Preventing Performance Degradation on Operating System Reboots

Kenichi Kourai
Tokyo Institute of Technology
kourai@is.titech.ac.jp

1. Introduction

The reboots of operating systems (OSes) are not avoidable because OSes have many bugs. When OSes crash or slow down critically, the administrators must reboot them. Although patches for fixing such bugs and security problems are frequently released, critical ones related to OSes and system libraries need the OS reboots. A technique called software rejuvenation [2] has been also proposed to avoid unplanned reboots due to software aging, but the simple implementation for the OS rejuvenation is to reboot the OSes.

Just after an OS is rebooted, its performance is degraded. The primary cause is to lose the file cache. An OS stores file contents in main memory as the file cache when it reads them from disks. An OS speeds up file accesses by using the file cache on memory. When an OS is rebooted, main memory is initialized and the file cache managed by the OS is lost. To fill the file cache after the reboot, an OS needs to read necessary files from slow disks. Since modern OSes use most of free memory as the file cache, it takes long time to fill free memory with the file cache.

Particularly, for servers consolidated with virtual machines (VMs), the performance degradation affects not only a rebooted VM itself but also all the other VMs on the same machine. The conflicts of disk accesses makes the performance of all the VMs degraded. Just after the OS in a VM is rebooted, it frequently accesses a physical disk. Since the disk is a shared resource for all VMs, increasing disk accesses in one VM affects the performance of the disk access by the other VMs. Also, prefetching does not work well in this kind of system because it issues too many disk accesses during a short period.

To reduce such performance degradation due to rebooting OSes, we propose a new reboot mechanism for OSes, which is called the *warm-cache reboot*. This mechanism enables an OS to be rebooted without losing the file cache. We claim that the file cache does not need to be discarded by the reboot as long as the integrity of the file cache is preserved. The purpose of the reboot is to initialize the internal state or to update the components in an OS kernel. The warm-cache reboot guarantees the integrity using the virtual machine monitor (VMM), which is an underlying software layer for VMs.

2. Warm-cache Reboot

2.1. Preserving the File Cache

The basic idea of the *warm-cache reboot* is to preserve the file cache on memory during the OS reboot and restore it after the reboot. This is implemented by the cooperation of an OS and the VMM. To preserve the file cache during the reboot, the VMM allocates the same physical memory as before the reboot to a rebooted VM. At this time, the VMM does not erase the contents of the memory. Normally, the contents are erased for security because the memory pages may include sensitive information used by another OS. For the warm-cache reboot, reusing the memory pages without erasing the contents is secure because it is guaranteed that those pages are reused for the same OS.

Each OS on a VM manages the relationship between the information on file blocks and cache pages using a *cache-mapping table*. This table is a hash table whose keys are a tuple of a device number, an inode number, and a file offset, and whose value is a page frame number assigned to a memory page used for the file cache. When the OS reads a file block from a disk and allocates a new cache page, it adds a new entry to the cache-mapping table. When the OS discards the file cache, it removes the corresponding entry from the table.

When the OS is rebooted, its kernel reserves the memory areas for the cache-mapping table and the file cache so that the contents are not used for other purposes. This reservation is performed at the early stage of booting the OS kernel, that is, before the kernel starts dynamic memory allocation. The kernel first obtains the address of the cache-mapping table preserved by the VMM and reserves its memory pages. Then, it reserves cache pages, based on the entries in the table. To reuse the file cache that was stored before the reboot, the OS looks up the restored table.

2.2. Protecting the File Cache

The VMM prevents the file cache from being corrupted by OS crashes. The OS protects cache pages in a read-only manner using the functionality of the VMM. Therefore, the OS can directly read cache pages without any overheads. In addition, the pages are protected *before* a file block is read from a disk so that the contents are not corrupted during file reads. This is impossible if the VMM is not involved because the OS must perform the write access to the pages at file reads. To enable this, our system installs a virtual disk device driver into the OSes like many other VM implementation. The driver passes a memory page to the VMM and the VMM reads a file block from a disk into the page using a real disk device driver. The VMM can perform the write access to the pages protected by the OS on a VM.

When the VMM completes to read a file block, it sets the *reuse flag* to the corresponding entry in the cache-mapping table if the cache page is still protected in the VM. The OS reuses the page as the file cache if the flag is set when the OS is rebooted. On the other hand, when the OS attempts to write the file block, it unprotects the cache page. At the same time, the VMM reset the reuse flag. The unprotected cache page is not reused because the integrity between the file cache and file blocks on a disk may be lost. Only the VMM can set the reuse flag at the completion of file reads. Therefore, it is guaranteed that the cache page is protected while the flag is set.

Also, the VMM protects the cache-mapping table using memory protection. Since the cache-mapping table is write-protected when it is created, the OS can access the cache-mapping table without any overheads. To safely modify the cache-mapping table, the VMM provides a new hypervisor call, which is a call to a VMM and similar to a system call to an OS. When the OS needs to add a new entry to this table or remove an existing entry, it issues the hypervisor call. The hypervisor call checks the request and modifies the table. The OS cannot modify the table without using the hypervisor call. The possibility of wrongly issuing these hypervisor calls is low, compared to that of directly corrupting the table without protection.

The warm-cache reboot is similar to the Rio file cache [1], but there are two main differences. One is that Rio enables the OS to reuse only dirty file cache. The purpose of Rio is to prevent data that is not written back to a disk from being lost by OS crashes. The warm-cache reboot reuses a larger amount of non-dirty file cache to prevent performance degradation after the OS reboot. The other difference is how to protect cache pages. Rio protects cache pages using the functionality of the OS while the warm-cache reboot uses that of the VMM. The OS itself may not be able to protect its file cache when it crashes.

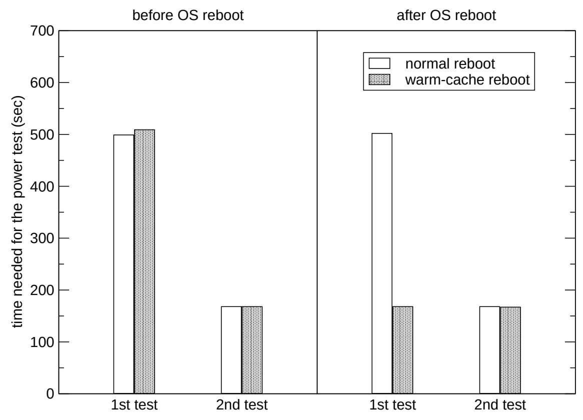

Figure 1. The results of the DBT-3 benchmark.

3. Experiments

We have developed our VMM and Linux, which are based on Xen 3.0.0 and Linux 2.6.12, respectively. Using our system, we performed experiments to show the usefulness of the warm-cache reboot. We used a PC with two dual-core Opteron processors Model 280, 12 GB of memory, and a 146 GB of Ultra-320 SCSI disk. We allocated 11 GB of memory to one VM. One physical partition of the disk was used for a virtual disk of the VM.

To examine performance degradation due to cache misses, we measured the time needed for the power test with the scale factor of one in DBT-3 before and after the OS reboot. DBT-3 is a benchmark tool for databases and its power test measures the performance of the read access to databases. To examine the effect of the file cache, we measured the performance of the first and second tests. We used PostgreSQL and all the file blocks were cached on memory in this experiment. We performed this experiment for the warm-cache reboot and the normal reboot.

Figure 1 shows the results. When we used the normal reboot, the performance just after the reboot was degraded by 67 %, compared with that just before the reboot. On the other hand, when we used the warm-cache reboot, the performance was not degraded. This improvement was achieved by no miss in the file cache even when a file was accessed at the first time after the reboot.

References

[1] P. Chen, W. Ng, S. Chandra, C. Aycock, G. Rajamani, and D. Lowell. The Rio File Cache: Surviving Operating System Crashes. In *Proc. Int'l Conf. ASPLOS*, pages 74–83, 1996.

[2] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton. Software Rejuvenation: Analysis, Module and Applications. In *Proc. Int'l Symp. Fault-Tolerant Computing*, pages 381–391, 1995.