NAND Flash Storage Device Performance in Linux File System

Yuanting Wei
School of ICE
Sungkyunkwan University
Suwon, Korea
Email: weiyuantin@skku.edu

Dongkun Shin
School of ICE
Sungkyunkwan University
Suwon, Korea
Email: dongkun@skku.edu

***Abstract*- To enhance reliability of the Linux file system, a new technique for disk storage management called a log-structured file system for the Sprite operating system was presented. A log-structured file system writes all modifications to disk sequentially in a log-like structure, thereby speeding up both file writing and crash recovery. Presently, NILFS, BTRFS and Ext4 are the most striking Linux file systems; each of them has its own characteristics, represents a different design and development of Linux file system. This paper measures the performance of NILFS2 comparing with EXT3 and EXT4 file systems by using SSDs in Linux file system.**

I. INTRODUCTION

The Log-structured File System (LFS) is a little different than other file systems with both advantages and disadvantages [1]. Rather than write to a tree structure such as a b-tree or an h-tree, either with or without a journal, a log-structured file system writes all data and metadata sequentially in a continuous stream that is called a log, no blocks are overwritten, and log-like structures are appended to the disk instead [2]. A log consists of a series of segments, where each segment contains both data and inode blocks. The motivation behind log-structured file system is that typical file systems lay out data based on spatial locality for rotating media (hard drives). But rotating media tends to have slow seek times limiting write performance.

A log-structured file system, because of its design, makes it very easy to create snapshots (in NILFS they are called checkpoints) of both the data and metadata. NILFS can then mount these checkpoints (or snapshots) alongside the primary NILFS file system. From these checkpoints, you can recover erased files (if the checkpoint has a date and time prior to when the file was erased) or you can use it for backups or even disaster recovery images. Another benefit of log-structured file systems is that recovering from a crash is easier than the more typical tree based file system (e.g. ext2, ext3, etc.). After a log-structured file system crashes, when it is remounted it can reconstruct its state from the last consistent point in the log. It starts at the head of the circular log and backs up until the file system is consistent. This point should be very close to the head so little if any data or metadata will be lost. This process is extremely fast regardless of the size of the file system [3].

A log-structured file system recovers from a crash extremely fast and the amount of time is independent of the size of the file system. In contrast, other file systems have to replay their journal and possibly even walk their data structures to make sure the file system is consistent. As we know, it is a so huge job to take how much time when it has run fsck (file system check) on a very large file system.
Because disk capacity is limited, a Garbage Collection (GC) is needed to collect deleted file blocks and logically overwritten blocks. Garbage collection is a major overhead of LFS. However, the garbage collector can efficiently restore fragmented file blocks. For efficient garbage collection, whole disk is divided into fixed sizes (ex. 4 mega bytes). This management unit is called a full segment. Writing out is done sequentially in full segments [4].

II. NILFS IMPLEMENTATION

The Nippon Telephone and Telegraph (NTT) CyberSpace Laboratories has been developing NILFS for Linux. It is released under the GPL 2.0 license and is included in the 2.6.30 kernel. It spent a great deal of time in the -mm kernels and underwent much testing since its initial announcement.

NILFS is a log-structured file system supporting continuous snapshotting. In addition to versioning capability of the entire file system, users can even restore files mistakenly overwritten or destroyed just a few seconds ago. Since NILFS2 can keep consistency like conventional LFS, it achieves quick recovery after system crashes. NILFS2 creates a number of checkpoints every few seconds or per synchronous write basis (unless there is no change). Users can select significant versions among continuously created checkpoints, and can change them into snapshots which will be preserved until they are changed back to checkpoints.

In NILFS, our design goals are to obtain high reliability and availability of the file system. We have not yet begun performance tuning. However, to be able to use the NILFS in the future, the file size and inode numbers are stored in 64-bit-wide fields, and file blocks are managed by a B-tree [5][6]. The root of the file block B-tree is placed on the inode structure. The inode is managed by the inode block B-tree, the root of the inode block B-tree is stored in the superblock structure of the file system.

The disk layout of NILFS is shown in Figure 1, divided into several parts [4].

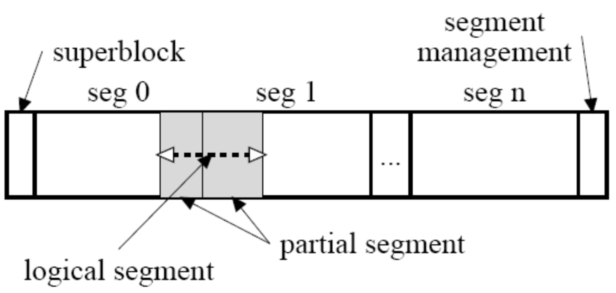

Fig. 1. Disk Layout of the NILFS

A. *Continuously Snapshot*

NILFS is a new implementation of a log-structured file system (LFS) supporting continuous snapshotting. The current major version of NILFS is version 2, which is referred to as NILFS2. NILFS2 realized online garbage collection that reclaims disk space with keeping multiple snapshots.

Continuous snapshot is the most attractive feature of NILFS2. It allows NILFS2 users to restore files mistakenly overwritten or destroyed just a few seconds ago. Since NILFS can keep consistency like conventional LFS, it achieves quick recovery after system crashes.

Some other file system also support snapshot, but it often requires human intervention, users must use the FS command to create the snapshot comes. However, misuse is often unpredictable, it is impossible to create a snapshot just before the mistake. Therefore, other file systems need very professional staff and tools to recovery files. NILFS2 users are more fortunate, because the system can automatically backup all file operations. Therefore, NILFS2 can not only recovery the deleted files timely, but also can restore the file contents before any changes. Furthermore, in NILFS2, users no longer need a special version of the file management tools to manage different versions. And all of this is automatic.

For system administrators, NILFS2 uninterrupted snapshot feature allows Online backup and other daily operations more convenient, do not need to learn complex backup and recovery commands, and can finally from the daily affairs of these complex freed.

However, when the users use all checkpoints as the snapshot, there is no disk space for garbage collection. The user can select any checkpoints as a snapshot, and the garbage collector collects other checkpoint blocks. The user does not need any commands “before” taking a snapshot [4].

B. *Efficient Crash Recovery*

For a long time, one of the most concerned issues for file system designers is to minimize system checks of the file system after a crash and recovery time. No matter what kind of file system, when hardware crashes occur, the file system is very likely in an inconsistent state. So after reboot, they need to run fsck.

Ext3 and many other Linux file systems use logging technology to reduce fsck time. NILFS2 is a log-structured file system, this is why the fsck time is shorter, and no matter how big the disk is, how many the files are, the fsck time of NILFS2 is certain.

Many studies have shown that the overall efficiency of the file system mainly by the efficiency of write operation. Because the efficiency of the read operation depends on cache design. In Linux, cache unified management by the VFS, thereby increasing the efficiency of write operations can improve the efficiency of the overall file system.

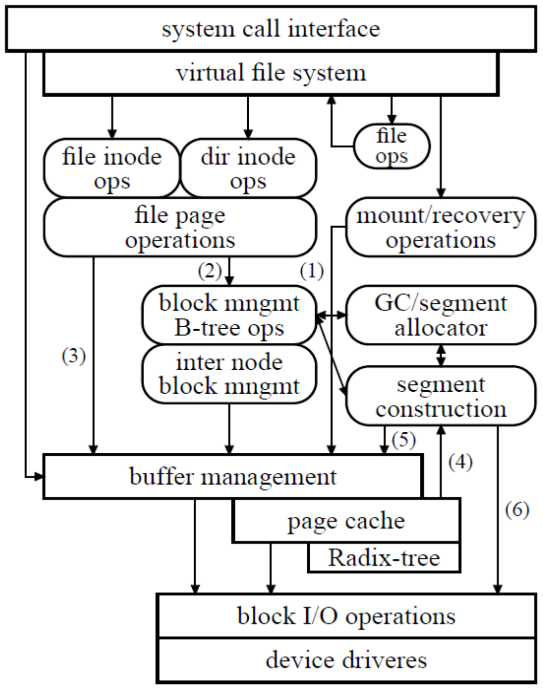

Fig. 2. Architecture of the NILFS

C. *NILFS Architecture*

Figure 2 shows the architecture of NILFS. Rounded box parts are implemented as NILFS.

Mount/recovery operations call a buffer management module (line (1) in Figure 2) of Linux Kernel 2.6 to read the superblock and segment summary that wrote last mounted time. File page operations use the NILFS’s block management module (2) to lookup/insert/delete appropriate disk blocks via the NILFS B-tree operations. Normal file read operations execute by the file page operations module using buffer management module directly (3). When amount of dirty pages are exceeded an internal limits, a segment construction module is triggered (4) to start a segment construction. The segment construction module calls the buffer management module (5) to arrange the dirty pages, and call block I/O operations (6) for writing out the constructed segments.

Linux Kernel parts (square box) are not modified to implement the NILFS [4].

D. Transaction Processing and Segment Construction

Many operations on files are formed by multiple sub-operations, each sub-operation only to modify a specific meta-data, only if all the sub-operations are completed, the file operation to be successful; any sub-operation failed, it should be rolled back to the file system previous state. This sub-operation is a transaction.

After the transaction is committed, the file system will be in a consistent state. As mentioned earlier, this is a checkpoint.

Create a checkpoint in NILFS2 terminology is called the segment construction. NILFS2 adopted a dedicated kernel thread to handle the segment construction work. The thread woke up at a definite time, if needed, will create a segment, to generate a checkpoint. This is the implementation of continuous snapshot in NILFS2. In addition, each time after committing the transaction, NILFS2 will wake up the background thread to create a checkpoint.

E. Garbage Collection

NILFS implements garbage collection in a unique way. Garbage collection (GC) in NILFS is executed by the user-mode process which is called "*cleanerd*". It uses a user-space daemon to perform the GC. This daemon is activated when the file system is mounted via the "mount" command. This also means that GC can be activated at any time (if the file system is mounted).

NILFS will delete checkpoints after a certain period of time unless the checkpoint is converted to a snapshot. The amount of time when the checkpoint is held before being deleted is controlled by parameters in the /etc/nilfs_cleanerd.conf file. You can adjust the garbage collection parameters in the file and restart the GC daemon so that the new parameter values are used (or unmounting and remounting the file system).

F. Differences between NILFS2 and Journal File System

Ext3 is a journaling file system while NILFS2 is a log-structured file system. Journal and log seems to be no difference in the dictionary, both can be translated into the log. In modern English, seems to be universal. On the contrary, log-structured file system and journaling file system are two different technologies. And the differences between two file systems are very simple:

1) Journal file system only stores metadata in the log, while log-structured file system uses logs recording all changes, including metadata and data.

2) Unlike random writes in the journal file system; there are only additional writes in the log-structured file system.

III. EXPERIMENT RESULTS

For the performance evaluation, we used Iozone [7] which is a well-known file system benchmark tool to measure the performance of SSDs in Linux 2.6.35. Iozone is useful for determining a broad file system analysis of a vendor's computer platform. We evaluated the performance using Iozone benchmark tool with several I/O sizes which are from 4Kbyte to 2Gbyte. Iozone executes sequential write, sequential rewrite, sequential read, random read and random write, respectively. And we adopted four kinds of SSDs which are Micron realSSD C300, Samsung 470 series, OCZ VERTEX2 and Intel X25-M, respectively, to evaluate the performance of NILFS2 comparing with EXT3 and EXT4.

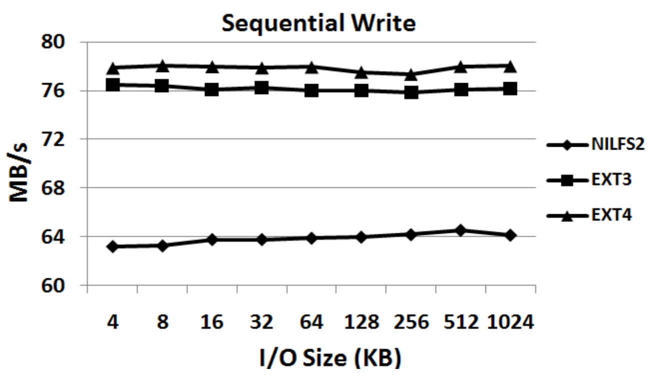

Fig. 3. Sequential Write (Micron realSSD C300)

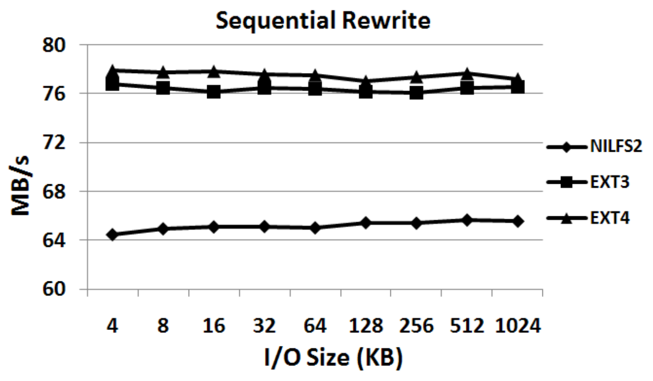

Fig. 4. Sequential Rewrite (Micron realSSD C300)

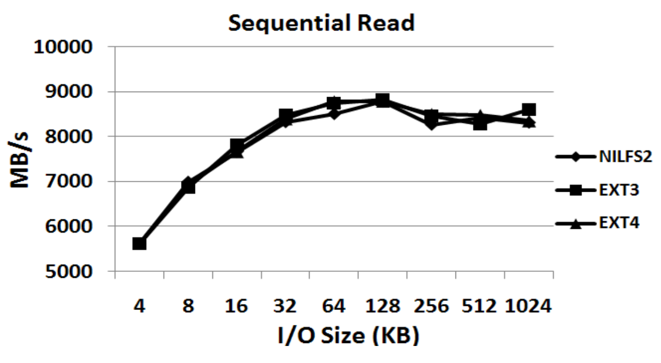

Fig. 5. Sequential Read (Micron realSSD C300)

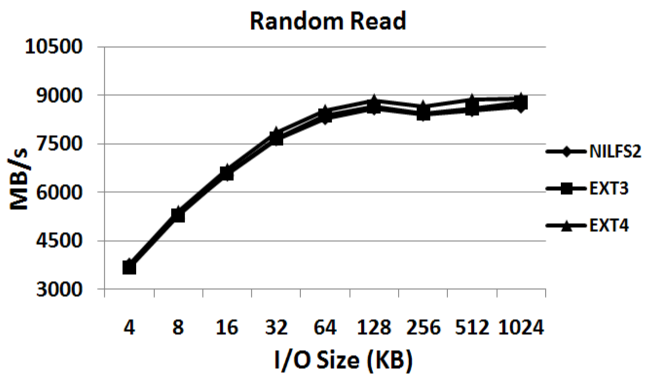

Fig. 6. Random Read (Micron realSSD C300)

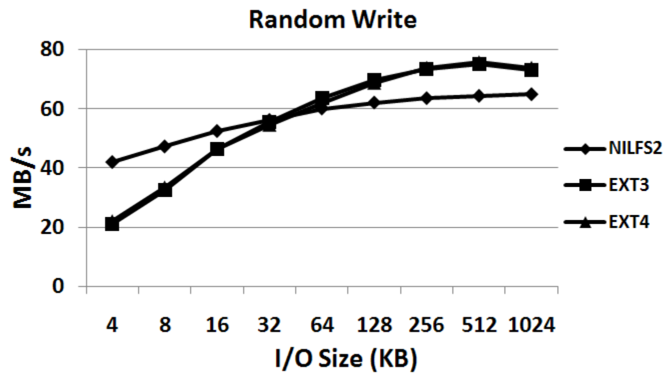

Fig. 7. Random Write (Micron realSSD C300)

In figure 3 and 4, NILFS2 shows poorer performance than both EXT3 and EXT4 in sequential write and sequential rewrite operations.

Figure 5 and figure 6 show similar results in sequential read and random read operations among all the file systems.

In figure 7, NILFS shows high performance than EXT3 and EXT4 when I/O size is small. This is because being a log-structured file system, there is no read-and-modify operation in NILFS. However, with the I/O size increases, EXT3 and EXT4 shows even more advantages for as a journal file system.

There are similar distributions in other three kinds of SSDs, thereby no longer instructions here.

IV. CONCLUSION

This paper described overview of the log-structured file system and a promising LFS supporting continuous snapshotting called NILFS. Snapshot and crash recovery features make NILFS a potential system administrators dream file system.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0010387).

REFERENCES

[1] Rosenblum, Mendel and Ousterhout, John K, “The LFS Storage Manager,” *Proceedings of the 1990 Summer Usenix*, pp.315-324, June 1990.

[2] Rosenblum, Mendel and Ousterhout, John K, “The Design and Implementation of a Log-Structured Filesystem”, *ACM Transactions on Computer Systems*, 10(1). pp.26-52, February 1992.

[3] Jeffrey B. Layton, “NILFS: A File System to Make SSDs Scream”, *Linux Magazine*, June 2009.

[4] Nilfs team, “the Nilfs version 1: overview”, *NTT Cyber Space Laboratories NTT Corporation*. http://www.osrg.net/nilfs/nilfs@osrg.net

[5] R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173-189,1972

[6] Douglas Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121-138,1979

[7] Iozone. http://www.iozone.org/.