Journal of the Korea Academia-Industrial cooperation Society
Vol. 20, No. 7 pp. 45-51, 2019

https://doi.org/10.5762/KAIS.2019.20.7.45
ISSN 1975-4701 / eISSN 2288-4688

RFJ: A Reliable and Fast Journaling Mechanism

Sejin Park
Department of Computer Engineering, Keimyung University

RFJ: 신뢰적 고성능 데이터 버퍼 저널링 기법

박세진
계명대학교 컴퓨터공학전공

Abstract Modern file systems have journaling mechanism to maintain their stored state consistently even under unexpected system crashes or disasters. However, the journaling makes I/O throughput lower. This performance degradation comes from the ordering mechanism between the data buffer and metadata buffer and two-staged buffer writing. Especially, if the data buffer and metadata buffer are journalled at the same time, then it incurs significant performance degradation due to the two-staged writing. That shows the trade-off relation-ship between I/O performance and system reliability. In this paper, we propose RFJ: a reliable and fast jour-naling mechanism to deal with this trade-off relationship. We propose an ordering enforced writeback journaling mode and selective journaling mechanism. The Ordering enforced writeback journaling mode achieves low I/O latency and the selective journaling mechanism achieves high reliability. The experimental result shows that the performance of RFJ is almost 5x faster than the journal mode of Ext3 file system but it still supports the same reliability with the journal mode.

요 약 현대 파일 시스템은 예기치 못한 시스템 크래시 또는 재난 상황에서도 데이터의 일관성 유지를 위해 저널링 메커니즘을 유지한다. 그러나 저널링은 I/O 처리율을 떨어뜨리는 문제가 있다. 이 성능 저하 문제는 데이터 버퍼와 메타데이터 버퍼간의 오더링 메커니즘과 2단계 버퍼쓰기에서 기인하는데. 특히, 만약 데이터 버퍼와 메타데이터 버퍼가 동시에 저널링이 되면, 2단계 쓰기 때문에 심각한 성능저하가 발생하며, 이는 I/O 성능과 시스템 신뢰도 간의 Trade-off 관계가 있음을 나타낸다. 본 논문은 RFJ 라는 신뢰성 있는 고속 저널링 기법을 제안한다. 이 기법은 Ordering enforced writeback 저널링 모드와 selective journaling 메커니즘을 도입해서 높은 신뢰도와 동시에 고성능 I/O 가 가능하게 한다. 본 논문에서 제안한 기법의 실험 결과 기존 Ext3 저널링 모드 대비 약 5배 이상 빠른 I/O 처리량을 지원하면서 동시에 Ext3 저널링과 동일한 수준의 신뢰성을 나타는 것을 확인 할 수 있었다.

Keywords : Journaling, Filesystem, Reliablility, Ordering, Data

1. Introduction

Unexpected crashes like power outage or hardware failure can make file system state inconsistently, which makes recovery harder. Journaling mechanism enables fast file system recovery even when the system is failed. When a new data is written to file system, a log for the write operation is committed in journal area before the data is stored to the file system. This

This research was supported by the Keimyung University Research Grant of 2018.
*Corresponding Author : Sejin Park(Keimyung Univ.)
email: baksejin@kmu.ac.kr
Received April 8, 2019 Revised May 15, 2019
Accepted July 5 ,2019 Published July 31,2019

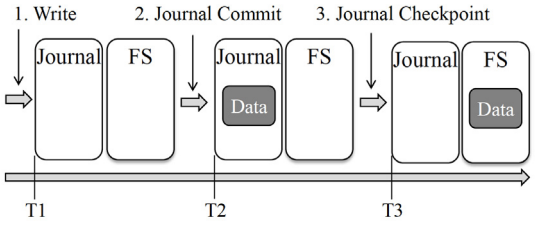

Fig. 1. Journaling mechanism. When a new write operation is issued at T1, then the data will be committed to the journal area at T2. After then, the data will be finally checkpointed to the file system at T3.

committed log is the key for file system recovery because the data will not be physically stored into the file system before the log is committed. Because of the reliability support, modern file systems include journaling mechanism to maintain their state consistently [1, 2, 3, 4]. Figure 1 shows the basic operation of journaling mechanism.

Although the journaling mechanism serves reliability in unexpected crash, there is significant performance degradation. The data block will be written twice - to the journal area and then to the file system. In order to mitigate this problem, many existing file systems support various journaling modes. For example, the Ext3 file system supports three journaling modes. The journal mode supports that all data blocks and metadata blocks are committed into the journal area prior to being written into the file system. This is the most reliable journaling mode because it can recover data blocks and metadata blocks simultaneously. However, this is the slowest mode as well. Thus under data reliability is seriously important situation, the system administrator sets journal mode. In the ordered mode, only metadata blocks are committed to the journal area but the written order is forced. The data blocks are directly written to the file system prior to metadata blocks being committed to the journal area. Although it only commits metadata blocks, the ordering control serves considerable recovery chance and it gives

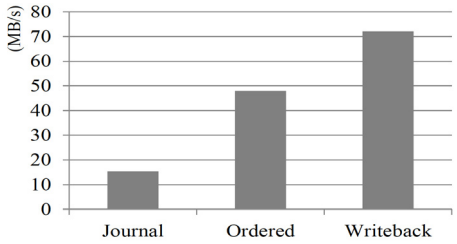

Fig. 2. I/O throughput of fileserver workload in the Filebench comparing with various journaling modes under Ext3 file system.

moderate write overhead. This is the default journaling mode for the file system. The last journaling mode is writeback. It only commits metadata blocks like ordered mode but there is no ordering control. Therefore, it has low chance to recover data but this is the fastest journaling mode. As explained above, there are trade-off relationship between reliability and write latency.

In this paper, we propose a reliable and fast journaling mechanism named RFJ to solve the trade-off relationship. The RFJ not only supports high reliability of the journal mode but also achieves low write latency of the ordered mode.

Contributions of this paper are as follows.

1. Detailed explanation of the trade-off relationship between file system reliability and performance under various journaling modes.
2. A new journaling mode named Ordering enforced writeback is proposed. It guarantees the ordering between the metadata and the data but it works as fast as writeback mode.
3. A dynamic journaling mode selection method makes this system as reliable as journal mode. It only journals overwritten data blocks.

2. Trade-off between I/O Performance and Reliability of File System

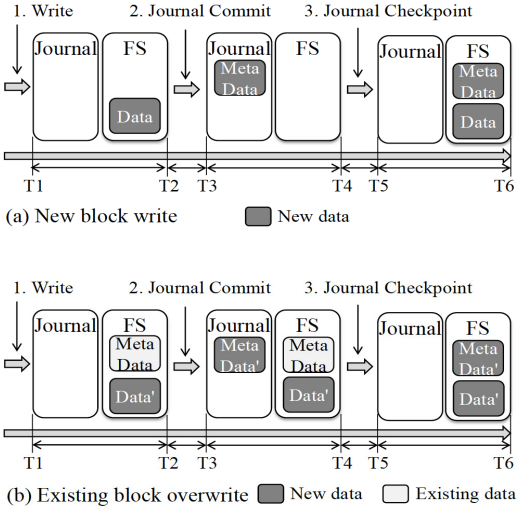

Fig. 3. Ordered mode in the Ext3 file system. (a) depicts new block write case and (b) depicts existing block overwrite

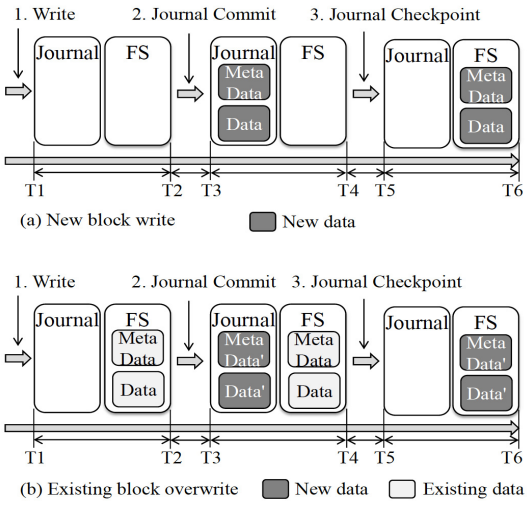

Fig. 4. Journal mode in the Ext3 file system. (a) depicts new block write case and (b) depicts existing block overwrite

In this section, we describe the file system performance and inconsistency for each journaling modes. As explained in the section 1, there are three journaling modes in the Ext3 file system. The relationship among various journaling modes is trade-off between file system reliability and I/O performance.

2.1 I/O Throughput

In order to see the performance difference, we evaluate each journaling mode of Ext3 file system using Filebench benchmark [5]. The workload fileserver emulates simple fileserver I/O activity. This workload performs a sequence of creates, deletes, appends, reads, writes and attribute operations on a directory tree. 50 threads are used by default. The evaluation is conducted on Intel Xeon E5620 2.4 GHz x 8 cores with 16 GB of RAM and the hard disk drive is 7,200 RPM with SATA interface. Figure 2 shows the result of the evaluation. The journal mode shows the worst performance for these two workloads since the journal mode journals not only metadata blocks but also data blocks. This causes doubled write for all write operation. In contrast, the ordered and writeback modes show much better performance because they journal only metadata blocks. Furthermore, the writeback mode does not enforce the ordering control. Thus it shows the best performance. Note that metadata size is much smaller then data blocks. However, the reliability of each journaling mode is also different.

2.2 Reliability of Writeback mode

Although the writeback mode shows best performance, it has the lowest reliability among all journaling modes. Though it journals the metadata into the journal area, there is no ordering control in this mode. Therefore, system crash can cause file system inconsistency when the data block has not been written. In this mode, when the file system recovery tool such as fsck[6] begins to recover, the metadata in the journal area does not guarantee anything. At most, the tool can recognize that there can be a problem when the metadata is remained in the journal area.

2.3 Reliability of Ordered mode

The ordered mode is the default journaling mode for the Ext3 file system because this mode supports a nice scheme. In this mode, the data blocks are directly written to the file system prior to metadata blocks being committed to the journal area. That is, the metadata block cannot be committed to the journal area without data blocks writing. This enforced ordering control efficiently guarantees file system consistency in many cases. Figure 3 shows the operation sequence of ordered mode. In the case of new block write (Figure 3-(a)), file system can always be consistent. Though written data block can be lost, the file system always maintains consistent state. Note that, guaranteeing file system consistency is not guaranteeing data block recovery. However the case of block overwrite (Figure 3-(b)), the file system can be remained inconsistent. If system is crashed from time T1 to T3, then the existing data will be lost. In this situation, the file system metadata points to the existing data but there are no such data anymore due to the overwrite. That is, the metadata - data connection semantics is invalid.

2.4 Reliability of Journal mode

Figure 4 depicts the journaling operation sequence for journal mode in the Ext3 file system. In this mode, all metadata blocks and data blocks are committed to the journal area and then checkpointed to the file system. Therefore, the file system can always be maintained consistently. Even in the block overwrite case, the metadata - data connection semantics is still valid. If the system is crashed from time T4 to T5 and the metadata block has not been overwritten and the data block has been overwritten. In this case, the file system recovery tool can correct the inconsistency because there are still metadata and data blocks in the journal area.

3. Design

In this section we describe some observations of file access pattern and the current ordering mechanism. Then, we show the architecture of the RFJ and we explain a new journaling mode named ordering enforced writeback mode.

3.1 Observations

In order to see the benefit of selective journaling, we evaluated real world traces. We classified each block write operation into two cases: new block writing and existing block overwriting case. Figure 5 shows the classification result of real world traces from several production servers at Microsoft [7]. Some traces such as BuildServer2 are block overwrite dominant. Because of the characteristics of building process, there are many file modifications.

However, most of traces are new block write dominant. If we apply journal mode to these workloads, the journaling mechanism will do unnecessary write operation because the new block write case can be covered by ordered mode. That is to say, to maintain file system consistently, journal committing of data block is not required for new block write case. Only block overwrite case requires journal committing of data block.

We analyzed the ordering mechanism of the existing file system (Ext3). The ordering is guaranteed by the journaling daemon such as kjournald kernel thread. The journaling daemon forces synchronous I/O request submission to flush data buffer to the I/O scheduler and waits for the completion of the request before it journals metadata. This mechanism always guarantees the ordering semantic between the data buffer and the metadata buffer. However, the ordering operation causes additional overheads that makes ordered mode slower. The forced data flushing operation breaks the I/O

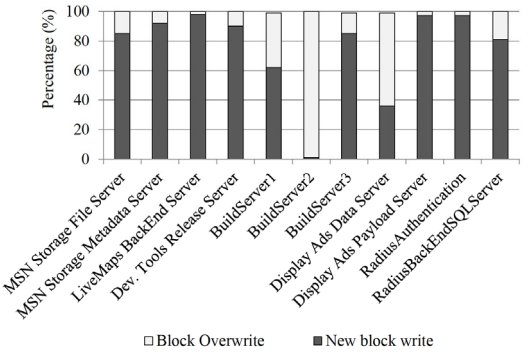

Fig. 5. Classification of write operation on real world workload

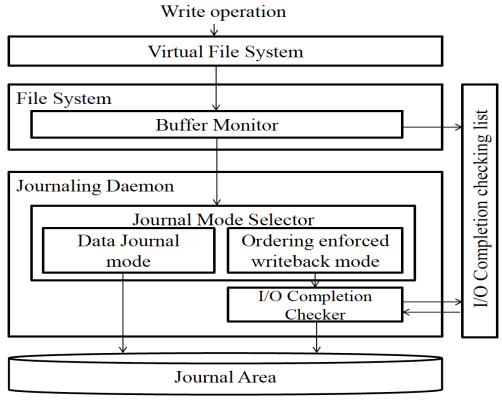

Fig. 6. Architecture of RFJ

submission optimization of the I/O scheduler. That is, if we use the kernel's default buffer flusher such as pdflush or per-BDI flusher instead of flushing data buffer directly in the journaling daemon, then we can achieve higher performance.

3.2 Architecture

Based on the observations, we propose a new journaling technique to achieve reliable and fast journaling. Figure 6 shows the architecture of RFJ. It consists of Buffer Monitor, Journal Mode Selector, I/O Completion Checker and I/O Completion checking list.

The Buffer Monitor marks each buffer's state. If current buffer is already existed then the buffer is marked as overwritten buffer and if it is newly allocated buffer, then it is marked as new buffer. If current buffer is marked as a new buffer, then the buffer entry is inserted to the I/O completion checking list. The Journal Mode Selector selects journaling mode based on the buffer's state. If the buffer is marked as overwritten, then it is processed as data journaling mode. That is the same with the journal mode in the Ext3 file system. If the buffer is marked as a new buffer then it is processed as Ordering enforced writeback mode. The Ordering enforced writeback mode does not directly flush data buffer to the disk. The data buffer is flushed by kernel's default flusher such as pdflush or per-BDI flusher. It just waits for the I/O completion of the data buffer and if the data buffer is written to the disk, then it begins to journal the metadata buffer. To do this, the I/O Completion Checker waits and polls checks the state of the buffers in the I/O Completion checking list. If the buffer is successfully written by the kernel's default flusher, the state of the buffer is modified to I/O completed state. Thus, the I/O completion checker can check the state of the buffer. Figure 7 depicts detailed operation.

4. Evaluation

4.1 Experimental Environment

We implement proposed method based on the Ext3 file system. The Ext3 file system uses Journaling Block Device as a journal area. We modified this JBD and the journaling daemon. Experimental environment has 8 cores of Intel Xeon E5620 2.4 GHz with 8 GB of RAM. We used Ubuntu 12.04 LTS as operating system that runs the Linux kernel version 3.5.0.

4.2 Reliability analysis

The RFJ has the same reliability with the Journal mode of the Ext3 file system. The original

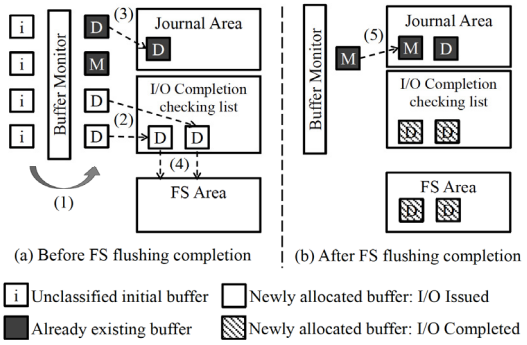

Fig. 7. Detailed operation. D means data and M means metadata. When a new write operation is issued, the buffer monitor classifies each buffer (1) and the newly allocated buffers are inserted to the I/O Completion checking list.(2). After then, the journaling daemon wakes up and journals the already existing buffers (3). At this point, it cannot journal the metadata buffer since there are still data buffers that are not written to the disk. Because there is nothing to do, the journaling daemon sleeps. Later, kernel's buffer flusher flushes the newly allocated buffers (4) then the buffers in the checking list are changed to I/O Completed. When the journaling daemon wakes up, now it can journal the metadata (5) because of the data buffers are resided in the FS area. After that, the checking list is cleared.

journal mode journals all data blocks but RFJ selectively journals data blocks that is overwritten. For the newly allocated blocks, ordering control is enough to maintain file system consistently. In additional, we do not add or modify data structure for the journaling for the compatibility with existing recovery tools.

4.3 I/O Throughput

Figure 8 shows the result of Filebench[5] benchmark. The RFJ shows almost 5 times better performance than the Journal mode. It also outperforms the ordered mode. We think that the Filebench benchmark contains lots of new writes. If it contains lots of overwrite, the performance will be lower. However, as we analyzed in section 3.1, this kind of workload is general case.

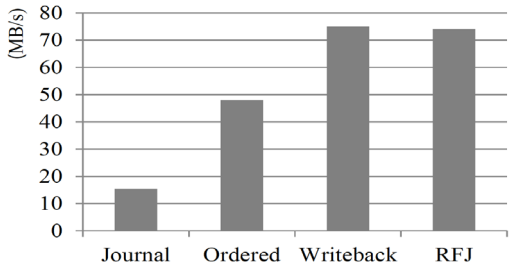

Fig. 8. Throughput comparison to the default journaling modes in the Ext3 File system. Result of File server workload in the Filebench.

This shows that we can achieve almost the same I/O throughput with the writeback mode but we can achieve the reliability of journal mode at the same time.

5. Related Works

Modern file system has journaling feature to maintain its state consistently. Some file system like Journaling File System, JFS [3] is natively designed for journaling. Or some file system like Ext2 [9] adds journaling feature to the next versions [1, 2]. Prabhakaran et al. [13] analyzed and gave various experimental results about several different file systems. In [11], it supports selective journaling method. However, the meaning of selective is different. In [11], a user or administrator can select the target files or directory for journaling.

Lee et al. [12] proposed a different approach named UBJ to achieve fast and reliable journaling. The UBJ uses non-volatile random access memory like Phasechange RAM. They unified the buffer cache with the journal area in the NVRAM. However, this approach requires special hardware support. Choi et al. [8] proposed a flash translation layer based on a journal remapping for a flash memory. They exploited the fact that NAND based flash devices use out-of-place update. They applied journal concept on the FTL

layer but this approach requires special interface from the flash device to the file system to communicate. Chidambaram et al. [15] proposed optimistic crash consistency. Although they proposed similar approach like selective journaling, their approach requires hardware modification to support new I/O interface.

6. Conclusion

In this paper, we proposed a reliable and fast journaling mechanism named RFJ. We have shown that the journal commit for new block write operation is unnecessary for the journal mode. In addition, the direct data buffer flushing in the journaling daemon degrades the I/O optimization of the I/O scheduler. Based on these observations, we proposed selective journaling mechanism and Ordering enforced writeback journaling mode. The proposed mechanism achieves high I/O throughput and high reliability at the same time. Moreover, the RFJ does not add or modify the data structures related to journaling and this gives perfect compatibility for existing recovery tool.

Because the result of the proposed mechanism is impressive, currently, we are working on adapting this algorithm to Ext4 which uses JBD2 as journaling daemon.

References

[1] S. C. Tweedie. EXT3, Journaling File System. ol-strans. sourceforge.net/ release/OLS2000-ext3/OLS2000-ext3.html

[2] Mathur, Avantika, et al. "The new ext4 filesystem: current status and future plans." Proceedings of the Linux Symposium. Vol. 2. 2007.

[3] S. Best. JFS Log. How the Journaled File System performs logging. In Proceedings of the 4th Annual Linux Showcase and Conference, pages 163- 168, Atlanta, 2000.

[4] Mason, Chris. "Journaling with reisersfs." Linux Journal 2001.82es (2001): 3.

[5] Filebench, http://www.solarisinternals.com/

[6] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. Fsck - The UNIX File System Check Program. Unix System Manager's Manual 4.3 BSD Virtual VAX-11 Version, April 1986.

[7] Kavalanekar, Swaroop, et al. "Characterization of storage workload traces from production windows servers." Workload Characterization, 2008. IISWC 2008.

[8] Jianxi Chen, Qingsong Wei, Cheng Chen, and Ling kun Wu, FSMAC: A file system metadata accele rator with non-volatile memory, MSST, May 2013.

[9] Bovet, Daniel P. Understanding the Linux kernel. O'reilly, 2007.

[10] Design and Implementation of the Second Extended Filesystem, http://e2fsprogs.sourceforge.net/ext2intro.html

[11] Symantec, Enterprise Vault, http://www.enterprisevault.com

[12] Lee, Eunji, Hyokyung Bahn, and Sam H. Noh. "Unioning of the Buffer Cache and Journaling Layers with Non-volatile Memory." 11th USENIX Conference on File and Storage Technologies. 2013.

[13] Prabhakaran, Vijayan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. "Analysis and Evolution of Journaling File Systems." USENIX Annual Technical Conference, General Track. 2005.

[14] Choi, Hyun Jin, Seung-Ho Lim, and Kyu Ho Park. "JFTL: A flash translation layer based on a journal remapping for flash memory." ACM Transactions on Storage (TOS) 4.4 (2009).

[15] Chidambaram, Vijay, et al. "Optimistic crash consistency." Proceedings of the TwentyFourth ACM Symposium on Operating Systems Principles. ACM, 2013.

박 세 진(Sejin Park) [정회원]

- 2007년 2월 : 금오공과대학교 소프트웨어공학과 (공학사)
- 2016년 2월 : 포항공과대학교 컴퓨터공학과 (공학박사)
- 2016년 10월 ~ 2018년 2월 : SKTelecom Manager
- 2018년 3월 ~ 현재 : 계명대학교 교수

〈관심분야〉
시스템 소프트웨어, 운영체제, 블록체인