Optimizing Live Migration of Virtual Machines with Context Based Prediction Algorithm

Yong Cui[1,2], Yusong Lin[1], Yi Guo[1,2], Runzhi Li[1], Zongmin Wang[1]
[1]Henan Provincial Key Lab on Information Networking, Zhengzhou University, Zhengzhou, China
[2]Institute of Information Engineering, Zhengzhou University, Zhengzhou, China
e-mail: {cuiyong, lys, guoyi, lrz, zmwang}@zzu.edu.cn

***Abstract*—With the increasing use of Virtual Machine (VM) in data center, live migration of virtual machine has become a powerful and essential instrument for resource management. Although the prevailing Pre-copy algorithm might perform well on the stage of lightweight, it cannot guarantee a desirable performance in the case of high dirty page rate or low network bandwidth. The resending problem results in striking performance degradation and waste of resource. Toward this issue, this paper presents a novel Context Based Prediction algorithm (CBP), which exploits PPM (Prediction by Partial Match) model to predict the dirty pages in the future iteration based on the historical statistics of dirty page bitmap. The transmissions of those frequently updated pages identified are postponed. Experiments demonstrate that CBP can achieve a satisfying balance between accuracy and overload, and shorten total migration time, downtime and total pages transferred significantly, comparing with KVM's default algorithm.**

Keywords-Virtual Machine; Live Migration; Context Based Prediction; Performance Evaluation

I. INTRODUCTION

Live migration is one of the key features of Virtual Machine technology, which provides the server administrator with a mechanism that moves a virtual machine from one physical server to another while the tasks inside the VM are continuously running[1]. It can facilitate hardware online maintenance, real-time load balancing, fault tolerance and power saving in data center.

As share storage (such as SAN or NAS) is commonly adopted in data center, among the live migration process, memory migration takes the primary part. Prevailing Virtual Machine Monitor (VMM) or Hypervisor, e.g. KVM [2], mainly leverage the state of the art Pre-copy algorithm [3] to carry out live memory-to-memory migration. The goal of Pre-copy is to migrate the VM in a short total migration time and downtime, and occupy less network bandwidth. Although Pre-copy could obtain a relatively remarkable downtime reduction by leveraging iterative memory pre-copy, its performance heavily depends on page dirty rate and network bandwidth [1]. Thus, in the case of write-intensive workload or low bandwidth (e.g. WAN), Pre-copy retransmits the same dirty pages many times during migration, which can lead to an extended total migration time and downtime, also waste of network bandwidth.

In this paper, we present a novel approach CBP to optimize Pre-copy. By exploiting PPM prediction model, we employ the up-to-date statistics of dirty page bitmap as the prediction context to identify the dirty pages in the following iterative rounds. The transmission of those pages with high dirty rate are delayed, which can avoid resending those pages repeatedly during migration, furthermore decrease the redundant transmission overhead and the extended total migration time and downtime.

The rest of this paper is organized as follows. The related work and problem statement is introduced in Section 2 and Section 3. Section 4 describes the design and implementation of the CBP algorithm. The performance results and analysis are present in Section 5, followed by the conclusion and future work in Section 6.

II. RELATED WORK

To address the resending problem in Pre-copy, many approaches have been proposed. Some aim at compressing the transferred page data. Jin et al. [4] propose an adaptive compression based scheme MECOM, which compresses memory page according to its characteristics. Different compression algorithm is applied to pages of different type. Paper [5] utilizes Delta Compression technique to calculate the delta page from current and previous version of a page by XOR, and then compresses the delta page with the help of RLE (Run Length Encoding compression). Zhang et al. [6] use hash based fingerprints to find identical and similar memory pages, and then also employ RLE to eliminate redundant data during transmission. These approaches do increase the throughput and accelerate page sending, but they neglect the retransmission of the same pages updated many times.

Some methods focus on identifying the pages with high dirty rate and postponing their transmissions. Clark et al. [3] introduce an optimization method called "Stunning Rogue Processes" to limit dirty pages transmission, which skips pages that have been dirtied in the previous iteration. Ma et al. [7] propose an improved pre-copy approach. They add a bitmap to mark those frequently update pages and send these pages at the last round. Liu et al.[8] propose a Hierarchical Copy Algorithm to improve Pre-copy, which records the dirty times of pages, and identifies high dirty pages with a preset threshold for the dirty times, then delays the sending of these pages to the last iterative round. Similarly, a times-

series based pre-copy approach [9] introduces a historical dirty pages set. If the count of a page that appears in the set exceeds a preset threshold, this page is not sent until the last round. These methods make decision for sending dirty pages simply based upon limit historical dirty times and ignore the page access pattern. Unlike these approaches, our scheme identifies dirty pages according to the page access context.

Regarding other schemes, Hines et al. [10] propose a novel post-copy strategy to carry out live migration instead of Pre-copy, which defers the memory page sending until after the VM has been resumed in the destination. The memory pages are transferred on demand to avoid the resending problem. Differing from them, our work mainly focuses on Pre-copy.

For predicting by context models, PPM (Prediction by Partial Match) is a widely used model. [11] uses PPM to facilitate the arithmetic coding and then performs data compression. In paper [12], a prediction tree based on PPM is constructed to do web prefetching. Compared to these work, we work in a specialized domain with a different context.

III. Problem Statement

The main procedure of Pre-copy is iteratively transferring memory pages from the source host to the target host. During the first iterative round, all pages are marked as dirty pages, and transferred to destination. Subsequent iterations transfer only those pages modified during the previous transfer round. This iteration would not terminate until the amount of remaining dirty pages becomes enough small to achieve a short stop time, namely downtime. Afterwards, the VM is suspended and the remaining dirty pages, as well as CPU registers and I/O device statuses, are send out to the destination. Then the VM is resumed at the destination. The duration of the whole process is the total migration time.

For iterative page copy, Pre-copy maintains a dirty page bitmap to mark which pages should be sent in each iteration. Given a memory page *p*, if *p* is dirty during the previous round, the bit of *p* in bitmap is set to '1', otherwise is '0'. At each iterative round, Pre-copy first updates the bitmap with the dirty page trace logging, then sends the pages of which the bit in bitmap is '1'. In the case of high dirty page rate, page *p* is written many times, e.g. the historical statistics of the bit of *p* is '0110110101101', which means among the previous 13 iterations, 8 replicates of *p* is copied to destination, but in fact it has the same effect as only transferring the last replica. When there are lots of pages like *p* in this case, this drawback can cause transferring a large number of unnecessary data and lead to a extended total migration time. Meanwhile, if the dirty rate exceeds the network bandwidth, the remaining dirty pages in the stop phase cannot converge to a small size. This will result in an enlarged downtime.

The reason of this drawback is that the default Pre-copy algorithm is unaware of the page access pattern, in which many appliances work commonly, according to the "Principle of Locality". If the pattern is known, we can predict whether a page will be dirty or not in the future. So, it is rational to exploit the historical statistics of dirty page bitmap reflecting the pattern to make a proper transferring decision for dirty pages.

IV. Context Based Prediction Algorithm

In this section, we describe our context based prediction algorithm optimized for Pre-copy in detail. We first demonstrate how to predict dirty page by PPM, and then introduce the implementation of CBP.

A. Predicting Dirty Page by PPM

In general, PPM uses *n*-order Markov model to describe the dataset, and keeps track of the probability of a symbol occurring under the condition of a specific sequence of symbols has already been seen, i.e. a context, where *n* is the length of a context. Prediction is made by the probability of the symbol. In the domain of predicting dirty page, the historical statistics of dirty page bitmap is the dataset, and the symbol set is {0, 1}, and the context is the bit sequence of a page. We introduce a two dimensional array *Bitmap_record* to present the above dataset. Given a page *p*, *Bitmap_record* [*p*][] indicates the historical statistics of the bit of *p*. We define the size of *Bitmap_record*[*p*][] as *m*. For a context *C* with the length of *n* ($n \in [0, m]$), Let C_0 = occurring times of *C* followed by '0' in *Bitmap_record*[*p*][], C_1 = occurring times of *C* followed by '1' in *Bitmap_record*[*p*][] and *P* is the probability of being dirty page, then we predict *p* is a dirty page in context *C* as:

$$\left(P = \frac{C_1}{C_0+C_1} \times 100\%\right) > 50\% \qquad (1)$$

Formula (1) means if the occurring probability of '1' following context *C* is greater than that of '0' in the occurred bit sequence, we identify page *p* as dirty in the future.

Also taking the page *p* illustrated in Section 3 as an example, *Bitmap_record* [*p*][] is {0110110101101}, and order *n* ranges from 0 to 13, e.g. order-2 context is {01}, order-3 is {101}. The prediction of *p* with order-2~7 is shown in Table 1.

As can be analyzed from the example, the combination of *m* and *n* has a strong impact on the accuracy and performance. Clearly we can get a more exact result with a larger *n*, but if *m* is not large enough, the *n*-order context would easily lead to obscure results (e.g. order 4~6 in Table I) or mismatch (e.g. order 7) due to lacking of sufficient occurrence. However, a lager *m* requires more resources utilization. Therefore, it is rational to confirm a effective *n* with the given *m*, in the purpose of achieving a trade-off between accuracy and performance. To address this issue, we propose a formula to determine the proper *n* as following:

For predicting a page *p*, Let C_i is the occurring times of the *i*-order context ($i \in [0, m]$) among *Bitmap_record* [*p*][], the desired order *n* is figured out as:

$$n = max\{\, i \mid C_i \geq 3 \cap i \in [0, m] \,\} \qquad (2)$$

Formula (2) indicates we choose the context with enough frequency and length. Therefore, order-3 is the right choice to do prediction in above example. In regard of the variable *m*, we will give an evaluation later in Section 4.

TABLE I. EXAMPLE OF PREDICTING DIRTY PAGE BY PPM

Order	Context	Probability of being dirty	Prediction result
2	01	75% (3/4)	Dirty
3	101	67% (2/3)	Dirty
4	1101	50% (1/2)	NOT Dirty
5	01101	50% (1/2)	NOT Dirty
6	101101	0 (0/1)	NOT Dirty
7	0101101	mismatch	mismatch

B. Algorithm Implementation

The proposed CBP algorithm utilizes the context based prediction model mentioned above to optimize the pre-copy based live migration. Based on the historical statistics of dirty page bitmap, we exploit PPM to identify the modified frequently pages in the following round and delay the transmission of these dirty pages to avoid the resending problem. With this solution, the total migration time, downtime and total pages transferred can be evidently decreased. We added the codes of CBP in KVM 0.14.0. As shown in Table II, CBP mainly comprises of two phases:

TABLE II. PSEUDO CODE OF CBP

```
Input:
Bitmap_record: the historical statistics of dirty page bitmap
m: the record size for each page

 open dirty page logging;
 for 1 to m {
        bitmap = get dirty page bitmap;
       update Bitmap_record with bitmap;
 }
for each iteration {
       bitmap = get dirty page bitmap;
       update Bitmap_record with bitmap;
       for each page{
           if ( bitmap[page] = 1 ) {
               is_dirty_next_round=
                   do_PPM(Bitmap_record[page][]);
               if(! is_dirty_next_round ){
                    send page;
               }
           }
}
```

a) Trace and Record. When the migration is launched, we open the dirty page logging mechanism at the same time to trace the write-operations for each memory page. According to the record size for each page, i.e. argument *m*, we record the dirty page bitmap into Bitmap_record at a regular interval which forms the historical statistics for prediction. This process finishes in a short period of time so as not to put off the total time. Not confined to this, in order to keep the statistics data available and up to date, we also update Bitmap_record at the beginning of each iteration with the dirty page bitmap mapped for page sending in a FIFO pattern.

b) Predict and Judge. For each iteration, when a page of which the mapping bit is '1' in the bitmap of this round is ready for transmitting to the destination, we intervene and predict whether the page will be dirty in the following round according to its access context, as stated in the previous section. If the probability of being dirty is larger than that of not, the page is delayed to send, otherwise, we transfer it as normal.

V. PERFORMANCE EVALUATION

To evaluate the performance of CBP, we first estimate its accuracy and overload, and assess the parameter *m*. Then we carry out a series of live migration experiments with differently configured VMs to compare our CBP with standard KVM algorithm, in terms of total migration time, downtime and total pages transferred.

A. Experimental Setup

Tests are conducted in three physical machines with the same hardware configuration. Each host has an Intel Core 2.93GHz dual processor and 4G RAM. The three computers are connected through a Gigabit Ethernet. NFS (Network File System) is installed in one of them as the share storage. Both of host OS and guest OS are Ubuntu Server 11.10, and the virtualization modules in the contrast experiments are standard KVM 0.14.0 and the modified KVM 0.14.0 with our adding CBP codes. Of particular note is that we keep the de-duplication mechanism available both, which only sends one byte instead of the whole page in case of bytes duplication. We run and move only one VM in the hosts.

The different workloads running in the migrated VM are listed as following:

a) Idle: The low dirty-page rate scenario that we migrate a VM without running any workload.

b) Kernel Compiling: We migrate a VM while it is executing kernel compiling, which stands for the high dirty-page rate scenario with a regular page access pattern.

c) Web Server: In this scenario, we migrate a VM providing dynamic web serving. We deploy an open-source web application WordPress in the migrated VM. With 10 concurrent connections from the clients, this represents the high dirty rate scenario with the sweeping write operations.

B. Experimental Results

1) *Prediction Accuracy and Overload:* Figure 1 shows the accuracy and CPU usage of CBP according to various values of *m*, when migrating a VM with 1G RAM running kernel compiling and web server separately. As shown, both of the accuracy and CPU usage are increased as the increasing of *m*, but when *m* exceeds about 35, the accuracy drops, which is probable due to the more outdated contexts with a larger *m*. Moreover, the operation on kernel compiling is more accurate due to its more regular page access pattern. For the trade-off between accuracy and overload, we set *m* 30 in the following tests.

2) *Total Migration Time and Downtime:* From figure 2 and figure 3 we can see the comparisons between traditional Pre-copy and CBP on migrating a VM with 1GB RAM. They show our scheme can shorten the total migration time and downtime at best by 35% and 22% respectively, which in the case of kernel compiling.

3) *Total Pages Transferred:* In this test, we migrate a VM configured differently with RAM of 512M, 1024M, 1536M and 2048M, when each of the three workloads is running in turn. Figure 4 depicts the total pages transferred in these scenarios, in which CBP reduces the memory pages we need to send significantly in case of high dirty page rate.

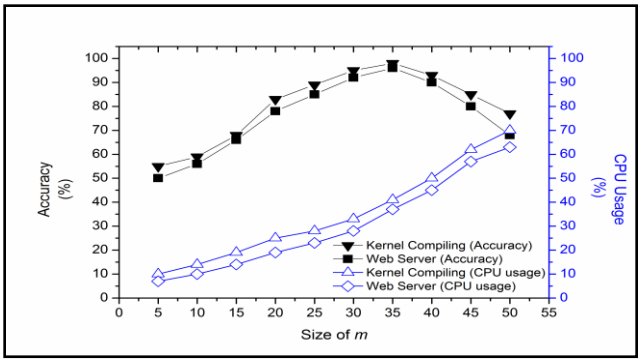

Figure 1. Accuracy and overload of CBP

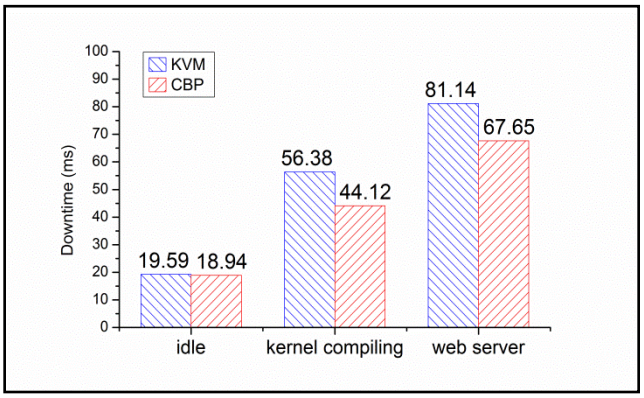

Figure 2. Downtime comparison

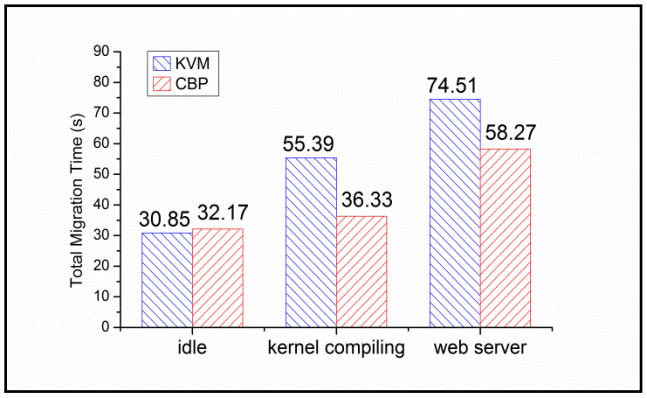

Figure 3. Total migration time comparison

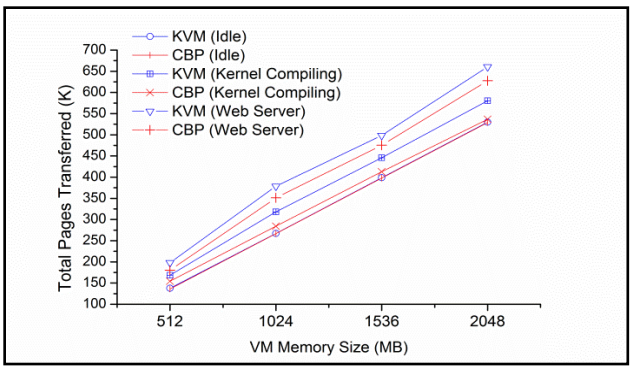

Figure 4. Total pages transferred comparision

VI. Conclusion

In this paper we introduce a novel Context Based Prediction algorithm optimized for the pre-copy based live migration, which exploits PPM model to predict the dirty pages and delays their transferring, avoiding the resending problem of Pre-copy. The experiments demonstrate the effectiveness of our approach. In the future, we will study predicting dirty pages with the contexts of unfixed sizes.

References

[1] M. Nelson, B. Lim, and G. Hutchines, "Fast transparent migration for virtual machines," Proc. The USENIX Annual Technical Conference (USENIX'05), 2005, pp.391–394.

[2] http://www.linux-kvm.org/.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," Proc. The 2nd Symposium on Networked Systems Design and Implementation(NSDI'05), 2005, pp.273–286.

[4] H. Jin, L. Deng, S. Wu, X. H. Shi, and X. D. Pan, "Live Virtual Machine Migration with Adaptive Memory Compression," Proc. The 2009 IEEE International Conference on Cluster Computing, 2009, pp.1-10.

[5] P. Svärd, B. Hudzia, J. Tordsson, "Evaluation of Delta Compression Techniques for Efficient LiveMigration of Large Virtual Machine," Proc. The 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2011, pp. 111-120.

[6] Xiang Zhang, Zhigang Huo, Jie Ma, Dan Meng, "Exploiting Data Deduplication to Accelerate Live Virtual Machine Migration, " Proc. The 2010 IEEE International Conference on Cluster Computing. 2010, pp. 88-96.

[7] F. Ma, F. Liu, and Z. Liu, "Live Virtual Machine Migration Based on Improved Pre-copy Approach," Proc. Software Engineering and Service Sciences, 2010, pp. 230-233.

[8] Z. B. Liu, W. Y. Qu, T. Yan, and H. T. Li, "Hierarchical Copy Algorithm for Xen Live Migration, " Proc. The 2010 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2010, pp. 361-364.

[9] Bolin Hu, Zhou Lei, Yu Lei, Dong Xu, Jiandun Li, "A Time-Series Based Precopy Approach for Live Migration of Virtual Machines," Proc. ICBPDS 2011, 2011, pp.947-952.

[10] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-Copy Live Migration of Virtual Machines," ACM SIGOPS Operating Systems Review, Volume 43 Issue 3, July 2009.

[11] J. G. Cleary and I. H.Witten, "Data compression using adaptive coding and partial string matching", IEEE Transactions on Communications, Vol. 32, No. 4, 1984, pp. 396-402.

[12] T. Palpanas and A. Mendelzon, "Web prefetching using partial match prediction", Proceedings of Web Caching Workshop, March 1999.