2011 IEEE 17th International Conference on Parallel and Distributed Systems

Extending Lifetime and Reducing Garbage Collection Overhead of Solid State Disks with Virtual Machine Aware Journaling

Ting-Chang Huang
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, ROC
tchuang@cs.nctu.edu.tw

Da-Wei Chang
Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan, ROC
dwchang@mail.ncku.edu.tw

***Abstract*—Virtualization is becoming widely deployed in commercial servers. In our previous study, we proposed Virtual Machine Aware journaling (VMA journaling), a file system journaling approach for virtual server environments. With reliable VMM and hardware subsystems, VMA journaling eliminates journal writes while ensuring file system consistency and data integrity in virtual server environments, allowing it to be an effective alternative to traditional journaling approaches in these environments.**

In recent years, solid-state disks (SSDs) have shown their potential as a replacement of traditional magnetic disks in commercial servers. In this paper, we demonstrate the benefits of VMA journaling on SSDs. We compare the performance of VMA journaling and three traditional journaling approaches (i.e., the three journaling modes of the ext3 journaling file system) in terms of lifetime and garbage collection overhead, which are two key performance metrics for SSDs. Since a Flash Translation Layer (FTL) is used in an SSD to emulate traditional disk interface and the SSD performance highly depends on the FTL being used, three state-of-the-art FTLs (i.e., *FAST*, *SuperBlock* FTL and *DFTL*) are implemented for performance evaluation.

The performance results show that, traditional full data journaling could reduce the lifetime of the SSD significantly. VMA journaling extends the SSD lifetime under full data journaling by up to 86.5%. Moreover, GC overhead is reduced by up to 80.6% when compared to the full data journaling approach of ext3. Finally, VMA journaling is effective under all the FTLs. These results demonstrate that VMA journaling is effective in SSD-based virtual server environments.

Keywords- journaling file system, flash translation layer, virtual machine

I. INTRODUCTION

In recent years, NAND flash-based solid-state disks (SSD) have gained in popularity in desktop/laptop computers and servers. SSDs show their potential to replace traditional magnetic disks in commercial servers due to their low access time, low power consumption, and shock-resistance capability.

Nevertheless, there are still two main obstacles for SSDs to be widely deployed. The first one is the reliability concern. The lifetime of an SSD is limited by the limited program/erase (PE) cycles of the flash memory used in an SSD. Generally, an SLC (single level cell) NAND flash block can sustain about 100K PE cycles, and an MLC (multi level cell) NAND flash block can sustain only 10K PE cycles. Once the limit of the PE cycles is reached, a block can no longer be updated. Such a problem could be serious in reliability-concerned server environment.

The second one is performance degradation resulted from garbage collection (GC) of an SSD. Modern SSDs adopt the *out-of-place update* approach, which writes the updated data to free pages1 and marks the out-of-date data as garbage. When the number of free pages drops below a specified threshold, the GC procedure is triggered to reclaim free space. The GC procedure selects a victim block, copies out the up-to-date data in the block, and then erases the block so that a block of free pages can be obtained. The overhead of GC involves data copying and block erasure, which is time-consuming and could have performance impact to the normal user requests. Therefore, GC overhead should be minimized in order to minimizing the performance impact to server workloads.

File systems, which generate I/O traffic to the storages, play an important role on the storage performance. Many modern file systems, for example, ext3 and NTFS, adopt journaling to maintain file system consistency and data integrity. However, file system journaling writes the update data (called journal data) to a preserved area on the storage (called journal area) before the data are flushed to the data area. Such additional journal writes increase the write traffic to the storage [1]. On an SSD, the increased write traffic leads to higher GC overhead and more block erasures, degrading the SSD performance and reducing the SSD lifetime.

In our previous study, we have proposed VMA journaling (Virtual Machine Aware journaling) [2], a new file system journaling approach used in virtual server environments, which are gaining in popularity in server platforms. VMA journaling eliminates journal writes while ensuring file system consistency and data integrity in virtual server environments. In recent years, virtual machines have been broadly used in server consolidation, disaster recovery, software testing, security and storage management. According to International Data Corporation (IDC), in 2008, there were more shipments of servers based on virtual machines than those based on

[1]A NAND flash memory chip is divided into a number of blocks and each block includes a fixed number of pages. Read/write operations are performed in units of a page, and erase operations are performed in units of a block.

1521-9097/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPADS.2011.61

physical machines, and the ratio will reach 1.5 in 2013 [3]. In a virtual server environment with reliable VMM and hardware subsystems, VMA journaling provides superior file system performance with similar level of data integrity and consistency when compared with traditional full data journaling. This is achieved by allowing cooperation between the journaling file systems and the Virtual Machine Monitor (VMM), which is a thin software layer running underneath the virtual machines and emulating the hardware interface to the guest operating systems (i.e. the operating systems running in the virtual machines). Unlike traditional journaling approaches, which write journal data to the on-storage journal area, VMA journaling retains journal data in the memory of the virtual machine and stores the information for locating these data (called the journal information) in the VMM. As a consequence, journal writes are eliminated.

We have evaluated the performance of VMA journaling on magnetic disks [2]. In this paper, we evaluate the performance of VMA journaling on reducing the GC overhead and extending the lifetime of an SSD. Due to the elimination of journal writes, VMA journaling reduces the write traffic to the SSD compared to the other journaling approaches, leading to lower GC overhead and a reduced number of block erasures.

For performance evaluation, we collected I/O traces of VMA journaling and three journaling approaches of ext3 under the execution of eight concurrent virtual machines. The collected traces were then fed into our SSD simulator for performance evaluation. Since a Flash Translation Layer (FTL) is used in an SSD to emulate traditional disk interface and the SSD performance varies depending on the FTL, three state-of-the-art FTLs: *FAST* [4], *SuperBlock* FTL [5], and *DFTL* [6] were implemented in the simulator and used in performance evaluation.

According to the simulation results, VMA journaling reduces the number of erase operations by up to 86.5%, while providing similar level of data integrity, when compared to traditional full data journaling of ext3. This indicates that VMA journaling can extend the SSD lifetime by 86.5% if the blocks of the SSD can be worn evenly. Compared to metadata journaling of ext3, which does not guarantee data integrity, VMA journaling has a similar number of erase operations. Moreover, 13.1% to 75.9% reduction in GC overhead is achieved when compared to traditional full data journaling of ext3.

The remainder of this paper is organized as follows. Section 2 describes the related work, followed by the design and implementation of VMA journaling in Section 3. Section 4 shows the performance results. Finally, conclusions are given in Sections 5.

II. Background and Related Work

A. Flash Translation Layer

Flash Translation Layers (FTLs) employed in SSDs emulate traditional disk interface to support disk-based file systems. An FTL maintains a mapping structure in the internal RAM of the SSD, which translates logical sector numbers to flash page or block numbers. The state of each flash page is maintained by the FTL. Once an erased (i.e., *free*) page is written with user data, it becomes a *live* page, and the page containing the old data is marked as *dead*. A time-consuming garbage collection (GC) procedure is invoked to maintain enough free pages for accommodating further writes. The procedure selects victim blocks, copies live pages to free pages and finally erases these victim blocks. The GC procedure, which involves data copying and block erasure, incurs performance overhead and could have an impact on the flash lifetime. Therefore, GC overhead is a key performance metric for an SSD.

Several FTLs have been proposed in the past few years, which can be classified into page-mapped, block-mapped, and hybrid-mapped FTLs according to their mapping granularities. Page-mapped FTLs, for example the *SuperBlock* FTL and the *DFTL* used in this paper, adopt a fine-grained translation method that directly translates each logical page to a physical page. Generally, GC overhead of page-mapped FTLs is usually smaller than the other types of FTLs [5][6]. However, due to fine-grained translation, a page-mapped FTL requires a large mapping table for a large-sized flash memory.

The *Superblock* FTL groups several contiguous logical blocks into a *superblock*. Pages belonging to a *superblock* are written to the physical blocks allocated for that *superblock*. Page-level mapping is used in a *superblock*. The page-level mapping information of *superblock* is stored in the spare area of the flash memory, instead of the internal RAM of the SSD. Therefore, the memory usage can be reduced. The GC procedure is invoked when the free pages of the target *superblock* is not enough to accommodate the incoming write. *DFTL* stores the page-level mapping information in the flash memory and caches the most recently used mapping information in RAM so as to reducing the RAM requirement of the mapping table. In *DFTL*, GC is triggered when the free pages of the flash storage drops below a threshold.

Block-mapped FTLs reduce the RAM requirement of the mapping table by directly reducing the number of entries in the mapping table. This is achieved by using a coarse-grained translation method, which translates each logical block number to a specific physical block number. In these FTLs, each logical page number is divided by the number of pages per block to obtain the logical block number (i.e., the quotient) and the page offset (i.e., the remainder). The former is used to index the mapping table to obtain the physical block number and the latter is used to locate the target page within the physical block. Such mapping approach leads to high GC overhead due to the limitation that each logical page can only be written to a fixed offset of a physical block. For example, frequently

updating a logical page would lead to frequent block erasure.

Several hybrid-mapped FTLs such as the *FAST* FTL [4] have been proposed to achieve low GC overhead while keeping the size of the mapping table small. *FAST* divides the blocks into two types: data blocks and log blocks. The former, which utilizes the coarse-grained mapping scheme, is used to satisfy the first-time write of each logical page while the latter, which utilizes the fine-grained mapping scheme, is used to accommodate page updates.

B. Journaling File System

Journaling file systems adopt the concept of write-ahead logging to maintain file system consistency. In-memory file system updates are grouped into a transaction and then committed into the journal area, which is a reserved space on the storage, before they are flushed to the data area of the storage. When a system crashes, the file system can be efficiently brought back to a consistent state by replaying the journal data (i.e. data in the journal area).

Many journaling file systems support more than one journaling modes with different performance and consistency strengths. For example, both ext3 and ReiserFS support three journaling modes: *writeback*, *ordered* and *journal* modes. The differences among these modes are the content in the journal area and the flush order of the data. Both writeback and ordered modes log only metadata and therefore do not ensure data integrity. In the writeback mode, data are exposed to the buffer flushing threads of the operating system immediately after each file operation, and metadata are exposed after each journal commit. Since no flushing order is enforced in the writeback mode, metadata may be flushed back to the data storage before the associated data, causing the dangling pointer problem. Therefore, this mode is generally regarded as having the weakest consistency semantic among the three modes. The ordered mode solves the dangling pointer problem by ensuring that the updated data are flushed back to the storage before the commit of the associated metadata. Although the strict flushing order provides strong consistency semantic, the ordered mode does not guarantee data integrity since the data updates are not journaled, as in the writeback mode. The journal mode supports full data journaling, that is, it logs both data and metadata and thus guarantees both file system consistency and data integrity. All the data and metadata updates are committed to the journal area before they are exposed. However, this mode has inferior performance under most workloads since all the data and metadata updates have to be written to the storage twice.

C. Virtual Machine

Virtualization technology allows multiple isolated virtual machines to run concurrently in one physical machine, sharing the resources of that physical machine. Each virtual machine is supported by the VMM, a thin software layer running underneath the virtual machine. A virtual machine is also called a *guest domain*, and the operating system running in a virtual machine is called a *guest operating system*. Generally, VMM can run on an operating system or a bare hardware. In this paper, we focus on the VMM running directly on bare hardware, which provides better performance and is commonly used in server environments.

VMM emulates the hardware interface and provides virtual computing resources, such as virtual CPU and disk, to the guest operating systems. Virtual disks are usually implemented in the form of partitions or image files on the physical storage. The guest operating systems can utilize their preferred file systems to manage the virtual disks.

D. Reducing Write Traffic to SSD

In addition to VMA journaling, several techniques have also been proposed to extend SSD lifetime by reducing the write traffic to an SSD. Griffin [7] employs a disk as the write cache of the SSD. The data are first written to the log-structured disk cache, and then migrated to the SSD periodically. Since duplicated data writes can be merged in the disk cache, the write traffic to the SSD is reduced. The CAFTL [8] uses the concept of Content-Addressable Storage (CAS) in the FTL layer to reduce write traffic to the SSD. Multiple blocks with the same content can be stored as a single copy in the SSD, reducing the write traffic to the SSD.

III. Overview of VMA Journaling

In order to eliminate journaling writes to the storage, VMA journaling commits journal information to the VMM instead of the storage. Figure 1 illustrates the difference between the handling of journal data in the traditional and VMA journaling approaches. In step 1, both traditional and VMA journaling group dirty buffers, which reflect metadata and data updates, into a transaction. Then, the traditional journaling approach commits these dirty buffers to the on-storage journal area, as shown in step 2 of Figure 1(a). However, VMA journaling commits the dirty buffers to the journal area residing in the VMM memory instead of the storage, as shown in step 2 of Figure 1(b). Therefore, no journaling writes to the storage are needed. After the commit, the guest domain can flush the committed dirty buffers in an asynchronous manner, as shown in step 3 of Figure 1(a) and (b). When a domain crashes, the information in the journal area can be used for file system recovery.

Committing dirty buffers to the in-VMM journal area does not cause the buffer data to be copied from the guest domain memory to the VMM. VMA journaling uses a single copy of buffer to represent both the updated data

and the journal data. Therefore, committing a buffer under VMA journaling involves transferring only the information for locating the buffers to the VMM. Specifically, only the journal information, i.e. the tuple (memory address, block number) denoting the machine memory address and the corresponding disk block number of the buffer, is transferred to the VMM. Once a domain crashes, VMA journaling allows the VMM to retain the memory of the committed dirty buffers, and it flushes the retained buffers back to the storage for maintaining file system consistency and data integrity.

Since the journal data are placed in the guest domain memory, unauthorized modifications to these data could disrupt the file system consistency and data integrity. For example, unintentional wild writes in the guest domain kernel might corrupt the journal data, leading the file system to an unrecoverable state. To prevent wild writes from modifying the journal data, VMA journaling write-protects the memory pages of the to-be-committed dirty buffers when committing a transaction. Therefore, the guest operating system can detect and stop wild writes to the journal data via page faults.

Unlike a wild write, a write issued from the file system to a protected buffer should be allowed since the buffer still represents the updated data. Copy-on-write (COW) is used for such a write. That is, the content of the protected buffer is copied to a free and unprotected buffer, which is used for satisfying the write. On the next commit, these two buffer copies are merged by write-protecting the buffer containing the most up-to-date data and freeing the original protected buffer that contains the stale data. After the merge, a single copy of buffer again represents both the updated data and the journal data.

Reclamation of the journal data is triggered when the buffer is going to be reclaimed by the guest operating system, for instance, under memory pressure. When the buffer is going to be reclaimed, VMA journaling unprotects the buffer and removes the journal information corresponding to the buffer via a hypercall (i.e., a call to the VMM). After that, the buffer no longer represents the journal data and can be reclaimed by the guest operating system. Note that, file system consistency and data integrity still remain since a buffer is reclaimed only after its content has been flushed to the data area. However, unprotecting each buffer during its reclamation could cause a significant overhead due to frequent hypercalls, which involves switches between the VMM and guest domains. To reduce the overhead, batch unprotection is adopted.

Reclamation of the journal data and information also takes place when the VMM cannot afford the journal area of the guest domain. When this situation occurs, the checkpoint procedure is triggered, during which the journal data are flushed to the data area until the size of the journal area has been reduced to below a specific

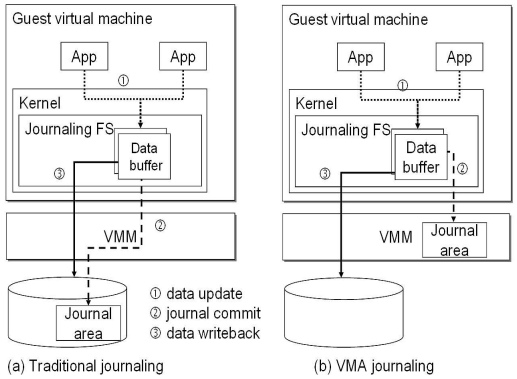

Figure 1. (a) Traditional journaling and (b) VMA journaling.

threshold. Nevertheless, since only metadata (i.e., the journal information) are stored in the journal area, the required size of a journal area is quite small in VMA journaling and thus this situation rarely occurs.

Similar to traditional journaling approaches, VMA journaling performs recovery by simply replaying the journal data. In VMA journaling, this is achieved through the cooperation of the VMM and the system management domain (e.g. domain 0 in the Xen virtualization environment). When a guest domain crashes, the VMM reclaims all memory pages of the domain except those containing the journal data (i.e. the protected buffers). Then, the VMM notifies the system management domain, which wakes up a recovery thread to start the following recovery procedure. First, for each file system mounted as VMA journaling in the crashed domain, the recovery procedure issues a query for the total size of the journal data. If the size is zero, the corresponding file system is consistent and the recovery thread goes on to check the next file system. Otherwise, the recovery thread prepares a free memory pool of that size for exchanging with pages containing these journal data. It then issues a hypercall to perform the memory exchange to obtain the journal data and to retrieve the corresponding journal information from the VMM. After the exchange has been completed, the recovery thread writes the journal data back to the data area according to the journal information. The memory exchange is implemented by page remapping. The journal data are remapped into the system management domain, and the pages in the free memory pool are remapped into the VMM. After the journal data have been written, the memory containing the journal data becomes free memory of the system management domain, and the recovery thread informs the VMM to reclaim the journal information about the file system.

TABLE I. ENVIRONMENT FOR TRACE COLLECTION

Hardware	**CPU**	Pentium 4 - 3.2 GHz
	Memory	DDRII 2 GB
VMM	Xen 2.0.7	
Domains	**Kernel**	XenoLinux 2.6.11
	Memory	128Mbytes for each guest domain 256Mbytes for domain 0
	Virtual disks	Virtual system disk: 8 GB Virtual data disk: 4 GB
Micro Benchmarks	Filebench: *seq_write* and *rnd_write*	
Macro Benchmarks	*Postmark* , *untar* and *kernel_compile*	

TABLE II. CONFIGURATION OF THE SSD SIMULATOR

Parameters	Values
Number of blocks	262144 (32GB)
Pages per block	64
Page size	2KB
Page read/write time	88 us / 263us
Block erase time	2000us
FTLs	*FAST*, *SuperBlock*, *DFTL*

IV. EVALUATION OF VMA JOURNALING ON SSDS

We have implemented VMA journaling as a new journaling mode of ext3 (called the VMA mode) on the Xen virtual machine environment. In this section, we compare VMA journaling with traditional journaling approaches (specifically, the original three journaling modes of ext3) in terms of erase counts (i.e., the numbers of erase operations) and GC overhead. Note that, the writeback and ordered modes of ext3 use the metadata journaling approach while the journal mode and VMA mode use the full data journaling approach.

The performance evaluation is performed by first collecting I/O traces under different journaling modes and then feeding the collected I/O traces to an SSD simulator to measure the erase counts and the GC overhead.

Table 1 shows the machine configuration and the workloads used for trace collection. Eight guest domains were run on a physical machine, with 128MB of memory allocated for each domain. In order to prevent system I/O activities from affecting the performance results, two virtual disks belonging to two separated physical disks were allocated for each domain. The virtual system disks stored the program images including the operating systems, libraries and benchmarks, while the virtual data disks stored the data accessed by the benchmarks. Each virtual disk was made up of a single partition on the corresponding physical disk.

Two micro-benchmarks (*seq_write* and *rnd_write*) and three macro-benchmarks (*postmark*, *untar* and *kernel_compile*) were used as the workloads for trace collection. In the seq_write benchmark, a single empty file is first created and then has 8 kB of data appended each time until the file size reaches 600MB. The rnd_write benchmark issues a sequence of 4 kB random writes to a 512MB file until 128MB of data have been written. The *postmark* [9] simulates the workload of a news or email server. During execution, it creates an initial set of small files and then applies a number of transactions on those files. Each transaction consists of a create/delete operation together with a read/append operation. The *untar* benchmark extracts a Linux source tree (version 2.6.11) from a bzip2-compressed image. Finally, the *kernel_compile* benchmark is a CPU-intensive workload, which builds a compressed kernel image from the source tree of the Linux kernel.

Eight concurrent guest domains were run during the trace collection, each of which executed an instance of the given workload. We modified the Xen backend disk driver running in domain 0 (i.e., the system management domain) to log the I/O requests during trace collection. Note, only the I/O requests of the virtual data disks were logged.

We have implemented an SSD simulator to report the erase counts and the GC overhead of a given I/O trace. Table 2 shows the configuration of the SSD simulator. A 32GB flash storage (i.e., 262,144 blocks) was simulated. All the time-related values were obtained from the specification of the Samsung K9K4G08U0M flash memory chip [10]. Three FTLs were implemented in the simulator: *FAST*, *Superblock* and *DFTL*. Note that, real SSDs were not used for performance evaluation since it is difficult to retrieve the internal information of real SSDs (e.g. the erase count, GC overhead, and FTL).

In the following, we first show the reduced write traffic of the VMA mode compared to the journal mode of ext3. Then, the results of lifetime extension and GC overhead reduction resulting from the write traffic reduction are shown.

A. Write Traffic Reduction

Figures 2 and 3 show the numbers of I/O requests and the amount of I/O traffic under different journaling modes, respectively. As seen, when compared to the journal mode of ext3, the VMA mode reduces the numbers of I/O requests (i.e., I/O counts) by 9.4% to 48.9% and the amount of I/O traffic by 42.4% to 56.7%, showing the effectiveness of journal write elimination. Compared to the metadata journaling modes, which do not ensure data integrity, the VMA mode presents a similar amount of I/O traffic.

Comparing the two metadata journaling modes, the ordered mode leads to more I/O requests and traffic than the writeback mode since, as mentioned in Section 2.2, the ordered mode needs to flush the updated data before committing the associated metadata. Such strict ordering limits the effect of delayed write, leading to more I/O traffic.

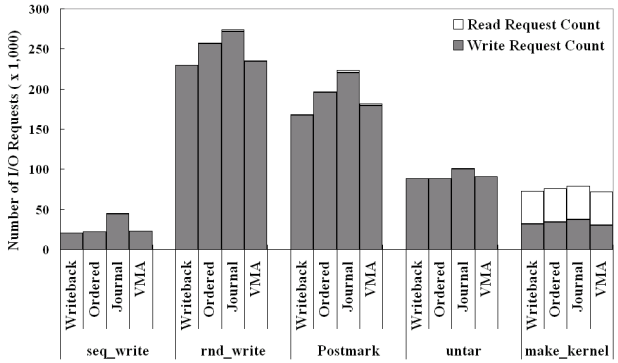

Figure 2. Number of I/O Requests under Different Journaling Modes

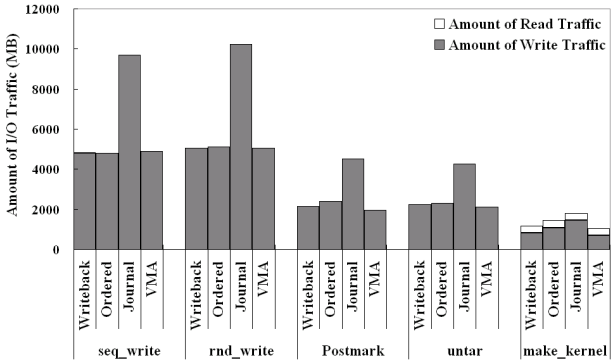

Figure 3. Amount of I/O Traffic under Different Journaling Modes

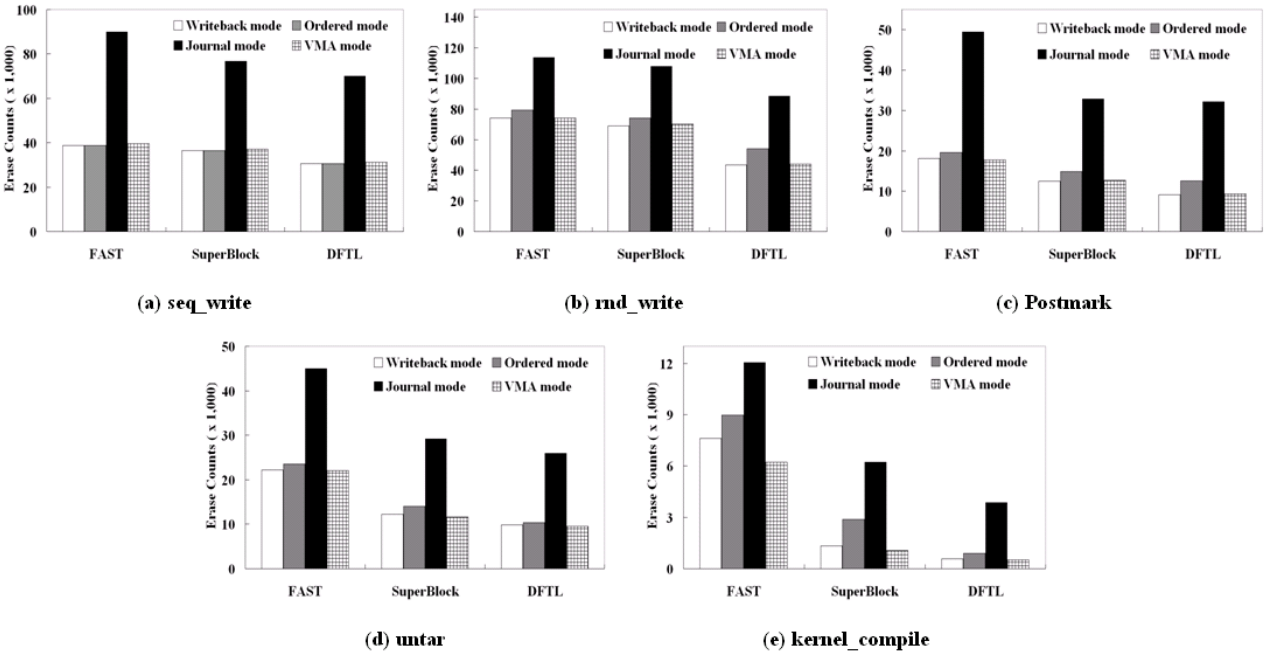

Figure 4. Erase Counts

B. Lifetime Extension

Figure 4 shows the erase counts of different journaling modes. From the figure, the journal mode has the largest erase counts than the other modes. For example, compared to the journal mode, the writeback and ordered modes result in only 37.8% and 40% of the erase counts, respectively, when *DFTL* is used under the execution of the *Untar* benchmark. The large erase count of the journal mode is due to the large write traffic of that mode resulting from logging the updates of both metadata and data.

As mentioned before, the lifetime of an SSD is limited by the limited PE cycles of the flash memory. Given that erase operations can be distributed evenly among flash blocks through the use of wear-leveling techniques, the lifetime of an SSD is proportional to the number of erase operations the SSD endures.

Therefore, from the results of *DFTL* under the *Untar* benchmark, the journal mode reduces the SSD lifetime by more than 60% when compared to the metadata journaling modes. Such significant lifetime reduction would be an obstacle for full data journaling to be widely employed in reliability-concerned server environment.

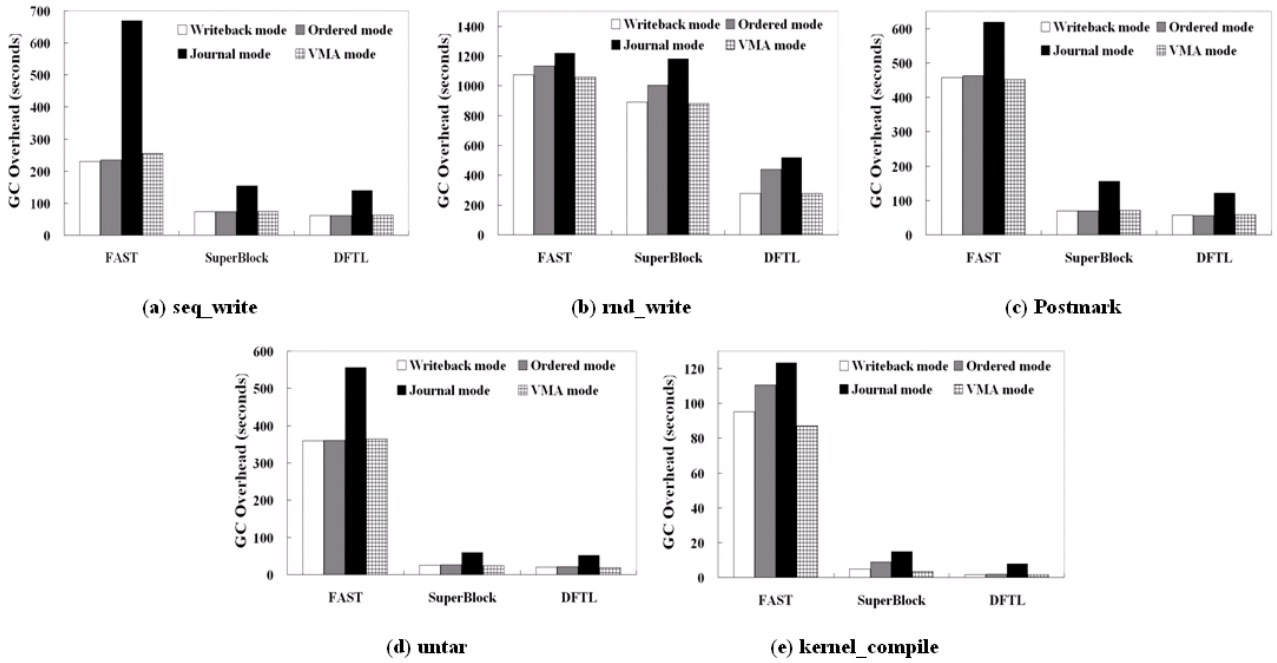

Figure 5. Garbage Collection Overhead

VMA mode also supports full data journaling since it also logs updates of both metadata and data, ensuring both file system consistency and data integrity. However, due to journal write elimination, VMA mode reduces the erase counts by 34.6% to 56% in the micro benchmarks and 48.4% to 86.5% in the macro benchmarks, compared to the journal mode. Furthermore, the VMA mode shows similar or even lower erase counts than the metadata journaling modes. Therefore, VMA mode extends the SSD lifetime by 86.5% when compared to journal mode, and it achieves similar SSD lifetime when compared to metadata journaling modes, which do not guarantee data integrity.

Note that, VMA journaling is effective for all the three FTLs. Up to 64%, 83% and 86.5% erase operations are eliminated in *FAST*, *Superblock* and *DFTL*, respectively, compared to the journal mode. Note, the erase counts of the *SuperBlock* FTL and *DFTL* are less than those of the *FAST* FTL. Such result is consistent with previous studies [5-6].

C. GC Overhead Reduction

As mentioned above, garbage collection could lead to unpredictable performance drop of the SSD. Figure 5 shows the garbage collection overhead of VMA mode and the other three journaling modes of ext3 under *FAST*, *SuperBlock* and *DFTL*. Not surprisingly, compared to the journal mode, the VMA mode reduces the GC overhead by 13.1% to 63.2% in micro benchmarks and by 27% to 80.6% in macro benchmarks, and it has similar GC overhead when compared with the metadata journaling modes. For each FTL, up to 63.2%, 75.9%, and 80.6% GC overhead are reduced in *FAST*, *Superblock* and *DFTL*, respectively, compared to the journal mode.

CONCLUSIONS

Solid state disks (SSDs) have gained in popularity and show their potential to replace traditional magnetic disks in commercial servers. In our previous study, we proposed Virtual Machine Aware journaling (VMA journaling), a new file system journaling approaches used in virtual server environments which have been broadly used. In this paper, we demonstrate the following. First, traditional full data journaling, which ensures both file system consistency and data integrity, could reduce the lifetime of the SSD significantly. Such significant lifetime reduction is an obstacle for full data journaling to be widely employed in reliability-concerned server environment. Second, VMA journaling, which also ensures both file system consistency and data integrity, effectively extends the lifetime of the SSD (by up to 86.5% compared to the traditional full data journaling approach) in virtual server environments. Moreover, the GC overhead is reduced by up to 80.6% when VMA journaling is used. Third, three state-of-the-art FTLs (i.e., *FAST*, *SuperBlock* FTL and *DFTL*) are used in the simulated SSD, and the results show that VMA journaling is effective under all these FTLs. From these results, we show that VMA journaling is an effective alternative to

traditional journaling approaches in SSD-based virtual server environments.

REFERENCES

[1] Prabhakaran, V., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H., "Analysis and Evolution of Journaling File Systems", in Proceedings the USENIX Annual Technical Conference (USENIX '05), Anaheim, CA, U.S.A., April 13-15 2005.

[2] Huang, T. C., and Chang, D. W., "VM Aware Journaling: Improving Journaling File System Performance in Virtualization Environments", Software: Practice and Experience, in press.

[3] Bateman K., "IDC Charts Rise of Virtual Machines", Available at: http://www.channelweb.co.uk/crn/news/2242378/virtual-machines-exceeding, May 2009.

[4] Lee, S. W., Park, D. J., Chung, T. S., Lee, D. H., Park, S., and Song, H. J., "A Log Buffer-based Flash Translation Layer Using Fully-associative Sector Translation", IEEE Transactions on Embedded Computing Systems, vol. 6, no. 3, pp. 18-18, July 2007.

[5] Kang, J. U., Jo, H., Kim, J. S., and Lee, J., "A Superblock-based Flash Translation Layer for NAND Flash Memory", in Proceedings 6th International Conference on Embedded Software (EMSOFT '06), Seoul, Republic of Korea, October 22-27 2006, pp. 161-170.

[6] Gupta, A., Kim, Y., and Urgaonkar, B., "DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings", in Proceedings 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '09), Washington, DC, U.S.A., March 7-11 2009, pp. 20-20.

[7] Soundararajan, G., Prabhakaran, V., Balakrishnan, M., and Wobber, T., "Extending SSD Lifetimes with Disk-based Write Caches", in Proceedings 8th USENIX Conference on File and Storage Technologies (FAST '10) , San Jose, CA, U.S.A., February 23–26 2010, pp. 8-8.

[8] Chen, F., Luo, T., and Zhang, X., "CAFTL: a Content-aware Flash Translation Layer Enhancing the Lifespan of Flash Memory Based Solid State Drives", in Proceedings 9th USENIX Conference on File and Storage Technologies (FAST '11) , San Jose, CA, U.S.A., February 15–17 2011, pp. 6-6.

[9] Katcher, J., "PostMark: A New File System Benchmark", Available at: http://communities-staging.netapp.com/servlet/JiveServlet/download/2609-1551/Katcher97-postmark-netapp-tr3022.pdf, August 1997.

[10] Samsung Electronics, 512M x 8 Bit / 256M x 16 Bit NAND Flash Memory, Datasheet.