB, equal to $25\times10^4 K$ pages. Run the memory intensive applications on the VM. Shown as Table VI.

TABLE VI: TEST PLAN FOR "SLUGGISH DIRTY PAGE"

No	VM		Configuration		Migration aglorithm
	System	Image	vCPU	Memory	/
01	Linux CentOS	10G	4	1024MB, 100MB read/write circularly for 100 times	FBA/Pre-copy
02	Linux CentOS	10G	4	1024MB, 500MB read/write circularly for 100 times	FBA/Pre-copy

2) Test Result

a) The relationship between bandwidth and transmission pages for each round

The pre-copy algorithm will fail in this scenario like Fig. 8 shows that the bandwidth is not adjusted to the peak until the third round. While the FBA algorithm as shown in Fig. 9 the bandwidth is adjusted to the peak immediately, then the "Push" phase stop. Compared with the pre-copy algorithm, transmission pages of FBA algorithm in the second round and third round are reduced by 20%-35%, so the FBA algorithm make a useful supplement.

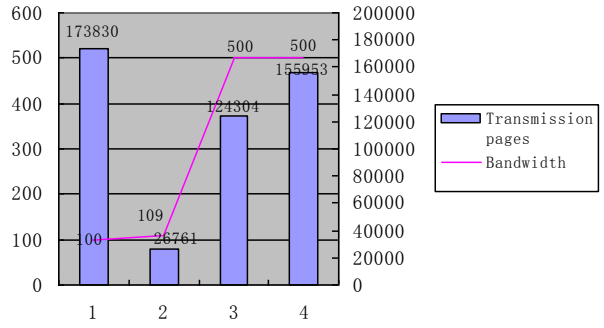

Fig. 8. The relationship between bandwidth and transmission pages of Pre-copy.

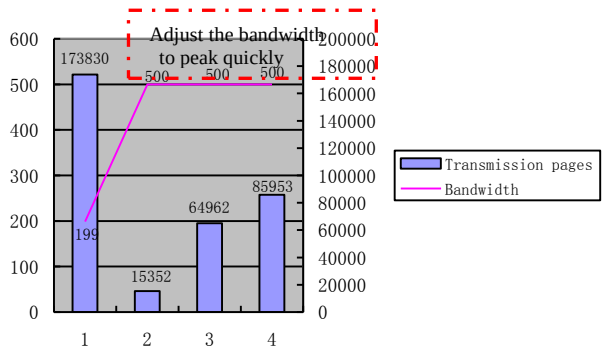

Fig. 9. The relationship between bandwidth and transmission pages of FBA.

b) The migration time

As Table VII shows, for memory intensive applications, the total migration time of the two algorithms is tens of seconds of magnitude, FBA algorithm is about 15 second less than pre-copy algorithm;the downtime is in the order of tens of milliseconds, FBA algorithm is about 6 ms less than pre-copy algorithm; FBA algorithm is about 1 or 2 rounds less than pre-copy algorithm.

TABLE VII: THE MIGRATION TIME OF "SLUGGISH DIRTY PAGE"

Migration Algorithm	/	migration time	Iteration time	Downtime	Rounds
Pre-copy	01	75353ms	75302ms	51ms	5
	02	48232ms	47019ms	74ms	4
FBA	01	60222ms	60177ms	44ms	3
	02	31731ms	31641ms	70ms	3

VI. CONCLUSION

Aiming at the existing VM live migration algorithms being unable to select different algorithms according to different features of upper applications. This paper propose FBA algorithm which can divide all the live migrations into two types called "swifting dirty page" and "sluggish dirty page" by monitoring the value of $skip\_to$. When it belongs to "swifting dirty page", this paper come up with two optimal strategies based on the pre-copy mechanism, the first is "Bandwidth Adaptive Allocation" which avoid the monotonous change of the bandwidth compared with the pre-copy algorithm. The second is Page Judegment which avoid the transmission of redundant data compared with the pre-copy algorithm. While when it belongs to"sluggish dirty page", the pre-copy mechanism will fail, and the strategy come up by this paper can solve this problem appropriately. Finally, the prototype system is implemented in the open source code of the Xen virtualization platform, and the FBA algorithm is verified by various types of scenarios. The results show that compared with pre-copy algorithm, the FBA algorithm not only avoid the failure in some cases that do not suit to the pre-copy mechanism, but also reduce the amount of transmitted data by 20% ~ 35%, shorten the VM rounds 1 to 2, reduce total migration time by more than 15s so it optimize the migration performance.

Future work will focus on the optimization of migration in more complex network environment. In this paper, the realization of the prototype may have other problems possibly not found already, in the future will improved algorithm and optimize the migration performance constantly.

REFERENCES

[1] C. Jo, E. Gustafsson, J. Son *et al*., "Efficient live migration of virtual machines using shared storage," *ACM Sigplan Notices*, vol. 48, no. 7, pp. 41-50, 2013.

[2] H. Liu, H. Jin *et al*., "Live virtual machine migration via asynchronous replication and state synchronization," *IEEE Transactions on Parallel & Distributed Systems*, vol. 22, no. 12, pp. 1986-1999, 2011.

[3] C. Clark, K. Fraser *et al*., "Live migration of virtual machines," in *Proc. Acm/usenix Symposium on Networked Systems Design & Implementation*, 2005, vol. 2, pp. 273-286.

[4] A. Strunk and W. Dargie, "Does live migration of virtual machines cost energy?" in *Proc. IEEE International Conference on Advanced Information Networking and Applications*, 2013, pp. 514-521.

[5] H. Jin, L. Deng, S. Wu *et al*. "Live virtual machine migration with adaptive, memory compression," in *Proc. International Conference on Cluster Computing & Workshops*, 2009, pp. 1-10.
[6] Z. Xiang, H. Zhigang, M. Jie *et al.*, "Exploiting data deduplication to accelerate live virtual machine migration," in *Proc. the 2010 IEEE International Conference on Cluster Computing*, Heraklion, Crete, 2010, pp. 88-96.
[7] J. Zheng, T. S. E. Ng, K. Sripanidkulchai *et al.*, "Pacer: A progress management system for live virtual machine migration in cloud computing," *IEEE Transactions on Network & Service Management*, vol. 10, no. 4, pp. 369-382, 2013.
[8] M. Nelson, "Virtual machine migration," US Patent 8260904 B2, 2012.
[9] D. Haikney, S. P. Mullen, and J. W. Walker, "Virtual machine migration," US Patent 20120159634 A1, 2012.
[10] R. S. Rice, A. Moore, and A. W. A. Hopper, "Predicting the performance of virtual machine migration," in *Proc. the Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, Florida, USA, 2010, pp. 37-46.
[11] G. Xiang, "The optimization of virtual machine migration based on Xen," HIT, 2010.
[12] G. F. Sun, "The optimization of VM live memory migration based on pre-copy mechanism," *Computer Engineering*, vol. 37, no. 13, pp. 36-39, 2011.
[13] P. Riteau, C. Morin, and P. T. Shrinker, "Efficient live migration of virtual clusters over wide area networks," *Concurrency Computat: Pract. Exper.*, vol. 25, pp. 541-555, 2013.
[14] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *Proc. ACM Sigplan/SIGOPS International Conference on Virtual Execution Environments*, 2009, pp. 51-60.
[15] M. Zhao and R. J. Figueiredo, "Experimental study of virtual machine migration in support of reservation of cluster resources," in *Proc. the 2nd International Workshop on Virtualization Technology in Distributed Computing*, ACM, 2007, pp. 1-8.
[16] V. Shrivastava, P. Zerfos, K. W. Lee *et al*., "Application-aware virtual machine migration in data centers," in *Proc. INFOCOM*, 2011, pp. 66-70.

Shuang Kai graduated in computer science and technology from Beijing University of Posts and Telecommunication and received his Ph.D. degree. He is currently working as an associate professor at State Key Laboratory of Networking & Switching Technology. His research interests include cloud computing, mobile Internet and big data.

Chen Yan majors in computer science and technology and did her master program at the State Key Laboratory of Networking & Switching Technology. Her researches interests include cloud computing and mobile Internet.