CacheMind: Fast Performance Recovery Using a Virtual Machine Monitor

Kenichi Kourai
Kyushu Institute of Technology
kourai@ci.kyutech.ac.jp

Abstract

The reboot of an operating system is a final but powerful recovery technique. However, the system performance is largely degraded just after the reboot due to losing the file cache. For fast performance recovery, we propose a new reboot mechanism called the warm-cache reboot. *The warm-cache reboot preserves the file cache on main memory during the reboot and enables an operating system to restore the file cache after the reboot. A virtual machine monitor (VMM) underlying an operating system guarantees that the reused file cache is consistent with the corresponding files on disks. We have implemented the warm-cache reboot mechanism in the Xen VMM and the Linux operating system. From our experimental results, the warm-cache reboot decreases performance degradation just after the reboot. In addition, we confirmed that the file cache corrupted by faults was not reused.*

1. Introduction

The reboot of an operating system is frequently used for the recovery of the whole system. When an operating system crashes due to Mandelbugs [1], it can usually recover from the crash by its reboot. Since the causes of Mandelbugs are so complex, the rebooted operating system rarely crashes again. The reboot is also used as a simple method for software rejuvenation [2]. Software rejuvenation [3] is a proactive technique to counteract software aging, which is the phenomenon that the state of running software degrades with time. Even if an operating system slows down due to aging-related bugs [1] such as memory leaks, the rebooted operating system can easily restore its normal state.

However, the system performance is largely degraded just after the reboot of an operating system. The primary cause is to lose the file cache. An operating system stores file contents in main memory as the file cache to speed up file accesses. When an operating system is rebooted, the contents of main memory are erased and the file cache is lost. Without the file cache, an operating system has to read files from slow disks. Worse, if the operating system runs in a virtual machine (VM), such cache misses may largely affect the whole system performance because disks are shared between VMs. The conflicts of disk accesses degrade the performance of not only the rebooted VM but also the other VMs.

For fast performance recovery, we propose a new reboot mechanism called the *warm-cache reboot*. The warm-cache reboot preserves the file cache on main memory during the reboot and enables an operating system to restore the file cache after the reboot. This mechanism prevents performance degradation caused by frequent cache misses. To maintain the consistency of the file cache after the reboot, we use a virtual machine monitor (VMM), which runs underneath an operating system. The VMM maintains the consistency of the file cache of operating systems by managing whether the contents of the file cache are the same as those of corresponding files on disks. Such a software layer like a VMM is necessary to reuse only consistent file cache through the crashes of an operating system.

We have developed *CacheMind* based on Xen [6] and implemented the warm-cache reboot mechanism in the VMM and the Linux operating system running on top of it. The VMM manages a *P2M-mapping table* for continuous memory allocation to rebooted VMs, *cache-mapping tables* for reusing the file cache after the reboot, and *reuse bitmaps* for maintaining cache consistency. From our experimental results, the performance degradation just after the warm-cache reboot was from 5 % to 16 % while that just after a normal reboot was 41 % to 90 %. In addition, our fault-injection test showed that a part of the file cache could be corrupted by faults but it was not reused by the consistency mechanism of the warm-cache reboot.

The rest of this paper is organized as follows. Section 2 describes problems of recovering the system performance after the reboot of an operating system. Section 3 presents the warm-cache reboot and Section 4 shows our experimental results. Section 5 examines related work and Section 6 concludes the paper.

2. Performance Recovery

After the reboot of an operating system, the system performance is largely degraded for a while. The primary cause is to lose various caches in an operating system. Particularly, losing the file cache largely affects the performance. An operating system stores file contents in main memory as the file cache when it reads them from disks. Since disks are much slower than main memory, an operating system can speed up file accesses by using the file cache on memory. When an operating system is rebooted, the contents of main memory are erased and the file cache managed by the operating system is lost. Just after the reboot of an operating system, the execution performance of applications running on top of it is degraded due to frequent cache misses.

It takes long time to recover the same performance as before the reboot. For regaining the same performance, an operating system has to re-fill the file cache. However, modern operating systems use most of free memory as the file cache for performance improvement. The amount of the file cache is almost equivalent to that of free memory, which tends to be larger because the size of memory installable to one machine is increasing due to cheaper memory modules. In addition, widespread 64-bit processors enable an operating system to deal with more than 4 GB of memory. Until the file cache is re-filled, an operating system has to read necessary files from slow disks and cannot recover the performance.

In a VM environment, the performance recovery needs longer time after an operating system in a VM is rebooted. Recently, server consolidation is performed with VMs for cost efficiency. In such an environment, physical disks are often shared among VMs. Although a different physical disk may be allocated to each VM exclusively, that is usually difficult due to physical constraints or its cost. In other words, one VM cannot occupy the whole disk bandwidth. Worse, disk bandwidth allocated to each VM may be limited for fairness. Since this disk sharing degrades the throughput of disk accesses, it takes longer time to re-fill the file cache by reading files from disks.

On the other hand, frequent disk accesses affect not only a rebooted VM but also all the other VMs. The conflicts of disk accesses degrade the performance of all the VMs. Just after an operating system in a VM is rebooted, it frequently accesses a physical disk. Increasing disk accesses in one VM affects the performance of the disk access by the other VMs. From the same reason, prefetching does not work well in a VM environment. Prefetching is a popular technique for hiding the initial cache misses particularly when the system is booted. Files are read from disks in advance before they are accessed. Prefetching issues too many requests for disk accesses during a short period because it is batch processing, not on-demand. This gives worse influences to the performance of the other VMs.

Thus, recovery by the reboot does not complete until the system performance is recovered. When an operating system is booted and all the applications on top of it are started, the system can start providing the same services as before the reboot. To make this reboot procedure faster, several techniques have been proposed [4, 5]. However, the system does not restore the same performance at that time. For example, server processes can accept new connections, but they may not return quick responses due to performance degradation caused by frequent cache misses. Such performance degradation lasts until the file cache is re-filled. The size of the file cache tends to increase as the size of main memory becomes large.

3. Warm-cache Reboot

To quickly recover the system performance after the reboot of an operating system, we propose a new reboot mechanism called the warm-cache reboot. The warm-cache reboot is achieved by the cooperation of an operating system and the underlying VMM. To use the VMM, operating systems run on VMs created by the VMM. A VMM is a useful software layer underlying operating systems to preserve the file cache through the reboot and maintain its consistency.

3.1. Preserving the File Cache

The basic idea of the warm-cache reboot is to preserve the file cache on memory during the reboot and enable an operating system to restore the file cache after its reboot. We believe that the file cache does not need to be discarded at a reboot. The purpose of a reboot is to initialize the internal state in an operating system or to update its components such as its kernel. Even if the data structures in an operating system are changed by its update, the contents of the file cache are reusable because they are just the copies of file blocks on disks. However, an operating system should not reuse corrupted file cache. Rather, it should read file blocks from disks. The warm-cache reboot discards only corrupted file cache by the consistency mechanism described in Section 3.2.

By reusing the file cache, the warm-cache reboot prevents performance degradation caused by cache misses just after the reboot. In other words, it recovers the system performance as well as the functionality. After the reboot, most of files accessed are expected to exist on the file cache as far as a working set is within the size of the file cache. The workload is not largely changed between before and after the reboot because the time needed for the reboot is not so long. Normally, the files accessed during the reboot are not

included in the working set just before the reboot. However, the access would just replace a very small part of the file cache in many cases.

While an operating system in a VM is rebooted by using the warm-cache reboot mechanism, the VMM preserves the contents of the memory allocated to the VM. The VMM allocates the same physical memory as before the reboot to the VM. The memory layout is the same as well. Without the VMM, it is not guaranteed that the contents of physical memory are preserved because of a hardware reset. A hardware reset may corrupt a part of memory, depending on hardware and temperature [7]. When the VMM reallocates physical memory, it leaves the contents of the memory as it is. Normally, when the VMM allocates memory pages to VMs, it erases the contents for security. The memory pages may include sensitive information used by another operating system. At the warm-cache reboot, reusing memory pages without erasing the contents is secure. Those pages are necessarily reused for the same operating system.

A rebooted operating system reserves all the memory pages used for the file cache. We call such memory pages *cache pages*. This reservation is performed at the early stage of booting the kernel, that is, before the kernel starts dynamic memory allocation. This prevents the cache pages from being used for other purposes and corrupted. Since the cache management in an operating system is initialized by the reboot, the VMM manages information for reusing the file cache of an operating system. When an operating system allocates a cache page, it registers the page to the VMM, with the information on the corresponding file block. When an operating system uses that page for other purposes, it unregisters the page. When an operating system is rebooted, it obtains the information on the file cache from the VMM.

3.2. Maintaining Cache Consistency

The warm-cache reboot reuses the file cache only when it is guaranteed that the file cache is consistent. We assume that the file cache is consistent when the contents of the file cache are the same as those of file blocks on disks. When a file block is read from a disk to a cache page, the cache page is consistent. After the cache page is modified by file writes or destroyed by faults, it becomes inconsistent. When the cache page is written back to a disk, it becomes consistent again.

The VMM maintains the consistency of each cache page. In a VM environment, device accesses in an operating system running on a VM are performed via the VMM. The VMM reads data on a disk into a cache page passed from an operating system or writes data in a cache page into a disk, as illustrated in Figure 1. When disk reads and writes complete, the VMM makes the cache page reusable because it is guaranteed that the contents of the cache page is the same as those of a file block on a disk. We assume that the VMM works as intended and disk reads and writes are performed correctly. Since the VMM is much smaller software than an operating system, this assumption is reasonable.

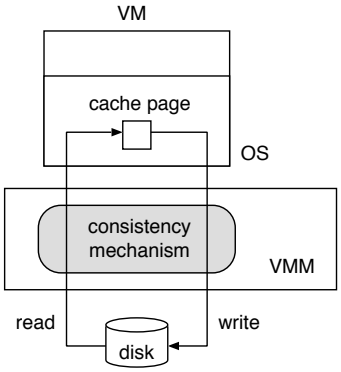

Figure 1. Tracking the consistency of the file cache.

To track the cache consistency, the VMM protects cache pages in a read-only manner. When the VMM reads a file block into a cache page, it protects the page *before* that file read so that it can detect the modification to the cache page. While a cache page is protected, it is reusable because it is guaranteed that the cache page is consistent. This memory protection also prevents a cache page from being corrupted by faults. When an operating system modifies the protected cache page to write data into a file, the VMM changes its protection mode to writable before that write so that an operating system can modify the cache page without any overhead. In this state, the cache page is not reusable because the page is not consistent. When the VMM writes back the contents of a cache page into a disk, it protects the page again before that file write. Thus the cache page becomes reusable again.

This cache consistency cannot be guaranteed without the VMM. An operating system cannot read data on disk into protected cache pages because they cannot write data in protected memory pages. Therefore, before an operating system protects cache pages, the contents in cache pages may be corrupted during disk reads. Since an operating system becomes unreliable by bugs, device drivers may not perform disk reads and writes correctly and the cache page may not be protected correctly. Even if the page is protected, the protection mode may be changed to writable by corrupting the page table. Although an operating system has to manage the reusability of cache pages, such management information may be corrupted accidentally. If that information is wrong, the warm-cache reboot reuses inconsistent cache pages.

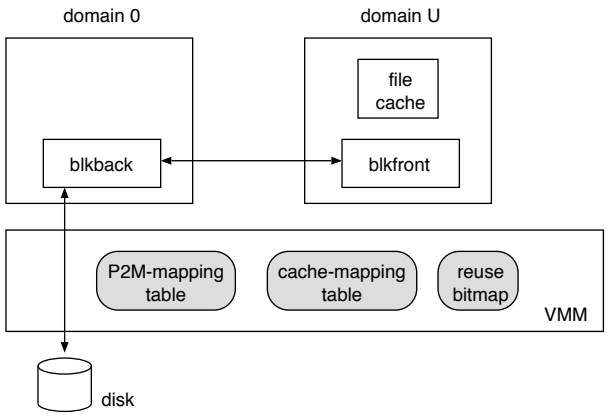

Figure 2. The system architecture.

The above definition of the cache consistency is stricter than ideal one. When the file cache is not corrupted, it can be reused even if the contents have not been written back to disk. This enables using the latest updates to files after an operating system is rebooted. However, it is difficult to distinguish correct modification from corruption because the correctness of modification depends on semantics. To avoid this semantic problem, we reuse a cache page only after the modification to the page is written back to a disk. Since the modification becomes persistent at that time, the cache page becomes reusable even if its contents are corrupted. In this situation, applications always use the corrupted file even without the file cache. The administrator should recover corrupted files, for example, from the backup.

3.3. Implementation

To achieve the warm-cache reboot, we have developed *CacheMind* based on Xen 3.0.0 [6]. Figure 2 shows the system architecture. Xen provides the VMM and VMs running on top of it. A VM is called a *domain* in Xen. In particular, the privileged VM that manages VMs and handles I/O is called *domain 0* and the other VMs are called *domain Us*. When an operating system in domain U accesses a virtual disk, its device driver called *blkfront* sends requests to the *blkback* driver in domain 0. The blkback driver accesses a physical disk drive and returns the results to the blkfront driver.

Our VMM manages a P2M-mapping table, cache-mapping tables, and reuse bitmaps. A *P2M-mapping table* is used for preserving the contents of the memory of domain U even through its reboot. In Xen, the VMM manages machine memory, which is physical memory installed in the machine. For each machine page frame, a machine frame number (MFN) is consecutively numbered from 0. The VMM allocates a part of machine memory to domains as pseudo-physical memory, which gives the illusion of contiguous physical memory to every domain. For each physical page frame, a physical frame number (PFN) is consecutively numbered. A P2M-mapping table is one-dimensional array that records mapping from PFN to MFN for each domain. Our VMM preserves the P2M-mapping table while a domain is rebooted.

A *cache-mapping table* is used for restoring the file cache after the reboot. It is a hash table whose keys are a tuple of a device number, an i-node number, and a file offset. Its value is a PFN assigned to a cache page. When an operating system reads a file block from a disk to a new cache page, it adds a new entry to this table by invoking a hypervisor call to the VMM. This hypervisor call performs the sanity check of a request and modifies the cache-mapping table atomically. Even if an operating system is unreliable, it cannot directly corrupt the table in the VMM. When an operating system is rebooted, its kernel first reserves cache pages to be reused, based on this table.

A *reuse bitmap* is used for maintaining the reusability of cache pages. It is a bitmap whose bit represents whether the corresponding pseudo-physical memory page is reusable as a file cache. When a file block is read from a disk to a cache page, the corresponding reuse bit is set if the contents of the page are not corrupted during disk I/O. To guarantee this, the VMM changes the protection mode of the page to read-only in domain U. If domain U changes the protection mode for writing the page, the VMM can detect it. In addition, the VMM checks that the page is not mapped in a writable manner during disk I/O anywhere except domain 0 because the blkback driver in domain 0 needs to write a file block into the page. Once the page has been mapped in a writable manner, there is a possibility of data corruption.

4. Experiments

We performed experiments to show that our technique is effective. For a server machine, we used a PC with two Dual-Core Opteron processors Model 280, 12 GB of PC3200 DDR SDRAM memory, a 36.7 GB of 15,000 rpm SCSI disk (Ultra 320), and Gigabit Ethernet NICs. We used our VMM and Linux 2.6.12 modified for the warm-cache reboot. For comparison, we used the original Xen 3.0.0 for a normal reboot. We used the 64-bit execution environment except experiments in Section 4.3. For a client machine, we used a PC with dual Core 2 Quad processors, 4 GB of memory, and Gigabit Ethernet NICs. The operating system was Linux 2.6.18.

4.1. File Access Performance

To examine performance degradation due to cache misses, we measured the throughput of file read access in

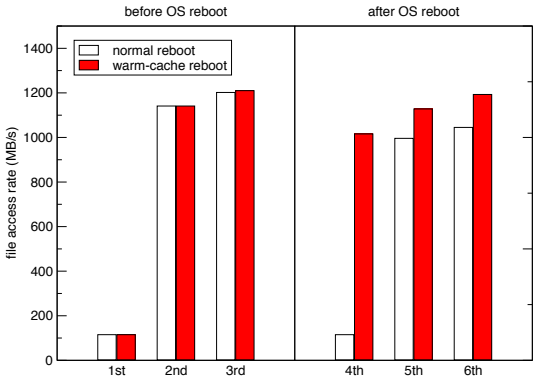

Figure 3. File access performance before and after a reboot.

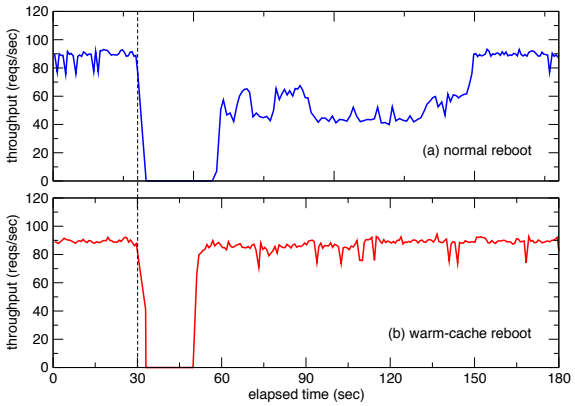

Figure 4. The changes of the throughput of a web server when an operating system is rebooted.

a VM before and after the reboot of an operating system. To examine the effect of the file cache, we measured the throughput of the first-, second-, and third-time accesses. We allocated 4 GB of memory to one domain U and 4 GB to domain 0. We measured the time needed to read a file of 1 GB. In this experiment, all the file blocks were cached on memory. We performed this experiment for the warm-cache reboot and a normal reboot.

Figure 3 shows the result. When we used a normal reboot, the throughput just after the reboot was degraded by 90 %, compared with that just before the reboot. The time needed for performance recovery was 8.9 seconds for a file of 1 GB. On the other hand, when we used the warm-cache reboot, the throughput just after the reboot was degraded only by 16 %. The time for performance recovery was 1 second. This improvement was achieved by no miss in the file cache even when a file was accessed at the first time after the reboot. The remaining performance degradation is caused by misses of other caches such as i-node cache. The reason why the third-time access is better than the second-time is because cache pages are linked at the active list in Linux at the second time.

4.2. Web Server

We measured the changes of the throughput of a web server before and after the reboot of an operating system. The Apache web server [8] 2.0.54 served 4000 files of 1 MB and httperf [9] 0.8 in a client host sent requests to the server one by one. Since we allocated 11 GB of memory to one domain U, all the files served by the web server were cached on memory. We allocated 512 MB of memory to domain 0.

Figure 4 shows the changes of the throughput of a web server when we used a normal reboot and the warm-cache reboot. We executed the reboot command in domain U at time 30 seconds. When we used the warm-cache reboot, the throughput was degraded only by 5 % after the reboot. In 60 seconds after the web server restarts its service, the throughput is recovered completely. On the other hand, when we used a normal reboot, the throughput was degraded by 41 % on average. The performance degradation lasts for 90 seconds after the web server restarts its service. During this period, the web server loses the profit to be gained by about 3300 requests, compared with before the reboot.

4.3. Fault Injection

We injected faults to an operating system in domain U and examined the consistency of reused cache pages. We have ported the fault injection tool used in the Nooks [10] project to the Linux 2.6 kernel. Originally, the tool was developed for the Rio file cache [11] project. Since the tool developed by the Nooks project strongly depends on Intel 32-bit architecture, we used the 32-bit execution environment.

This tool injects various types of faults. DST flips a random bit of the destination of an instruction. PTR flips a random bit of the address for memory reference. INIT deletes an instruction that initializes a local variable on the kernel stack. I/F deletes an instruction that reads a function parameter. BR deletes a branch instruction or a repeat prefix. LOOP inverts the termination condition for a repeat prefix or a branch instruction. PANIC causes a kernel panic. ALLOC returns NULL at the kmalloc function. FREE releases a memory region that is still used. LEAK does not release a memory region at the kfree function. COPY overruns the length of memory copy by one byte to four pages. TEXT flips a random bit of a random instruction in the kernel. STACK flips a random bit in a stack of a random pro-

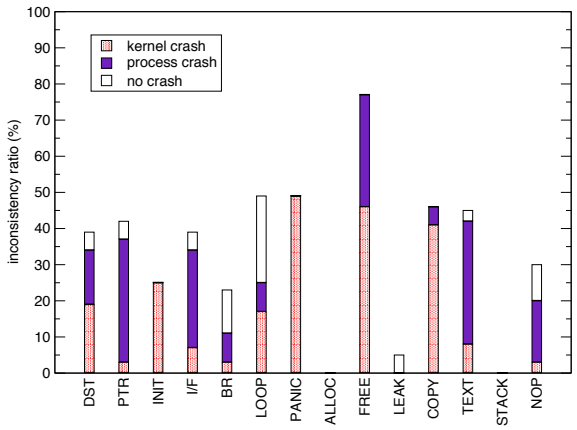

Figure 5. The ratio of cache inconsistency when the consistency mechanism was disabled.

cess. NOP deletes a random instruction in the kernel.

We examined the consistency of the file cache after we injected faults into an operating system and rebooted it using the warm-cache reboot. First, we boot an operating system in a VM and waits until the file cache is filled by sending HTTP requests. Then we injected ten faults of the same type into the kernel and waited for 60 seconds. Finally we rebooted the operating system and checked the consistency of the file cache by comparing it with files on a disk. We repeated this fault injection 50 times for each fault type.

Figure 5 shows the ratio at which the file cache was inconsistent when the consistency mechanism in the VMM was disabled. The VMM did not manage the file cache. Instead, an operating system managed a cache-mapping table and a reuse bitmap without the help of the VMM. For most of fault types, the file cache was inconsistent at a high ratio. This figure also shows the breakdown of the results of this fault injection when the file cache was inconsistent. Although fault injection did not always cause crash, the file cache became inconsistent.

When the consistency mechanism in the VMM was enabled, the file cache was consistent for all fault types except DST. According to our deep inspection, some faults were injected into the ext3 file system in this exceptional case. Then the file system failed to write back cache pages to a disk. This resulted in the inconsistency between the file cache and files on the disk. However, the contents of the file cache were correct while those of files on the disk were incorrect. Therefore, reusing the file cache is correct although the consistency is not maintained.

5. Related work

The Rio file cache [11] enables dirty file cache to survive crashes of an operating system. When an operating system crashes, Rio saves dirty cache pages to a disk and prevents any modification to files from being lost by the reboot. The biggest difference between Rio and CacheMind is that Rio is designed for reliability while CacheMind is for high performance. When an operating system is rebooted, Rio discards non-dirty cache pages because saving them is not necessary for improving reliability. To the contrary, CacheMind reuses non-dirty cache pages but discards dirty ones because dirty cache pages are inconsistent with a disk. In addition, because Rio has to read saved file cache from a slow disk, the performance is degraded just after the reboot. CacheMind prevents such performance degradation by reusing the file cache preserved on memory.

The other big difference is that Rio relies only on an operating system (and hardware) while CacheMind relies on the VMM. For example, Rio provides two mechanisms to save the file cache to a disk on a crash. One is to perform a warm reboot, which preserves memory contents during the reboot, and save dirty file cache after the reboot [11]. The other is to save the file cache using a BIOS routine before a reboot [12]. The former depends on hardware and is not generally supported in PCs. The latter might fail because Rio does not always execute the BIOS routine after a crash. In CacheMind, an operating system in a VM can perform a warm reboot, independent of hardware, because the VMM guarantees to preserve memory contents during the reboot of the VM.

Besides, Rio uses memory protection to prevent the file cache from being corrupted by crashes of an operating system. Rio protects the file cache by using functions in an operating system while CacheMind protects it by the VMM. In Rio, if the page table is corrupted by a crash of an operating system, memory protection might be ineffective. In CacheMind, although the page table may be corrupted, the VMM tracks any modification and maintains the reusability of the file cache. Also, Rio cannot atomically modify its registry for cache management because the registry is also managed by an operating system. Therefore, the consistency of the registry is not guaranteed when an operating system crashes. In CacheMind, a cache-mapping table for cache management is managed by the VMM. The modification of the table can be atomically performed by the VMM.

Non-volatile disk cache such as Microsoft hybrid hard drive (HHD) and Intel Turbo Memory is useful for fast performance recovery. HHD includes non-volatile memory inside a disk drive while Turbo Memory is attached to a motherboard. They enable an operating system to read file blocks from fast non-volatile memory even if the file cache on memory is lost after the reboot. Furthermore, a

solid-state drive (SSD) speeds up the whole disk accesses by using flash memory. However, it is necessary to copy file blocks from non-volatile memory to the file cache on main memory. CacheMind does not need any memory copies because it preserves the file cache on main memory through the reboot.

Recovery Box [13] preserves the state of an operating system and applications on non-volatile memory for fast recovery. It restores the state quickly after rebooting an operating system. The state stored in that memory is protected by checksum. In addition, Recovery Box speeds up a reboot by reusing the kernel image left on memory. This is less effective recently because recent disks are fast enough to read the file for the kernel image. RootHammer [14] enables quickly rebooting only the VMM by leaving VM images on memory. It uses the fact that VM images can be reused after the reboot of the VMM. Similarly, CacheMind uses the fact that the file cache can be reused after the reboot of an operating system as far as it is not corrupted.

6. Conclusion

In this paper, we proposed a new reboot mechanism, called the *warm-cache reboot*, for fast performance recovery. The warm-cache reboot preserves the file cache on main memory during the reboot and restores the file cache quickly after the reboot. The VMM guarantees that the file cache reused after the reboot is consistent with the corresponding files on disks by maintaining reuse bitmaps. We have implemented the warm-cache reboot mechanism in Xen. From our experimental results, when we used the warm-cache reboot, the performance degradation just after the reboot of an operating system was 16 % at maximum. In addition, it was shown that the reused file cache was not corrupted by faults. One of our future work is to reuse other caches in an operating system such as i-node cache for improving the performance after the reboot.

Acknowledgments

This research was supported in part by JST, CREST.

References

[1] M. Grottke and K. S. Trivedi, "Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate," *IEEE Computer*, vol. 40, no. 2, pp. 107–109, 2007.

[2] S. Garg, A. Puliafito, M. Telek, and K. Trivedi, "Analysis of Preventive Maintenance in Transactions Based Software Systems," *IEEE Trans. Computers*, 1998.

[3] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, "Software Rejuvenation: Analysis, module and Applications," in *Proc. Int'l Symp. Fault-Tolerant Computing*, 1995, pp. 381–391.

[4] A. Pfiffer, "Reducing System Reboot Time with kexec," http://www.osdl.org/.

[5] H. Kaminaga, "Improving Linux Startup Time Using Software Resume," in *Proc. Linux Symp.*, 2006.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proc. Symp. Operating Systems Principles*, 2003, pp. 164–177.

[7] J. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten, "Lest We Remember: Cold Boot Attacks on Encryption Keys," in *Proc. USENIX Security Symp.*, 2008, pp. 45–60.

[8] Apache Software Foundation, "Apache HTTP Server Project," http://httpd.apache.org/.

[9] D. Mosberger and T. Jin, "httperf: A Tool for Measuring Web Server Performance," *Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[10] M. Swift, B. Bershad, and H. Levy, "Improving the Reliability of Commodity Operating Systems," in *Proc. Symp. Operating Systems Principles*, 2003, pp. 207–222.

[11] 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*, 1996, pp. 74–83.

[12] W. Ng and P. Chen, "The Design and Verification of the Rio File Cache," *IEEE Trans. Computers*, vol. 50, no. 4, pp. 322–337, 2001.

[13] M. Baker and M. Sullivan, "The Recovery Box: Using Fast Recovery to Provide High Availability in the UNIX Environment," in *Proc. Summer USENIX Conf.*, 1992, pp. 31–44.

[14] K. Kourai and S. Chiba, "A Fast Rejuvenation Technique for Server Consolidation with Virtual Machines," in *Proc. Int'l Conf. Dependable Systems and Networks*, 2007, pp. 245–254.