ITC-LM: A Smart Iteration-Termination Criterion Based Live Virtual Machine Migration

Liangwei Zhu, Jianhai Chen, Qinming He, Dawei Huang, Shuang Wu

ITC-LM: a Smart Iteration-Termination Criterion Based Live Virtual Machine Migration

Liangwei Zhu, Jianhai Chen, Qinming He, Dawei Huang ,Shuang Wu
College of Computer Science, Zhejiang Universiy,
Zheda Rd. 38, Hangzhou 310027, China
{zhulw,chenjh919,hqm,tossboyhdw,catting}.zju.edu.cn

Abstract. Live migration of virtual machines (VMs) plays an important role in grids, clouds and datacenters, and has become the cornerstone of resource management in virtualized systems. The efficiency of live migration depends on the downtime, total migration time and total transferred data. However, while migrating a memory-intensive VM, XEN/KVM often do many useless iterations of memory copy in order to reach expected downtime which can never be reached, leading to a great deal of useless data transferring and insufferable total migration time. It consumes mass of network bandwidth and CPU resource when transferring memory from one to another node. Hence, a critical task is to determine the optimal time to terminate the copy iteration for live migration. In this paper, we propose a smart iteration-termination criterion based live migration which is termed as ITC-LM, to self adaptively control when to terminate iteration. We have implemented ITC-LM into KVM/QEMU. The improvement is significant, especially when migrate a memory-intensive VM. The experimental results show that, our approach can decrease 50.33% of total transferred data on average without impairing migration downtime.

Keywords: Virtualization, Live migration, Iteration-Termination Criterion, Terminating Conditions

1 Introduction

Virtualization technology plays an important role in cloud computing [1, 2], in which a large number of virtual machines (VMs) are dynamically allocated to multiple physical machines (PMs). Virtualization enables live migration of VMs, which provides a flexible way to relocate VMs from one physical node to another, leading to efficient resource management [3], such as load balancing and power-saving etc.

Live migration of virtual machine is the essential mechanism of virtualization, which is included in all current mainstream virtualization platforms such as KVM [4], XEN [5], VMware [6], etc. Pre-copy as default live migration method of these platforms is the most popular algorithm of live migration, which first sends VM's memory to the destination host and then resumes VM in it. In order to improve migration performance, some research employs memory compression to reduce the transferred pages during live migration.

However, the efficiency of current live migration methods are not always satisfying, especially in a memory-intensive scenario. Now methods keeps the iterations of copying memory until the downtime is short enough, and if the expected downtime is always too high, it stops memory copy iterations after a fixed number of memory-copy iterations. As a result, these methods usually stop memory--copy iteration too late in memory-intensive scenarios and will consume a lot of network resource of a data center, even resulting in the performance degradation of data center.

Unfortunately, the existing live migration approaches commonly ignore the problem of determination about when to terminate the iterations. In pre-copy, memory is transferred from the source node to the destination node while VM is still running on source. Modified pages which are generated in the iteration are transferred in subsequent iteration. This process based on that the VM remaining dirty pages can converge to a small value. However, there are still lots of remaining dirty pages after multiple iterations when low-bandwidth or VM with a high workload. In other words, more iterations of copy memory cannot decrease the remaining dirty pages and no more benefit to the service downtime.

In this paper, we propose a smart iteration-termination criterion (ITC) based live migration method, termed as ITC-LM. We use ITC value to determine when to terminate the iteration in live migration. Actually, ITC value changes dynamically during the live migration process. In our method, we terminate the iteration when the ITC value is less than a given threshold.

The main contributions of this paper are as follows:

First, we proposed a smart ITC-LM technique to decide when to stop iteration during live migration. To the best of our knowledge, our method is the first to consider both the iteration rounds and the convergence of remaining dirty pages.

Second, we have implemented our proposed algorithm in recent stable release of KVM/QEMU [9, 10] and show that our methods can be conveniently deployed in virtualization platform.

Eventually, we demonstrate the effectiveness of ITC-LM method in our experiments by four different kinds of workloads. The results show that ITC-LM decreases 53.35% of total migration time and 50.33% of total transferred data on average.

2 Background and Motivation

There are various algorithms of live migration. Most of these studies are mainly based on the pre-copy live migration algorithm. It basically works [21] as follows:

1) The resources of memory and VCPUs are reserved on the destination.
2) Memory of VM is sent to destination and using bitmap to log the dirty pages which rewrite during the memory copy.
3) The source continuously copies VM's memory dirty pages to the destination. A number of iterations are performed to retransfer the pages which are dirtied in previous iteration.
4) Suspend the running VM at the source, and copy remaining pages to the destination.

5) The VM is resumed on the destination.

In the best case, the approach of pre-copy can achieve an expected downtime by several iterative copy operations.

2.1 Terminating Conditions

There are some common conditions to decide the time to terminate the iterative copy operations.

Remaining Dirty Pages. Ideally, the default size of remaining dirty pages can be reached using iterative copy operations. However this is completely depending on the assumption of remaining dirty pages can converge expected size. Some application rewrite memory frequency that remaining dirty pages still very large over multiple rounds [17].

Maximum Number of Iterations. The terminative conditions of exact values for the maximum number of iteration are arguable, sometimes when the dirty rate is low compared to the transfer rate, the remaining pages will decrease quickly and down time will not have benefit from more iteration.

Hybrid Terminating Condition. Some of approaches use terminative conditions simultaneously of above two as shown in figure 1. Each iteration check the remaining pages whether below the expected remain pages, and check the number of iteration is more than the maximum number of iterations. This condition has been used in XEN, KVM/QEMU etc.

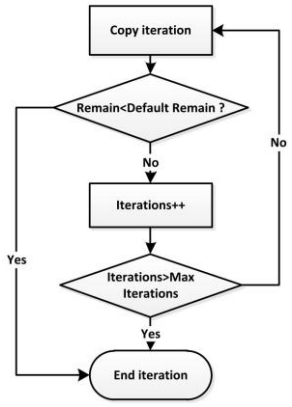
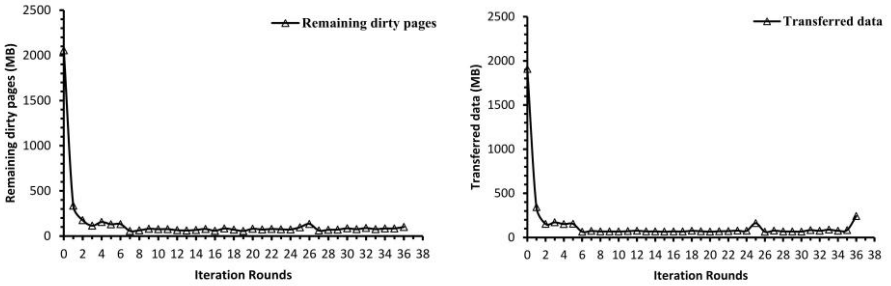


Fig. 1. A typical algorithm of use two terminative conditions simultaneously

2.2 The Problems of Common Terminating Conditions

The hybrid terminating condition can finish the live migration but the default remaining pages size and maximum number of iterations is hard to estimate. Beside, we can know that in order to minimize the size of remaining dirty pages the maximum

iterations need to be set to a value that larger enough for most cases of live migration. However, a lot of useless iterations of pre-copy caused large amount of transferred data, especially when migrate a memory-intensive VM.



(a)Remaining dirty pages (b)Transferred data

Fig. 2. Number of remaining dirty pages and transferred data of each migration iteration rounds when migrate a VM which running workload of RUBiS webserver workload. VM size is 2GB.

From figure 2 we observe that the remaining pages converge from the fifth iteration. But the remaining dirty pages size still bigger than the default size 30MB (some approaches use remaining pages size divide transfer rate). So iterations of copy will continuous perform until the number of iterations exceeds the maximum number which is set as thirty-seven that larger enough for most cases of live migration in our experiments.

3 Algorithm Design

Although, the live migration complete after number of iteration exceeds the maximum number, the more iteration the more network bandwidth and CPU resource it consumes and it would prolong the total migration time. Moreover, it makes applications in migrated VM suffer longer time of performance degradation.

The iterations of pre-copy are completely depending on the assumption of the remaining pages can converge to a small value. However iteration of pre-copy may never converge to a small value or even the convergence of iteration never happen when the VM's workload is high or bandwidth instability. So there should be some smart threshold to force the final iteration of a live migration which does not converge.

3.1 Terminating Condition Based on Remaining Dirty Pages

First, some notations are defined as follows:

M_{r_i} : The remaining dirty pages after i*th* iteration in the migrated virtual machine.

M_{t_i} : The data is transferred to destination of i*th* iteration.

T_i : The time of complete i*th* iteration used.

R_{page_i} : The average dirty pages rate of i*th* iteration.

R_{tran_i} : The average transferred rate of i*th* iteration.

The average dirty page rate of i*th* iteration can be calculated as

$$R_{page_i} = \frac{M_{r_i} - (M_{r_{i-1}} - M_{t_i})}{T_i} \quad . \tag{1}$$

The average transferred rate of i*th* iteration can be presented as:

$$R_{tran_i} = \frac{M_{t_i}}{T_i} \quad . \tag{2}$$

Only when R_{tran_i} is greater than R_{page_i}, the iteration can reduce the remaining pages. From formula (1) and (2) we can get:

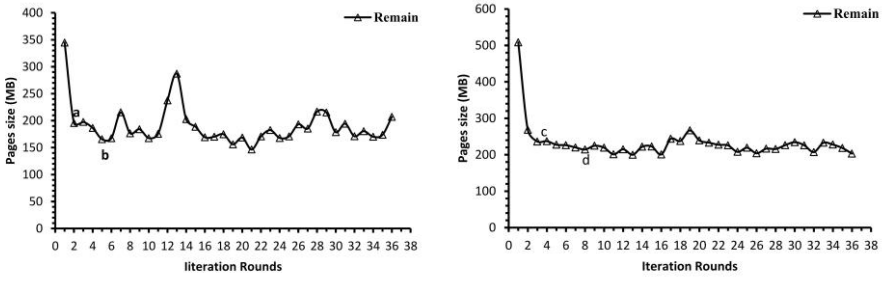
$$M_{r_i} < M_{r_{i-1}} \quad . \tag{3}$$

Different between two consecutive remaining pages is defined as:

$$D_i = M_i - M_{i-1} \quad . \tag{4}$$

The above inequality indicates the remaining pages should be less than its previous round. Thus the iteration can reduce the remaining page size. Otherwise the rounds have no contribution to reach to the default value.

From formula (4), it is clear that when D_i is not less zero is a good terminative condition. However, VM's dirty page rate and transfer rate are not stable in real virtualized system, thus once we find it unreliable that the remaining dirty pages is less than its previous one.



(a) RUBiS 1GB (b) RUBiS 2GB

Fig. 3. Remain dirty pages in successive iterations. The experiments choose benchmark of RUBiS as workload for each VM memory size. In order to see clear the trend of remaining dirty pages, the first iterations are taken off from the figures. The default remaining dirty pages set as 30MB and maximum iterations is thirty-seven (a number that larger enough for most cases of live migration).

Figure 3 illustrates the remaining dirty pages fluctuate during iterations and pages will continue to reduce after a shock. In detail, if only use the condition of difference between two successive remaining dirty pages will terminate the iteration at the first shock points (see the point of a and c in Figure 3). However the remaining dirty pages will continuously decrease after these points (see the point of b and d in Figure 3).

3.2 Iteration-termination Criterion

Based on the analysis of the iterative characteristics in the VM live migration, the key idea of the proposed method is to avoid directly terminating iterations when the remaining dirty pages do not decrease. So we give a smart iteration-termination criterion (ITC) based live migration method, termed as ITC-LM. In our method, we use ITC value to accumulative the variation tendency of remaining dirty pages and determine when to terminate the iteration in live migration. The value of ITC is relation to the difference of remaining pages between two consecutive iterations.

According to the variation of the remaining pages of iterations, if the remaining pages of current iteration are less than the previous one, then we add a trust value to ITC. The bigger ITC value means the more valuable to do iterations. On the contrary, if the remaining pages of current iteration are not less than the previous one, it means doing iterations has no benefit for reducing downtime, and then we cut down ITC value by dividing ITC value to a distrust coefficient which is used to constraint the rate of ITC descent.

We conclude ITC by formula (5) as follows.

$$ITC = \alpha(ITC + c\_trust) + (1 - \alpha)ITC / c\_distrust, \qquad \textbf{(5)}$$

Where α is a 0-1 constant, $\alpha = \begin{cases} 1, D_i < 0 \\ 0, D_i >= 0 \end{cases}$, c_trust is a constant value which denotes the trust value, and $c\_distrust$ is a constant value which denotes the distrust coefficient.

According to our experiments, the value of c_trust and $c\_distrust$ is empirical and correlative. For example, when we set c_trust to 1 and set $c\_distrust$ to 2, then we obtain a good performance of live migration in our experiments and we will do more research about values of c_trust and $c\_distrust$ in our future work.

3.3 ITC-LM Algorithm

We have designed the ITC-LM Algorithm and implemented it in recent stable release of KVM/QEMU using ITC. QEMU is an open source machine emulator. KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware which containing virtualization extensions. When QEMU use KVM for its virtualization acceleration, can get a better performance. The major part of ITC-LM was implemented in QEMU. We use bitmap of memory to calculate the size of dirty pages. The pseudo code of ITC-LM algorithm is listed in follows:

Pseudo Code of ITC-LM algorithm.

```
ITC=0;c_trust=1;c_distrust=2;
pre_remain_pages_size=full_memory_size();
while(true){
  copyiteration()
  if(remain_pages_size()<pre_remain_pages_size){
        ITC=ITC+c_trust;
        pre_remain_pages_size=remain_pages_size();
     }
  else{
        ITC=ITC/c_distrust;
        if(ITC<=1)
             break;
        else
             pre_remain_pages_size=remain_pages_size();
     }
}
end_iteration();
```

4 Evaluation

In this section, we perform a series of live migration experiments with some various characteristic workloads on VM with varying working set sizes to evaluate the performance of ITC-LM algorithm by comparing with QEMU/KVM default pre-copy algorithm (Pre-default) which implemented hybrid terminating condition. In the following, we first introduce the experimental environment, and then we present the results of different benchmarks.

4.1 Experimental Environment

All live migration experiments are performed on two identical hosts as source and destination host respectively. Each host has dual Intel(R) Xeon(R) CPU E5606 @ 2.13GHZ with a total of eight cores. Each one has 16GB RAM. The host runs Ubuntu

12.04LTS with KVM module and QEMU-1.4.0. The source and destination hosts are connected via Gigabit switched Ethernet. The OS of Linux VM are all Ubuntu12.04LTS with Kernel-3.2.0 and all VM images are stored in a Network File System (NFS). We perform experiments VMs are configured with two virtual CPUs. In each experiment the nodes of source and target only run the live migration VM. Besides, we use another identical host to deployment the client emulator of RUBiS.

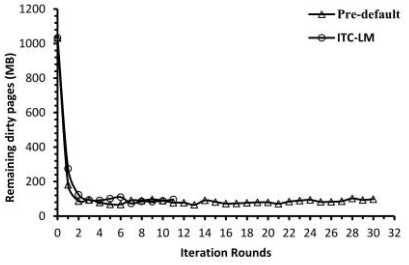
4.2 Overview of Workload

We perform our experiments with the following VM workloads:

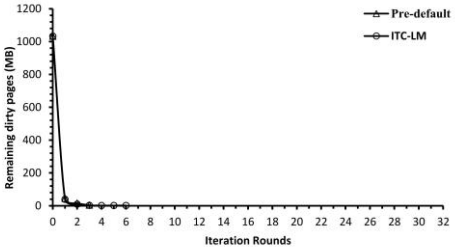
1. Kernel-complication. Linux kernel compilation is a balanced workload to use the source of VM. Two parallel threads were used to run Linux 3.8.5 kernel [12] compilation.
2. Parallel Benchmarks. The NAS parallel benchmarks (NPB) [14] are a set of programs which evaluate the performance parallel performance, available in commonly-used programing models like MPI and OpenMP. In our experiments, we use Embarrassingly Parallel (EP) of NPB (NPB-EP) to simulation the parallel computing workload.
3. SPEC jbb2005 [13]. It is a SPEC'S benchmark for evaluating the performance of server side java. It provides enhanced workload with a more object-oriented manner to reflect real-world applications.
4. Dynamic web Server. Rice University Bidding System (RUBiS) [11] which is a prototype modeled after eBay.com used to evaluate patterns and application servers performance. It contains a client-browser emulator, and we implement it in a third physical host.

4.3 Experiment Results

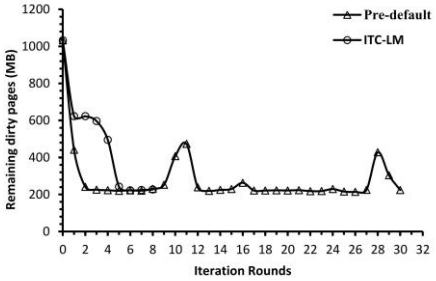
In this section, the results from the four workloads are present. The evaluation metric of experiments primarily includes total migration time, total transferred data and downtime during live migration of virtual machine. We run live migration five times for each workload and use the arithmetic mean for each metric.



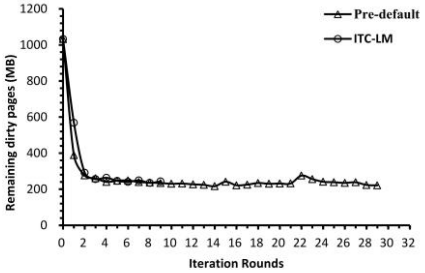
(a)Kernel-Compiling



(b) NPB-EP



(c) SPECjbb



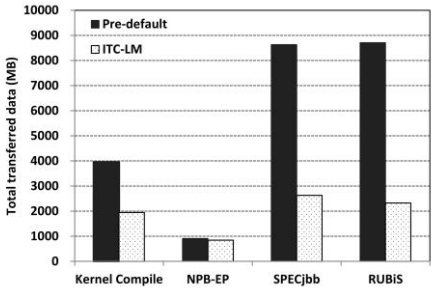
(d) RUBiS

Fig. 4. Remaining dirty pages of each iteration compared with ITC-LM and Pre-default for each workload. (Memory 1GB)

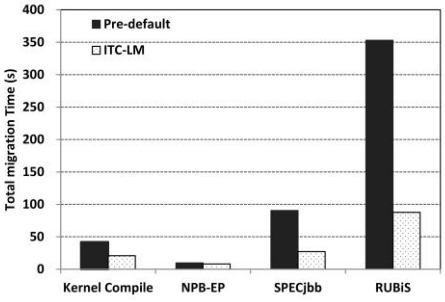
Total Transferred Data. Figure 4 respectively shows the each iteration remaining data of during live migration of VM running. The iteration rounds of ITC-LM far less than the pre-copy default algorithm in KVM/QEMU under the workload of kernel-compile, SPECjbb and RUBiS. As figure 4 (b) shows the benchmark of NPB-EP is a compute-intensive workload and VM produce less dirty pages, and iteration ending soon. Thus two approaches both in less iteration.

Experimental results in figure 5 (a) show that compare with KMV/QEMU's default migration algorithm, ITC-LM can reduce total transferred data 50.73%, 7.74%,69.55%,73.29% respectively in diverse workload of above, an average of 50.33%. This will lighten greatly network loads of data center.

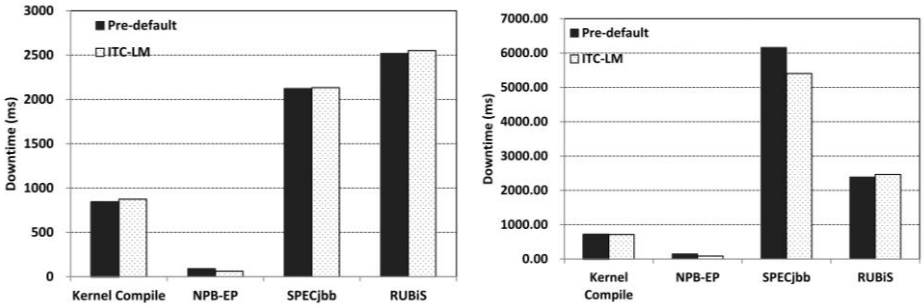
Total Migration Time. The results in figure 5(b) show that benefit from less iteration, ITC-LM can reduce total migration time 50.54%, 17.70%, 70.03%,75.14%, an average of 53.35%. For this, the service running in live migration VM can suffer less time of the decrease of quality.



(a)Total transferred data (Memory 1GB)



(b) Total migration time (Memory 1GB)



(c) Downtime (Memory 1GB) (d) Downtime (Memory 2GB)

Fig. 5. Total transferred data, total migration time and downtime of ITC-LM and Pre-default during live migration for four different kinds of workloads.

Downtime. In order to evaluate the influence of ITC-LM on downtime, experiments are performed with two memory size: 1GB and 2GB. Figure 5 (c) and (d) shows ITC-LM can get a good performance in total transferred data and total migration time while has slight influence on the downtime of live migration. This is because ITC-LM based on a smart threshold not a default remaining dirty pages or maximum iteration rounds.

6 Related Work

The technology of live migration is widely used in virtualization. At present, there are several types of live migration methods. The pre-copy approach is a main migration method in the mainstream virtualization platform such as KVM [4], XEN [5], VMware [6], etc.

Some research has been done to improve the performance of live migration based on pre-copy in which the widespread used method is memory compression. Zhang et al. [15] proposed a novel approach MMD to find identical and similar memory pages to redundant memory data. Delta compression technique [16] applied XOR on the current pages with kept previously sent pages in source host. Jin et al. [17] designed an adaptive memory compression based on memory page characteristics. ME2 [18] identified useful pages and then used RLE algorithm to compress data. Page rewriting frequency is related to dirty page rate. Microwiper [19] ordered dirty memory pages according their rewriting rate. Petter et al. [20] designed a page priority map on top of the dirty page bitmap and proposed dynamic page transfer reordering based on it. CR/TR-Motion [22] log execution trace on source and replay it on target host. Chiang et al. [27] proposed a bootstrapping VM introspection technique to get the information of memory pool, and skips free memory pages during migration. Jo et al. [28] first sent the memory-to-disk mapping to the host, and then fetch the contents directly from the shared storage.

Post-copy [7] resumes running VM on the target host with only its CPU state before copying the VM's memory from source host to target. Adaptive pre-paging and dynamic self-ballooning [8] can improve the post-copy performance. Hirofuchi et

al. [23] through a lightweight extension implement post-copy to KVM. Besides, some researches proposed hybrid live migration [24, 25, 26] approach that use pre-copy and post-copy methods simultaneously. They do some memory copy iteration of pre-copy before the stage of post-copy method.

7 Conclusions and Future Work

In this paper, we have presented the design and implementation of a smart ITC-LM technique for live migration of virtual machines. We choose four representative server applications in modern data center to verify our algorithm. The results show that ITC-LM has a good performance in different kinds of workload. In future work, we will study trust value and distrust coefficient of ITC-LM to make our approach more effective. Furthermore, we will implement our approach to other virtualization platforms.

8 Acknowledgments

This work was partly supported by the National Key Technologies R&D Program under Grants No. 2011BAD21B02

References

1. Armbrust M, Fox A, Griffith R, et al. A view of cloud computing[J]. Communications of the ACM, 2010, 53(4): 50-58.
2. Fox A, Griffith R, Joseph A, et al. Above the clouds: A Berkeley view of cloud computing[J]. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, 2009, 28.
3. Wang X, Du Z, Chen Y, et al. Virtualization-based autonomic resource management for multi-tier Web applications in shared data center[J]. Journal of Systems and Software, 2008, 81(9): 1591-1608.
4. Kivity A, Kamay Y, Laor D, et al. kvm: the Linux virtual machine monitor[C]//Proceedings of the Linux Symposium. 2007, 1: 225-230.
5. Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 164-177.
6. Nelson M, Lim B H, Hutchins G. Fast transparent migration for virtual machines[C]//Proceedings of the annual conference on USENIX Annual Technical Conference. 2005: 25-25.
7. Hines M R, Deshpande U, Gopalan K. Post-copy live migration of virtual machines[J]. ACM SIGOPS operating systems review, 2009, 43(3): 14-26.
8. Hines M R, Gopalan K. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning[C]//Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM, 2009: 51-60.
9. Kernel Based Virtual Machine, KVM, http://www.linux-kvm.org/page/Main_Page

10. QEMU, http://wiki.qemu.org/Main_Page
11. Rice University Bidding System, RUBiS, http://rubis.ow2.org/
12. Linux-kernel, https://www.kernel.org/
13. Standard Performance Evalution Corporation, SPECJbb2005, http://www.spec.org/jbb2005/
14. NAS Parallel Benchmarks, NPB, http://www.nas.nasa.gov/publications/npb.html
15. Zhang X, Huo Z, Ma J, et al. Exploiting data deduplication to accelerate live virtual machine migration[C]//Cluster Computing (CLUSTER), 2010 IEEE International Conference on. IEEE, 2010: 88-96.
16. Svärd P, Hudzia B, Tordsson J, et al. Evaluation of delta compression techniques for efficient live migration of large virtual machines[C]// Virtual Execution Environments (VEE). ACM, 2011, 46(7): 111-120.
17. Jin H, Deng L, Wu S, et al. Live virtual machine migration with adaptive, memory compression[C]//Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on. IEEE, 2009: 1-10.
18. Ma Y, Wang H, Dong J, et al. ME2: Efficient Live Migration of Virtual Machine with Memory Exploration and Encoding[C]//Cluster Computing (CLUSTER), 2012 IEEE International Conference on. IEEE, 2012: 610-613.
19. Du Y, Yu H, Shi G, et al. Microwiper: Efficient Memory Propagation in Live Migration of Virtual Machines[C]//Parallel Processing (ICPP), 2010 39th International Conference on. IEEE, 2010: 141-149.
20. Svard P, Tordsson J, Hudzia B, et al. High performance live migration through dynamic page transfer reordering and compression[C]//Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. IEEE, 2011: 542-548.
21. Clark C, Fraser K, Hand S, et al. Live migration of virtual machines[C]//Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2. USENIX Association, 2005: 273-286.
22. Liu H, Jin H, Liao X, et al. Live virtual machine migration via asynchronous replication and state synchronization [J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(12): 1986-1999.
23. Hirofuchi T, Nakada H, Itoh S, et al. Enabling instantaneous relocation of virtual machines with a lightweight vmm extension[C]//Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on. IEEE, 2010: 73-83.
24. Sahni S, Varma V. A Hybrid Approach to Live Migration of Virtual Machines[C]//Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on. IEEE, 2012: 1-5.
25. Shribman A, Hudzia B. Pre-Copy and post-copy VM live migration for memory intensive applications[C]//Euro-Par 2012: Parallel Processing Workshops. Springer Berlin Heidelberg, 2013: 539-547.
26. Chen Y, Huai J P, Hu C M. Live migration of virtual machines based on hybrid memory copy approach[J]. Chinese Journal of Computers, 2011, 34(12): 2278-2291.
27. Chiang J H, Li H L, Chiueh T. Introspection-based memory de-duplication and migration[C]//Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM, 2013: 51-62.
28. Jo C, Gustafsson E, Son J, et al. Efficient live migration of virtual machines using shared storage[C]//Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM, 2013: 41-50.