Vol.2013-OS-126 No.12
2013/7/31

高可用システムのための待機系仮想マシンの集約

李 ヨンジュン[1,a)]　山田 浩史[2,b)]　古藤 明音[1,c)]　河野 健二[1,d)]　大村 圭[3,e)]　湯口 徹[3,f)]
盛合 敏[3,g)]

概要：仮想化環境において，ハードウェア障害はサービスの可用性を著しく低下させる要素のひとつである．ハードウェアが故障すると，その上で動作している仮想マシン (VM) が全て停止してしまうためである．そこで，VM の複製を別の物理ホスト (待機系ホスト) に作成し，ハードウェア障害が生じたらその複製 VM に制御を切り替える方式が提案されている．しかしながら，保護したい物理ホストが複数ある場合，同数の待機系ホストを用意しなければならない．そのため，待機系ホストの管理コストや電力コストがかさばりがちになる．本研究では，複製 VM の集約を実現する Backup VM Consolidation を提案する．複製 VM を集約することで，待機系ホスト台数の減少を可能にする．待機系 VM を集約するために，VM のメモリ情報に着目し，buffer cache といった起動に必ずしも必要のないメモリを転送せず，複製 VM のメモリサイズを小さく保つ．提案手法を Linux 3.6.0 および Qemu 0.15.1 上に実験を行った．実験結果より，提案手法が複製 VM のメモリサイズを小さくできることがわかった．

キーワード：高可用性, 仮想マシン技術，Cloud Computing,

Abstract: Users have come to expect 24×7 availability even for simple non-critical applications. Handling hardware failures is crucial to achieve high available services in virtualized environments because all the virtual machines (VMs) fail on the failing machine. Virtualization-based high availability is attractive for its cheap nature but it does not come for free; one backup machine is necessary to protect one primary machine. To survive the failure of any single machine in the data center, the number of physical machines must be doubled. This paper proposes backup VM consolidation that advances the cheap nature of virtualization-based high availability. The backup VM consolidation enables one backup machine to protect two or more machines, and thus can reduce the number of backup machines and achieves the availability in proportion to the number of available backup machines. The key insight behind the backup VM consolidation is that the whole memory image is not necessarily synchronized between a primary and backup machines. We implemented a prototype on Linux 3.6.0 and QEMU 0.15.1. The experimental results show that our prototype successfully consolidates backup VMs and reduces required physical memory of backup machines.

Keywords: High Availability, Virtual Machines, Cloud Computing

1. Introduction

Handling hardware failures is crucial to achieve high available services. Users have come to expect 24×7 availability even for simple non-critical applications, and businesses can suffer costly and embarrassing disruptions when hardware fails. Hardware failures become more severe in virtualized environments in data centers and cloud platforms. Since multiple VMs are consolidated on a small number of physical machines, a hardware failure affects all the services running on the failed machine.

Virtualization-based high availability [1], [2] is attractive for cheap high availability. In virtualization-based high availability, no special-purpose hardware or reengineering of existing software stacks is needed. This feature makes high availability handy for the services that prefer higher availability but whose affordable cost is limited. In *non*-mission-critical commercial systems, higher availability is preferred but the affordable hardware cost

1 慶應義塾大学
2 東京農工大学
3 日本電信電話株式会社ソフトウェアイノベーションセンタ
a) leeyj@sslab.ics.keio.ac.jp
b) hiroshiy@cc.tuat.ac.jp
c) koto@sslab.ics.keio.ac.jp
d) kono@sslab.ics.keio.ac.jp
e) ohmura.kei@lab.ntt.co.jp
f) yuguchi.toru@lab.ntt.co.jp
g) moriai.satoshi@lab.ntt.co.jp

Vol.2013-OS-126 No.12
2013/7/31

is limited due to severe price competition in the market. In virtualization-based high available systems, a primary machine can be protected with an off-the-shelf machine.

This paper describes *backup VM consolidation*, which advances the cheap nature of virtualization-based high availability in data centers or cloud environments. To protect all the machines in a data center, the number of machines must be doubled with existing techniques of virtualization-based high availability. A backup machine retains the complete copy of all the VMs on a primary machine. This implies that a backup machine has memory equal to or greater than the primary machine. If the backup machine does not have enough memory to retain the memory images of the VMs, it must swap in/out memory pages from/to disks. The overhead caused by disk accesses cannot be accepted in virtualization-based high available systems. The backup machine receives updated memory pages at very high frequency — as often as every 25ms in Remus — to keep consistency between primary and backup machines.

In the backup VM consolidation, a single backup machine can protect more than one primary machine. A key insight behind the backup VM consolidation is that the whole memory image is not necessarily retained by a backup machine. If we can reduce the amount of memory that must be synchronized between primary and backup machines, a backup machine can retain more VMs than a primary machine. In backup VM consolidation, memory pages are that can be reconstructed after a failure occurs are not synchronized between between primary and backup machines.

Backup VM consolidation enables the trade-off between the availability and the cost of backup machines. In the existing approach, if we have one backup machine for two primary machines, we have to choose one primary that should be protected. In the backup VM consolidation, both of the primaries can be protected by a single backup machine. If one of the primaries fails, the backup machine takes over the failed primary. If the other primary also fails, it cannot be protected because there is only one backup machine.

A prototype has been implemented in Linux 3.6.0 and QEMU 0.15.1, and some experiments are conducted. The experimental results show that our prototype successfully reduces the size of memory images of backup VMs.

2. Backup VM Consolidation

In backup VM consolidation, we assume that all the machines (at least a primary machine and a backup machine) share a single storage. In large-scale data centers, a storage system is usually shared by a number of physical machines because a share storage usually reduces the maintenance costs and is superior in reliability. In this configuration of data centers, a backup machine can access to the storage that a primary machine uses to store its persistent data. This assumption plays an important role in our design of backup VM consolidation.

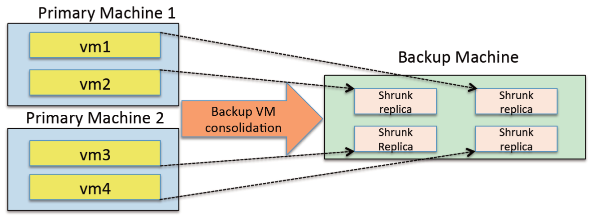


図 1 Basic concept of backup VM consolidation

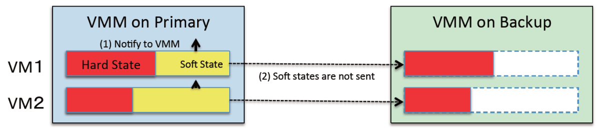


図 2 Basic approach. In backup VM consolidation, no soft states are transferred to backup machines.

Figure 1 illustrates the basic concept of backup VM consolidation. As shown in this figure, a memory image that must be retained in backup machines are shrunk in backup VM consolidation. By shrinking or distilling a memory image of a primary VM, a single backup machine can retain virtual machines of more than one physical machine.

In backup VM consolidation, memory pages are divided into two classes: 1) *soft* pages and 2) *hard* pages. A soft page is a memory page that can be reconstructed from the data saved in shared storage in a data center. File cache is a typical example of soft pages. Since the file contents can be fetched from the shared storage after the backup takes over the primary, backup VMs need not retain the memory pages (i.e, page cache) used for file cache. A hard page is a memory page that cannot be reconstructed from the data in shared storage. Memory pages used for heap and stacks are typical examples of hard states. Backup VM consolidation synchronizes only the hard states of each protected VM between primary and backup machines. This saves the memory space in backup machines and enables the consolidation of more VMs than the primary machine.

Figure 2 illustrates the basic behavior of backup VM consolidation. When a memory image must be synchronized between a primary and backup machines, the guest operating system (OS) notifies the VMM of the memory usage information that describes which memory page is soft. Using this information, the VMM running on the primary machine removes all the soft pages from the pages that must be transferred to the backup machine to keep the consistency of memory images between the primary and backup machines. By doing this, backup VM consolidation reduces the memory size of backup VMs.

In our approach, soft states such as page caches are not transferred to backup machines, there is the possibility that the performance of the resumed VM is terribly low because all the page caches are gone and must be fetched from the shared storage. This performance penalty is peculiar to our backup VM consolidation. In the conventional virtualization-based high availability systems, all the page caches are kept consistent between a primary and backup machines. Therefore, once the backup machine resumes its operation, the expected performance is the same as the primary machine if the machine of the same specification is used.

We believe that this restriction of the backup VM consolidation can be mitigated if we regard extremely hot page caches as hard states (not soft states).

3. Design and Implementation

We designed a mechanism to embrace backup VM consolidation, and implemented a prototype on Linux 3.6.0 and Qemu 0.15.1. We make use of Kemari [2] to replicate a VM state on another physical host. Kemari transfers memory pages to a backup machine in an iterative manner like Remus [1].

Our prototype consists of three modules: *softpage monitor*, *address keeper*, and *page sender*. The softpage monitor, which runs in the guest Linux kernel, lists up the addresses of soft pages and sends them to the host Linux kernel. It also updates the kernel data objects for softpages when the backup VM begins. The address keeper, which runs in the host Linux kernel, receives the address of soft pages from softpage monitor and maintains the address list. In addition, it sends an event to trigger the update of kernel data objects to be performed by the softpage monitor. The page sender, which runs in a QEMU process, avoids sending softpages to the backup machine.

Behavior of the current prototype is overviewed in Fig. **??**. Fig. **??**(a) shows prototype behavior during normal operation. The softpage monitor traces soft pages, lists up their guest physical addresses, and periodically notifies the address keeper of the addresses vis a hyper call. The address keeper receives the addresses of softpages, maintains the addresses, and returns them to the page sender. The page sender avoids to sending softpages of the target VM, following the address list received from the address keeper.

Fig.**??**(b) shows how our prototype behaves when a failure occurs. To successfully continue services after the active VM failed, the backup VM first updates the kernel objects to keep the consistency of the kernel state. When the control is transferred to a backup VM, the address keeper inserts a virtual interrupt so that the softpage monitor updates kernel objects for softpages. This means that the update is the first task the kernel running on the backup VM performs. And then, the backup VM performs normal operations.

3.1 Softpage monitor

To effectively notify the address keeper about which pages are softpages, the softpage monitor periodically issues a hyper call. Our prototype uses this information as a *hint* to select which pages are necessary to transfer. When we finished setting up a backup VM, the softpage monitor starts to trace which pages are softpages. The softpage monitor memorizes the addresses of the detected softpages. The addresses are maintained as the list, and the pointer of the list is passed from the softpage monitor to address keeper. The current prototype issues the hyper call in 1 second. Also, in the current implementation, the softpage monitor lists up the address of buffer cache pages only. We have a plan to extend the implementation to trace other softpages such as free pages.

Note that this design releases us from strictly notifying the VMM of the addresses of softpages. Our prototype transfers dirty pages even if the pages are softpages. By doing so, we guarantee that pages containing the hard-state kernel objects are copied to the backup machine. The guest kernel can be resumed with the consistent state by combining this notification and the update of the kernel state.

3.2 Address keeper

The address keeper maintains softpages addresses informed by the softpage monitor. When the address keeper receives the addresses via a hyper call, it traces the list. To save memory space, it generates a bitmap that shows which pages are softpages. The bitmap is updated every time the address keeper receives the hyper call from the softpage monitor. address keeper returns the bitmap when the page sender requests it to know which pages should be sent. The page sender sends pages of the active VM, following our bitmap and a dirty page bitmap that is originally managed by the Kemari code in QEMU. The details are described in the next section.

The address keeper also inserts a virtual interrupt when the backup VM begins so that the softpage monitor updates the kernel data for managing soft-state kernel objects. When the active VM failed, control is transferred to the backup VM. At this time, the address keeper inserts a virtual interrupt with Intel VT feature. Specifically, the address keeper modifies the virtual machine control data structure (VMCS) of the backup VM to input an interrupt for the update of the kernel objects inside the guest kernel. The modification is performed when the execution of the backup VM is triggered.

3.3 Page sender

The page sender makes use of a part of the original Kemari functions in QEMU to transfer pages of the active VM to a backup machine. To create a backup VM, Kemari first sends all pages of the active VM to the backup machine and then iteratively sends dirty pages. To detect dirty pages, Kemari manages a bitmap that shows which pages get dirty in an interval. The bitmap is generated by the dirty page tracking feature supported by KVM. KVM starts/stops to track dirty pages by using system calls related to KVM. The result of the tracking is summarized into a bitmap.

The page sender leverages a dirty bitmap and a softpage bitmap to transfer pages of active VMs without softpages. When the page sender sends pages of the active VMs, it first gets a bitmap of softpages addresses from the address keeper and transfers pages that is not marked in the bitmap. This means that the page sender does not send pages of softpages. After sending the all pages, the page sender iteratively sends dirty pages and updated softpages by using a bitmap that is the conjunction of both bitmaps. The reason why updated softpages are sent to the backup machine is that the softpages gets to contain a hard-state kernel object. Following the bitmap, the page sender selects pages to be transferred.

4. Preliminary Experiments

To confirm the effectiveness of our prototype, we conducted experiments. In this thesis, we answer two fundamental questions;

(1) How does our prototype reduce memory usage of a backup VM?
(2) How much overhead does our prototype incur?

4.1 Experimental Setup

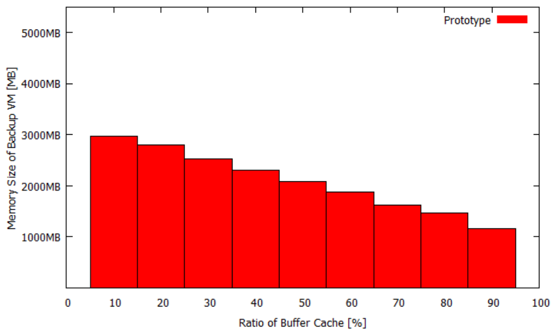
We used three DELL Poweredge T610 machines each of which has an Intel Xen 2.8GHZ Quad-Core processor with 32 GB of memory and SAS 500 GB HDD. The machines are connected to each other through a Gigabit Ethernet. We ran our prototype on the two machines; one is the active machine while the other is the backup machine. Also we ran an NFS server on the other machine. The NFS server machine is mounted by the other machines. We ran a VM whose memory size is 4 GB of memory. After starting a VM, we replicate it on the backup machine with our prototype.

4.2 Memory Usage of Backup VM

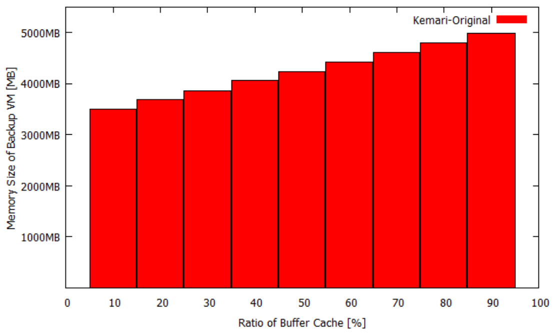
To demonstrate how our prototype reduces memory usage of a backup VM, we measured its memory size with several benchmarks. The benchmarks include an artificial workload and real applications. The artificial workload creates specified size of buffer cache. We changed the buffer cache size from 10% of the memory assigned to VM to 90%. The real applications are *postmark*, *Make*, and *Gzip*. Postmark is modeled after an e-mail server. Make compiles a Linux kernel source. Gzip compresses 2 GB of a file.

Fig.3 and Fig.4 are the result. The memory size of the backup VM with the artificial workload is shown in Fig.3. The result reveals that our prototype reduces more memory size as the buffer cache becomes bigger. When ratio of the buffer cache is 10%, the backup VM memory is 2909 MB. This means our prototype reduces the backup VM memory by 14.95%, compared to the original Kemari. When the ratio is 90%, it reduces the backup VM memory by 76.64%.

The result of the real applications is shown in Fig.4. In the graphs, we can see that our prototype can reduce the backup VM memory effectively. In the case of postmark, we can reduce the memory size by 1416 MB, compared to the original Kemari. Our prototype also reduce the memory size of the backup VM by 237 MB In GNU Make experimentation. In addition, our prototype reduces the memory by 1242 MB in the case of Gzip.



(a) Our prototype



(b) Original Kemari

図 3 Memory usage of backup machine's QEMU process with our artificial workload.

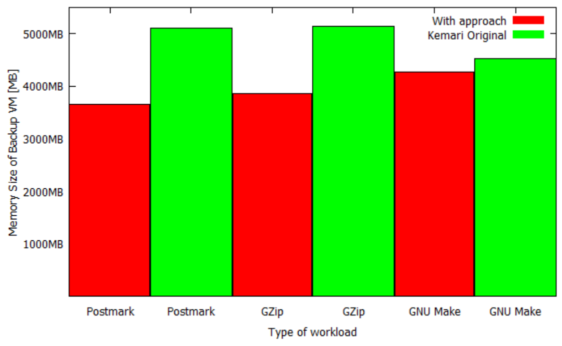


図 4 Memory usage of backup machine's QEMU process with real applications

表 1 Overhead of our prototype

Workload	prototype	Original
Postmark Total Transaction Time	111 sec	83 sec
Postmark Total Data read	24.38 MB/sec	29.8 MB/sec
Postmark Total Data write	28.94 MB/sec	35.37 MB/sec
Gzip Total time	137.674 sec	124.973 sec
GNU Make Total time	58m28.351s	48m41.907s

4.3 Overhead

To show the overhead of our prototype, we measured a performance penalty with the real applications described in the previous section. Compared to the original Kemari, our system additionally performs a hyper call of the softpage monitor, the softpages addresses maintenance of the address keeper, and the bitmap calculation of the page sender. We observed the throughput of the applications when we ran the original Kemari and our prototype. We recorded the benchmark score of postmark, the execution time of Make and Gzip as their throughput.

The result is exhibited in Tab.1. The table reveals that the overhead incurred by our prototype is negligible. In the case of postmark, our prototype incurs only 33.3% penalty. Also, the overhead is 10.2% when we ran Gzip.

5. Related Work

Remus [1] and Kemari [2] are typical systems that synchronize the whole status of virtual machines between primary and backup machines. Bressoud and Schneider [3] is based on deterministic replay of events to keep the consistency between primary and backup machines. These systems assume that backup machines have the same capacity as primary machines, and do not address the consolidation of backup VMs.

RemusDB [4] extends Remus to protect a database management system (DBMS). Since the DBMS is highly memory-intensive, the synchronization overhead is terrible. RemusDB compresses the delta of frequently updated pages to reduce the transferred data, and also ignores pages in the buffer pool of DBMS. Although RemusDB does not concern about backup VM consolidation, these techniques can be combined with the backup VM consolidation.

Reducing the size of VM images is addressed in various contexts. Park et al. [5] propose a technique to reduce the size of checkpoints. It tracks I/O operations so as to avoid saving the pages in non-volatile storage. In the context of live migration, the post-copy approach [6] reduces the total number of transferred pages by sending them on demand. MECOM [7] and Svaard et al. [8] compress the transferred pages. CR/RT-Motion [9] transfers execution trace logs to replay the execution on the destination. SonicMigration [10] avoids sending softpages.

6. Conclusion

Handling hardware failures is crucial to achieve high available services in virtualized environments since a hardware failure affects all the VMs on the failed physical machine. Although the VM replication is an attractive technique to survive hardware failures, it requires backup machines to have the same capacity as primary machines. This diminishes the cheap nature of the virtualization-based high availability.

This paper describes *backup VM consolidation*, which advances the cheap nature of virtualization-based high availability. The backup VM consolidation distills memory images that must be synchronized between primary and backup machines, and enables two or more primary machines to be protected by a single backup machine. This design of the backup VM consolidation gives greater flexibility in the trade-off between availability and the cost of backup machines. Our prototype based on Linux 3.6.0 and QEMU 0.15.1 successfully reduces the memory size of backup VMs, and enables the consolidation of backup VMs.

参考文献

[1] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N. and Warfield, A.: Remus: High Availability via Asynchronous Virtual Machine Replication, *Proceedings of the 5th USENIX Symposium on Networked Systems Design and implementation (NSDI '08)*, pp. 161–174 (2008).

[2] Nippon Telegraph and Telephone Corporation: Kamari Project,http://www.osrg.net/kemari/ (2008).

[3] Bressoud, T. C. and Schneider, F. B.: Hypervisor-Based Fault Tolerance, *ACM Transaction Computer System*, Vol. 14, No. 1, pp. 80–107 (1996).

[4] Minhas, U. F., Rajagopalan, S., Cully, B., Aboulnaga, A., Salem, K. and Warfield, A.: RemusDB: Transparent High Availability for Database Systems, *PVLDB*, Vol. 4, No. 11, pp. 738–748 (2011).

[5] Park, E., Egger, B. and Lee, J.: Fast and Space-Efficient Virtual Machine Checkpointing, *Proceedings of the 7th ACM International Conference on Virtual Excution Environments (VEE '11)*, pp. 75–86 (2011).

[6] Hines, M. R. and Gopalan, K.: Post-Copy Based Live Virtual Macine Migration Using Adaptie Pre-Paging and Dynamic Self-Ballooning, *Proceedings of the 2009 ACM International Conference on Virtual Execution Environments (VEE '09)*, pp. 51–60 (2009).

[7] Jin, H., Deng, L., Wu, S., Shi, X. and Pan, X.: Live Virtual Machine Migration with Adaptive Memory Compression, *Proc. of 2009 IEEE International Conference on Cluster Computing (Cluster '09)*, pp. 1–10 (2009).

[8] Sväard, P., Hudzia, B., Tordsson, J. and Elmroth, E.: Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines, *Proceedings of the 7th ACM International Conference on Virtual Excution Environments (VEE '11)*, pp. 111–120 (2011).

[9] Liu, H., Jin, H., Liao, X., Hu, L. and Yu, C.: Live Migration of Virtual Machine Based on Full System Trace and Replay, *Proc. of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC '09)*, pp. 101–110 (2009).

[10] Koto, A., Yamada, H., Ohmura, K. and Kono, K.: Towards Unobtrusive VM Live Migration for Cloud Computing Platforms, *Proceedings of the 3rd ACM Asia-Pacific Conference on Systems (APSys '12)*, pp. 7–7 (2012).