R-Barrier: Rapid Barrier for Software RAID Cache Using Hints from Journaling Filesystem

Chulmin Kim
Electrical Engineering
KAIST
Daejeon, Republic of Korea
cmkim@core.kaist.ac.kr

Sung Hoon Baek
Computer Science
Jungwon University
Goesan, Republic of Korea
shbaek@jungwon.ac.kr

Kyu Ho Park
Electrical Engineering
KAIST
Daejeon, Republic of Korea
kpark@ee.kaist.ac.kr

***Abstract*—While adopting cache in software RAID brings performance benefit, it can cause data loss at power failure which results in the broken filesystem consistency. Though I/O Barrier can be used to remove the consistency issue, it sacrifices the write performance of software RAID.**

To mitigate this tradeoff, we suggest R-Barrier for software RAID. The basic idea of R-Barrier is to avoid the synchronization of the entire cache space. Instead, based on given hints from the filesystem layer, R-Barrier makes software RAID cache select one of the following methods when the cache writes a certain write request into the disk: (1) *Strictly ordered destaging* for the write requests affecting the filesystem consistency; (2) *Reordered destaging* for the rest of write requests. We show that R-Barrier guarantees same filesystem consistency with I/O Barrier while the performance degradation of R-Barrier is less than that of I/O Barrier.

***Keywords*-Caching; Barrier; Filesystem; RAID;**

I. Introduction

For the filesystem consistency, the recent filesystem contains journaling [1] technique for its write operation. Even when the write operation is halted unexpectedly, the technique always guarantees to atomically update the *atomic unit* (a set of the blocks modified by *write()*) by writing *atomic unit* twice in the underlying storage (once in temporal region, called *journal*, and once in the home location).

However, due to the cache in the storage such as RAID, the complete filesystem consistency can be obtained only when the storage supports it. While the cache brings performance benefit from request reordering such as STOW [2], it can result in the cached data loss at power failure which breaks the filesystem consistency. The problem can be resolved when hardware RAID is used. However, it needs relatively high cost for non-volatile RAM or battery-backed DRAM. In case of software RAID, though there is no way to prevent the data loss, I/O Barrier [3] which frequently synchronizes the cache has been adopted so far to prevent the broken filesystem consistency.

While I/O Barrier is able to avoid the problem, the frequent cache synchronization of I/O Barrier nearly disables the write cache functionalities including the request reordering. As a result, the software RAID with I/O Barrier should give up the performance benefit from the request reordering.

To mitigate the demerit of I/O Barrier, we suggest a new concept replacing I/O Barrier, named R-Barrier. The basic idea of R-Barrier is to avoid the frequent synchronization of software RAID cache. Instead, R-Barrier makes software RAID cache select one of the two different destaging methods when the cache destages (writes cached data to the disk) a certain write request: (1) *Strictly ordered destaging*; (2) *Reordered destaging*. Through *strictly ordered destaging*, R-Barrier keeps the order of the requests for the required filesystem consistency. Other requests are freely destaged by *reordered destaging* to increase the write performance of software RAID.

II. Design of R-Barrier

R-Barrier adopts the methodology shown in Figure 1. The design starts from avoiding the cache synchronization which make the write cache malfunction. Instead, R-Barrier controls the destaging order of write requests in software RAID cache. Based on whether a request is related with the filesystem consistency or not, R-Barrier classifies write requests into two groups with different destaging methods : *strictly ordered destaging* and *reordered destaging*, respectively. With R-Barrier, the reordering algorithm can obtain the performance benefit by reordering the write requests in the second group.

To enable the operation of R-Barrier We assume that R-Barrier can receive two per-request hints from the filesystem. The request type (JD,JM, and etc. in the figure) indicates the relationship with the filesystem consistency while the another, R-Window number in the figure, is for recognizing the order between write requests.

In case of Ext3 Filesystem [1] used for our prototype, the write requests for the *atomic unit* belongs to the first group while the others are in the second group. For data-level consistency, the *atomic unit* includes both metadata and data blocks. No request type exists for the second group. For metadata-level consistency, the *atomic unit* includes metadata blocks only, and the data blocks belong to the second group. Therefore, R-Barrier works better than I/O

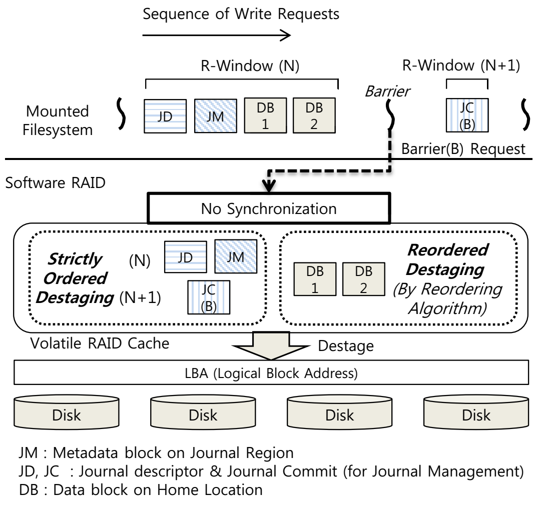
Sequence of Write Requests
R-Window (N)
R-Window (N+1)
Barrier
Mounted Filesystem
JD
JM
DB 1
DB 2
JC (B)
Barrier(B) Request
Software RAID
No Synchronization
Strictly Ordered Destaging
(N)
(N+1)
Reordered Destaging (By Reordering Algorithm)
Volatile RAID Cache
Destage
LBA (Logical Block Address)
Disk
JM : Metadata block on Journal Region
JD, JC : Journal descriptor & Journal Commit (for Journal Management)
DB : Data block on Home Location

Figure 1. R-Barrier Methodology

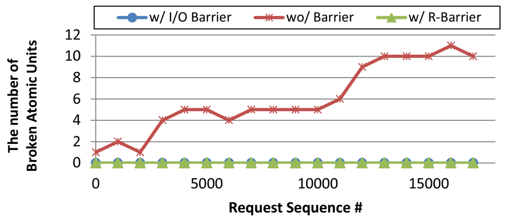

Figure 2. Trace Simulation detecting Broken *Atomic Units*

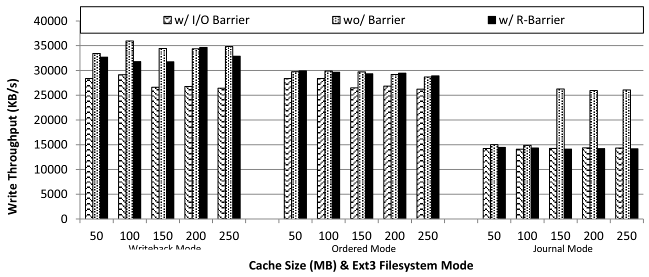

Figure 3. Write Throughput of Barrier Schemes varying Cache Size and Modes of Ext3 Filesystem

Barrier for metadata-level consistency while it works in the same manner with I/O Barrier for data-level consistency.

1) Strictly Ordered Destaging: The destaging method of this group has two kinds of rules. First, the method keeps the given order from the filesystem between the write requests in this group. Since the journaling is based on writing the *atomic unit* twice, the sequence of the write requests for the *atomic unit* should be kept. Second, the method does not allow the buffering of the write requests. When the buffering is allowed, certain data even can be eliminated before it is written to the physical media. The destaging order will not be same with that given by the filesystem. Since this will finally results in the broken *atomic unit*, R-Barrier prevents the buffering.

2) Reordered Destaging: For this group, there is only one rule for the operation. R-Barrier passes the write requests in this group to its reordering algorithm, so that the algorithm can earn the benefit of write performance.

III. Evaluation

To test the filesystem consistency, we logs I/O activity and replay the log to count the number of broken *atomic units* for software RAID with Journal mode Ext3 Filesystem varying the barrier scheme (I/O Barrier, R-Barrier, and ”without Barrier”). If power failure occurs at certain time, the broken *atomic unit* will result in the broken filesystem. Figure 2 illustrates the result. We can check that R-Barrier has no broken *atomic unit* at all as I/O Barrier does.

In Figure 3, we measured the throughput result varying cache size and the modes of Ext3 Filesystem. we used the 2 threaded sequential workload (1.5GB/thread) generated by IOzone. Software RAIDs with various barrier schemes have been set with 1 disk and RAID 0 configuration. We can see that the performance of R-Barrier outperforms that of I/O Barrier and almost approaches to that of ”without Barrier” scheme except in Journal mode. Since the journal area is the circular buffer, RAID cache with ”without Barrier” scheme overwrites the write request on that on the same journal location. This buffering results in the doubled throughput. However, since it cannot provide the filesystem consistency at power failure, the throughput is meaningless.

IV. Conclusion

We propose a new barrier scheme, named R-Barrier, to replace I/O Barrier. With the assist from the filesystem, R-Barrier selectively forces the ordering of the write requests, or passes the write requests to the reordering algorithm. We proved that R-Barrier guarantees the same level of the filesystem consistency with I/O Barrier, and it outperforms I/O Barrier as well.

Acknowledgment

This work was supported by Ministry of Knowledge Economy, Republic of Koread (Project No. 10035231).

References

[1] S. Tweedie, “EXT3, Journaling Filesystem,” in *Proceedings of the Ottawa Linux Symposium*, 2000.

[2] B. S. Gill, M. Ko, B. Debnath, and W. Belluomini, “STOW: a spatially and temporally optimized write caching algorithm,” in *Proceedings of the 2009 conference on USENIX Annual technical conference*, ser. USENIX’09. Berkeley, CA, USA: USENIX Association, 2009, pp. 26–26. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855807.1855833

[3] T. Heo. (2005) Linux Kernel Document : I/O Barriers. [Online]. Available: http://www.mjmwired.net/kernel/Documentation/block/barrier.txt