Live Migration of Virtual Machine Based on Recovering System and CPU Scheduling

Weining Liu
School of Computer
Chongqing University, Chongqing, 400030, China
Email: lwn_cq@163.com

Tao Fan
School of Computer
Chongqing University, Chongqing, 400030, China
Email: fantaocqu@yahoo.cn

***Abstract*—With the development of distributed computing and cloud computing, live migration of virtual machines across distinct hosts has become a hot research topic.Many previous approaches focused on transferring memory states, which make VM migration cost much downtime.This paper describes a novel approach based on recovering system and CPU scheduling to provide fast, transparent live migration. Target host executes log files generated on source host to synchronize the states of source and target hosts, during which a CPU scheduling mechanism is used to adjust the log generation rate. We also make a mathematical analysis about our approach and give a experiment to evaluate the performance. The experiment shows our approach can reduce the downtime and total migration time compared with pre-copy algorithm: up to 62.12% on downtime and 43.84% on total migration time.**

***Index Terms*—live migration, virtual machine, checkpoint, CPU scheduling**

I. Introduction

Virtual Machine (VM) has become a hot research topic in recent years with the development of distributed computing and cloud computing, although VM technology was implemented on IBM mainframes in 1960s. In cloud computing and data center, virtualization has played an important role in resource management because it abstracts the resources such as storage and CPU through generating VM to support resource assignment[1].

Live migration is a technology that can move a running virtual machine or application among different physical machines without disconnecting the clients or applications. The capacity of migration virtual machines among multiple distributed hosts provides a significant benefit for VM-based environments. VMware, Intel, Microsoft and other companies are researching the live migration of VM, which is a powerful tool used in load balancing, online maintenance, proactive fault tolerance and power management.

In order to resume running the migrated VM on the new host, the states of VM's physical memory, network connection, file system and other resources should be migrated. The most complicated issue is migrating physical memory, because the content of memory is dynamic and it is a main factor that affects the migration performance. Downtime and total migration time are the key performance evaluation metrics of live migration. The term downtime refers to a period of the time when services provided by VM on source host and target host are unavailable. Total migration time is the duration between the time migration begins and the time the migration ends. When the process ends, the VM on the target host gets a consistent states with the one on source host. An ideal situation is that the live migration has the short downtime and total migration time at the same time. But it is hard to achieve that goal. In fact, many migration policies keep balance between downtime and total migration time.

In this paper, we design a novel live migration approach and implement a prototype based on an interactive mechanism that can reduce downtime by CPU scheduling, and ReVirt[2], a full system trace and replay system. Checkpointing/recovery and trace/replay technology[2] are adopted to provide fast, transparent VM migration in a LAN. A trace demon logs the events of source VM. The files are iteratively transferred to target host to synchronize states of the migrated VM. The interaction between source and target hosts can adjust log files growth rate to reduce downtime.

Our major contributions are listed as follows:(1)Based on Checkpointing/recovery and trace/replay technology and an interactive mechanism with CPU scheduling we proposed a live migration approach; (2)We built a model for our approach and analyzed it; (3)We implemented a prototype in Xen environment and provided experimental data for performance evaluation in real applications.

This paper is organized as follows. Section II gives a brief introduction about related work. Section III presents the design of our live migration algorithm and gives analysis about our approach. The experiments are undertaken and results can be obtained in section IV. And finally, we conclude our work in section V.

II. Related Work

A classic approach of migration is stop-and-copy. Many famous systems implemented this approach, such as Internet Suspend/Resume(ISR)[3,4] and Collective[5,6], just stop the VM and copy the state data, then resume the VM on the destination host. Stop-and-copy causes the applications unavailable during the migration process. Collective system applied Copy-on-Write technology, which records and transmits the update information, to reduce downtime and total data transmitted.

To migrate the VM between hosts in local area network with short downtime, VMotion[7] and Xen[8] used pre-copy

978-1-4244-8625-0/11/$26.00 ©2011 IEEE

algorithm to perform live migration. Pre-Copy is an important algorithm used in live migration and has been described in many papers. The basic idea of iterative pre-copy is, transferring dirty data iteratively until reaches an acceptable dirty data size or a predefined iterative times, and then stops source VM to copy the data to target host, so the downtime could be short. But the rate of dirty data generation is the most important parameter that determines performance of this algorithm. High rate of dirty data generation could lead to an increase rounds of pre-copies and large amount of data to be transmitted. In case when dirty data generation rate is faster than memory copy, the live migration can fail. Jin H et al[9] gave a mathematical analysis of pre-copy algorithm. They modified the basic pre-copy algorithm to adjust certain time slices of VM in each pre-copy round.

Many other approaches can improve the performance of live migration. Memory balloon[10] is used widely, which could eliminate unused memory to save time for the first pre-copy round. Hash-Based Compression[5] sends the hash of memory page before transfer and the memory page existed on target host could not be copied. Clark[1] listed some other approaches, such as Dynamic Rate-Limiting and Rapid Page Dirtying.

Recovering systems use checkpointing/recovery and trace/replay technology. The basic concept is simple: starting from a checkpoint of prior state, then rolling forward using log files which recorded events that can affect the system's computing to reach the desire states. ReVirt[2] is a typical full system logging and replay tool ported on UMLinux. The logged information helps system replay a long term execution of virtual machine. Replay can be executed on any host with the same type of processors as the source physical host and there is no any deviation generated during the replay process. In addition, ReVirt adds acceptable time and space overheads. Logging adds up to 8% performance overhead. The I/O intensive workloads generate feasible log traffic at a rate of hundreds of kilobyte per second. Some approaches learned from idea of checkpointing/recovery and trace/replay. Bradford[10] gave a live migration algorithm that combined trace/replay with pre-copy. CR/TR-Motion[11] is also based on checkpointing/recovery and trace/replay. But it only has simple and inefficient interaction between source and target VM.

Keeping the services running and reducing the downtime while the VM is being moved to another machine are considered as a primary goal to achieve for live migration. Some approaches lack efficient interaction mechanism, some need to modify the OS running services and some just passively adapt to dirty memory rate and can not make any adjustment. Our solution uses a similar mechanism as checkpoint and execution log files, during migration a CPU scheduling is used to limit the log rate of VM, which helps to improve performance of live migration.

III. Algorithm design and analysis

This section describes our live VM migration approach based on checkpointing/recovery and trace/replay technology with CPU scheduling. A interaction and synchronization protocol is introduced to show the details of live migration. We also give a formalized characterization about the evaluation metrics and make analysis. It will help to understand our algorithm.

A. Algorithm design

Our live VM migration approach is based on instructions execution trace and replay with CPU scheduling. Unlike pre-copy algorithm, what we copied is execution log files of source VM but not the dirty pages, and this could reduce amount of data transferred. Our algorithm reduces downtime by combining a bounded iterative log transferring phase with a short stop-and-copy time which can be reduced by CPU scheduling. In the process of migration, log files are transferred round by round and the log transferred in round n is generated in round $n-1$, of course the checkpoint is transferred in the first round at the beginning of migration. If the process is convergent, the last log file transferred in stop-and-copy is small so that the downtime can be decreased to an acceptable level.

The key to success of pre-copy algorithm is that at the last round of transferring the dirty data size should be small, in other words, the process of iteration should be convergent. The rate of dirty data generation is the most important parameter that determine the performance of algorithm. If the speed of dirty data generation is higher than transfer rate or memory copy rate, e.g., the process of live migration will fail. Like the limitation of pre-copy algorithm, there are some prerequisites for our approach. One is that the log files should be transferred quickly, or log on source host will accumulate. In fact, the bandwidth of local area network can provide enough transfer speed. Anther is log replay rate must be faster than the log growth rate. If this condition is not satisfied, the downtime may be longer than the time transferring checkpoint from source to target host and the process is not convergent which results in the last stop-and-copy round takes much time. So we design a interaction mechanism with CPU scheduling to avoid this situation as possible as we can. We defined a threshold log size value as S_c. If the size of log is larger than S_c, we will reduce the percentage of processor quantum allocated to source VM to slow down the source VM response to events, and thus log generation rate will be reduced. One of the two conditions or both, the size of log generated on source host or accumulated on the target host larger than S_c, will trigger CPU scheduling.

Our CPU scheduling is based on the experiential rule[9] that a certain log generation rate related with events response speed approximately liner increases with the growing speed of VM's execution by host CPU. Assuming a VM is writing log files x KB/s, when we set the CPU timeslice occupied by this VM from 100% to 50%, the VM will write log files with the speed of $x/2$ KB/s. In fact, there are other factors affect the relationship between log rate and execution speed, such as process priority, but in most situation, we consider

that approximately relationship is reasonable to decrease the complexity of our algorithm. Another reason support our approach is, many applications are more sensitive to execution than efficiency, For example, a web server would increase a little respond latency rather than lose connection ready built on and some online game players may feel much better with a slow experience rather than with a frozen one.

The description of our approach of live migration as follows:

(1)Initialization: A selected target host must have sufficient resources to guarantee the requirement of migration. How to select a reasonable host which belongs to load balancing field is not in our approach. Source host makes a request of migration to target host. If target host is ready to receive migrated VM, it will respond with a firm message to the request and the migration will begin.

(2)Make checkpoint and transfer it: An image file of source VM which has the states of system, virtual memory ,CPU registers, virtual disk, et al., is generated on source host. While generating the checkpoint, the source VM is freeze. After generation of checkpoint, the source VM continues to run and the checkpoint is transferred to target host.

(3)Iteratively transfer log files: In the first round, the checkpoint is copied from source to target host, while the VM on source is running and events are recorded in a log file. Target host is replaying with received log files once it had recovered from the checkpoint. Other iterations transfer log file generated during the previous round. The iterative process is convergent if the log is replayed faster than it generated.

(4)CPU Scheduling: This procedure is not necessary. One of the two conditions or both, the size of log generated on source host or accumulated on target host larger than S_c that is the threshold size of log files, will trigger CPU scheduling. The scheduling policy is source host reduces the percentage processor quantum allocated to source VM by C(We define it in next subsection). If the processor quantum of source VM has been at a low level, the source host will not adjust processor quantum after it receive a slow-down request from target host. If and only if the size of log file on source host is smaller than S_c and the source host received a reset message from target host, the processor quantum of source VM will be resumed.

(5)Stop-and-Copy: This procedure will be executed when the size of log files unconsumed on target host is lower than a specified size(we define this low threshold value as S_f in our analysis) or the number of iteration reaches N, the maximum number of iteration we defined . The source VM is suspended, then the last log file will be transferred and replayed. After that, the source and target hosts have the consistent states.

(6)Service Takeover: After target host informs source host that it has synchronized their states, source VM can be discarded. The migrated VM running on target host is the primary host now and takes over the services offered by source VM.

This approach has a tolerant process. The source VM would be discarded until it received firm message from target VM. If some failure occurs during migration, the VM on source host can be resumed.

B. Algorithm analysis

In this section, we will give the mathematical analysis of our algorithm. Some important notations and their corresponding definitions listed as follows:

V_c: the size of checkpoint which will be transferred at first round.

R_t: log transfer rate, which lies on the network bandwidth between source and garget hosts. It is an average value.

R_r: log replay rate on target host, which denotes the average rate of replay at the whole live migration process.

R_i: the average log growth rate at round i, which denotes the workload and processor quantum adjustment of source VM.

S_f: the threshold value of log files size at which the iterative transfer process could terminate.

S_c: the threshold value of log files size at which the processor quantum allocated to source VM should be adjusted. Of course, $S_c > S_f$.

C: the ratio of CPU quantum after adjustment to original.

We define the log files list transferred at each round as $L = \langle log_1, log_2, \cdots, log_n \rangle$, and their sizes as $S = \langle S_1, S_2, \cdots, S_n \rangle$. The elapsed time sequence at each round is defined as $T = \langle t_0, t_1, t_2, \cdots, t_n \rangle$ and t_0 presents the elapsed time to transfer the checkpoint file.We set the S_f as default value 1KB to make our model simple. In the following analysis and experiment, we see the transfer rate R_t and replay rate R_r in different rounds as constant values.

The elapsed time in each round can be calculated as follows:$t_0 = \frac{V_c}{R_t}$, $t_1 = \frac{R_0 t_0}{R_t}, \cdots, t_n = \frac{R_{n-1} t_{n-1}}{R_t} = \frac{V_c \prod_{i=0}^{n-1} R_i}{R_t^{n+1}}$, where t_0 presents the time to transfer checkpoint, and t_n is the time cost to transfer the log file generated during previous round. Then the total migration time(T_{total}) can be calculated as:

$$T_{total} = \sum_{k=0}^{n} t_i = \frac{V_c}{R_t} \left(1 + \sum_{k=1}^{n} \frac{\prod_{i=0}^{n-1} R_i}{R_t^k} \right) \tag{1}$$

With (1), the total data transferred($S_{transfer}$) during migration becomes:

$$\begin{aligned} S_{transfer} &= V_c + \sum_{i=1}^{n} S_i = T_{total} R_t \\ &= V_c \left(1 + \sum_{k=1}^{n} \frac{\prod_{i=0}^{n-1} R_i}{R_t^k} \right) \end{aligned} \tag{2}$$

Now, we can analyze the downtime of the whole live migration. It is composed of three parts: t_n, the time of last log file transferred in stop-and-copy phase; the time of replaying last log file on target host; other time spend on start-up and services takeover. All the three parts can be done within a very short time. The total downtime can be represented as: $T_{downtime} = t_n + \frac{S_n}{R_r} + t_{other}$. The CPU scheduling can reduce the rate of log generation. The size of log generated on source host or accumulated on target host larger than S_c will trigger CPU scheduling. We give the mathematical expression of the trigger condition. $R'_i * t'_i > S_c$, represents the log generated on source host exceed the threshold value S_c. R'_i is the average rate of log generation in round i before the

trigger and it is higher than R_i because of low CPU quantum. t'_i represents a period of time from the beginning of round n to the time of CPU scheduling occurs. $(R_t - R_r)(\sum t_i + t'_i) > S_c$,represents the size of unconsumed log on target host is larger than threshold value. At last, we give the iterative round of migration if the process is convergent. The log generated on source host and unconsumed log on target host must smaller than another threshold value S_f at the same round i, so the iteration round is:

$$n = min(\{i | R_{i-1} t_{i-1} \leq S_f \cap (R_t - R_r) \sum_{k=1}^{i-1} t_i \leq S_f\}, N)$$

If the process is not convergent, the iteration will finish at round N, defined maximum number of iteration. From all the equations above, we can make conclusion: a small checkpoint and fast transfer rate can improve convergence rate and reduce the total migration time, the log growth rate also effect the iteration.

IV. PERFORMANCE EVALUATION

In this section we describe the experiment to evaluate the performance of our migration approach. It presents measurements of migration downtime and total migration time when a VM migrated in a LAN. With different workloads, experiment shows that our approach can make migration of VM fast and transparent.

A. Experimental Setup

Our experimental environment is built on two PCs with the same hardware configurations. Each host has 2 GB DDR RAM and a Intel Atom CPU D410. Two hosts are connected by a 1000 Mb/s Ethernet network. We used CentOS 5.5 with kernel 2.6.18 ported to UMLinux as guest OS and host OS. We chose Xen 3.0.1 as our virtual machine monitor. All the VMs were configured to 512 MB of RAM.

In each experiment a single VM was migrated five times between two physical hosts. The results we listed were the average of five trials. The experiments use different workloads:

(1)Idle:an idle Linux OS for daily use;

(2)Static web application: Apache 2.0.63 was used to measure static content web server performance. Each of 3 clients was configured with 100 simultaneous connections and repetitively downloading a 128 KB file from the server;

(3)Dynamic web application: We chose SPECweb99, a complex benchmark for evaluating web server, to evaluate our approach. Require dynamic content generation, HTTP POST operations and execution of a CGI script compose the workload of SPECweb99;

(4)Unixbench: It is a benchmark suit for Linux. It provides a basic indicator of the performance of a Unix-like system.

To compare our approach with previous migration algorithm, we use pre-copy algorithm implemented in XenMotion and make the same experiments for the above workloads.

B. Algorithm performance

Many other migration algorithms are focus on downtime and total migration time to evaluate the performance. We mostly concern about the downtime. Because the services running on VM are unavailable during downtime and it plays an import role in good user experience.

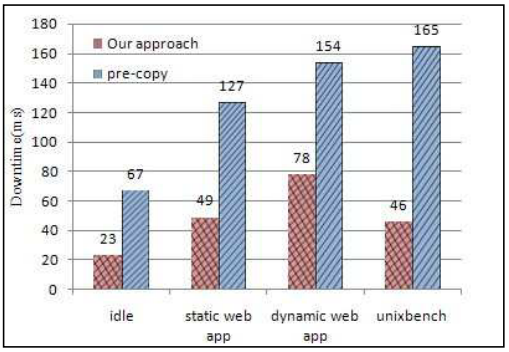

Fig. 1. The downtime for different workloads

Fig.1 shows the downtime of migration for different workloads. To compare our approach with pre-copy algorithm, we did the same migration experiment with pre-copy algorithm. The result shows that our approach has much less downtime than pre-copy. It reduced the downtime by 65.67%, 61.41%, 49.35% and 72.12% respectively for the above workloads, an average of 62.12%.

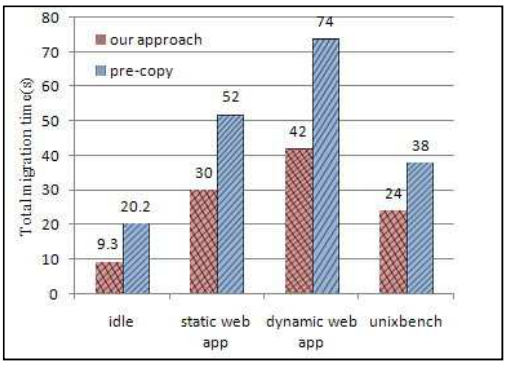

Fig. 2. The total migration time for different workloads

Fig.2 shows the total migration time of our approach and pre-copy algorithm for different workloads. For the workload of dynamic web application, the pre-copy algorithm spent 74 seconds, more than a minute, in completing the whole process of migration. Our approach reduced the total migration time by 53.96%, 42.31%, 42.24% and 36.84%, compared with pre-copy algorithm, an average of 43.84%. The total migration time for dynamic web applications was longer than other workloads in our approach. The reason is that the log growth rate was high and the process has twice CPU scheduling, which we will describe in the following subsection. In the term of downtime and total migration time, our approach made some improvement.

C. CPU Scheduling

Our approach has a CPU scheduling mechanism to adjust log growth rate. In our experiment for the different workloads, average log growth rate were 9.6KB/s, 32.5KB/s, 753.1KB/s and 321.4KB/s respectively. Dynamic web application and unixbench had much higher log rate than others. CPU scheduling is a not necessary procedure which will be triggered by size of log generated on source host or accumulated on target host larger than S_c(we defined 1MB). Migration for idle and static web application did not have the CPU scheduling.

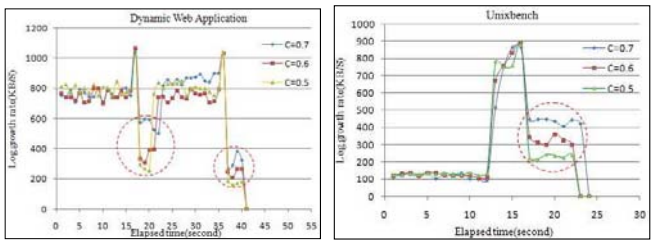

Fig. 3. Log growth rate

Fig.3 shows the log rate of dynamic web application and unixbench. To test impact of different percentages of CPU quantum, we set different C values, 0.7, 0.6 and 0.5. 0.7 was default value. In the figures the areas indicated by circles show the log rate after CPU scheduling. Migration for dynamic web application had twice CPU scheduling. As we described in III-A, if and only if the size of log file on source host is smaller than S_c and the source host received a reset message from target host, the processor quantum of source VM will be resumed.The smaller the C value was, the lower log rate would be. But there was not liner relationship between percentage of CPU quantum and log rate.

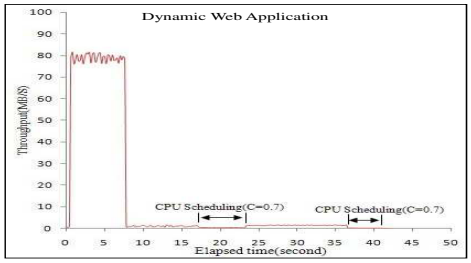

Fig. 4. Network throughput of dynamic web application

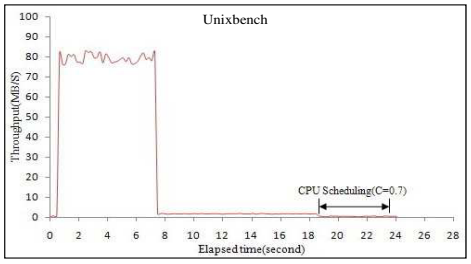

Fig. 5. Network throughput of Unixbench

Fig.4 and 5 show the network throughputs of migration for dynamic web application and unixbench. The high throughput phases were the transfer of checkpoint at beginning. The parts indicated in the figures were the phases of CPU scheduling. Compared with log growth rate,we can get this conclusion: reducing processor quantum allocated to VM low response speed of system, which can decrease the log growth rate and network throughput.

V. Conclusion

In this paper, we design and evaluate a novel approach for live VM migration. It shows how we combine technology of recovering system, checkpointing/recovery and trace/replay, with CPU scheduling to provide fast and transparent migration. Our approach has short downtime and reasonable total migration time. Experimental measurements show our algorithm gets better average performance compared with pre-copy scheme: up to 62.12% on downtime and 43.84% on total migration time.

However, we are not sure our approach can work well in more complex environment, such as multi-processor environment or cluster consisted of hosts with different hardware. In the future, we plan to mend this approach to make it fit more complex environment and get a better performance.

References

[1] 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 2nd Symposium on Networked Systems Design and Implementation (NSDI'05), May 2-4, 2005, Boston, MA, USA, pp.273–286.

[2] G. W. Dunlap, S. T. King, S. Cinar, M. Basrai, and P. M. Chen. *ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay*. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02), ACM Press, December 8-11, 2002, Boston, MA, USA, pp.211–224.

[3] M. Kozuch, and M. Satyanarayanan. *Internet suspend/resume*. In Proceedings of the IEEE workshop on mobile computing systems and applications, 2002. pp. 40C46.

[4] M. Kozuch ,M. Satyanarayanan , T. Bressoud , C. Helfrich,S. Sinnamohideen. *Seamless mobile computing on fixed infrastructure* . IEEE Computer ,2004 ,32 (7) :65 - 72.

[5] 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 5th Symposium on Operating Systems Design and Implementation (OSDI'02), December 8-11, 2002, Boston, MA, USA

[6] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M. S. Lam. *The collective: a Cache-based system management architecture*. In Proceedings of the Second USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2005). Boston ,MA ,2005. 259 - 272.

[7] M. Nelson, B. H. Lim, and H. Hutchins. *Fast transparent migration for virtual machines*. In Proceedings of USENIX'05, 2005.

[8] C. A. Waldspurger. *Memory resource management in VMware ESX server*. In Proceedings of the 5th symposium on operating systems design and implementation (OSDI), ACM operating systems review, 2002, Special issue. p. 181-194.

[9] Jin H, Gao W, Song W, Xuanhua S, Xiaoxin W, and Fan Z. *Optimizing the live migration of virtual machine by CPU scheduling*. Journal of Network and Computer Applications,2010.

[10] R. Bradford, E. Kotsovinos, A. Feldmann, H. Schioberg. *Live wide-area migration of virtual machines including local persistent state*. In Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE'07). New York ,NY,USA :ACM ,2007. 169 - 179.

[11] Haikun L, Hai J, Xiaofei L, Liting H, Chen Y. *Live Migration of Virtual Machine Based on Full System Trace and Replay*. In Proceedings of the 18th ACM international symposium on High performance distributed computing,2009