Enabling Manycore Scalability in F2FS Metadata for *unlink()* Operation

Soon Hwang, Chang-Gyu Lee, Youngjae Kim
Dept. of Computer Science and Engineering, Sogang University, Seoul, Republic of Korea
{hs950826,changgyu,youkim}@sogang.ac.kr

CCS CONCEPTS

• **Software and its engineering → File systems management**;

KEYWORDS

Operating System, File System

1 MANYCORE SCALABILITY IN F2FS

Manycore systems enable massive parallel I/O in a single server due to the number of cores. Among file I/O operations in a file system, C. Lee et al. [1] applied range lock in F2FS for parallel data I/O, and showed scalable performance. However, little research has been done on metadata I/O scalability.

To investigate this, we analyzed *unlink()* with related data structures in F2FS. File metadata in F2FS (inode) is called *Node* and identified via *nid*. *Nodes* are stored in an on-disk structure, called Node Address Table (NAT), which is cached in memory with a pool of free *nids*. F2FS keeps a certain number of free *nids* for fast *nid* allocations during *create()*. Every time *unlink()* is called, the number of free *nids* in the pool is checked. If it is not sufficient, a *Free nid Scan* function is performed to secure sufficient free *nids*.

We evaluated the I/O performance when multiple threads call *unlink()* in F2FS in a manycore system, and it shows no performance scalability. From the analysis above, we identified that a large critical section(CS) in *Free nid Scan* by a mutex lock is the leading cause of the scalability bottleneck.

2 PROBLEM AND PROPOSED DESIGN

Free nid Scan has two steps in F2FS. The first step is to scan the free *nid* bitmap in memory, and the second step is to scan NAT directly from the SSD. These two steps are held in a large CS by a mutex lock in *Free nid Scan*. Thus, if there is a thread (T_1) running *Free nid Scan*, other threads will be blocked until it exits *Free nid Scan*. The parallelism of threads executing *Free nid Scan* is limited by such a large CS. If T_1 has obtained sufficient free *nids*, then other threads don't have to go through *Free nid Scan*. However, all competing threads currently run both steps unnecessarily. This inefficient design increases the blocking time of threads. To solve this problem, we propose the *Optimistic Free nid Scan* to minimize the blocking time of threads. In the *Optimistic Free nid Scan*, the large CS is split into two smaller CSs. Thus, it increases the parallelism among competing threads. Consider *n* threads competing in the *Optimistic Free nid Scan*. If T_1 is in the first step and T_2 is executing the second step, $T_3 - T_n$ can avoid *Free nid Scan* as T_1 or T_2 will ensure to secure sufficient free *nids* to a threshold. With the *Optimistic Free nid Scan*, threads that were previously blocked will not be blocked any longer, increasing a thread's parallel execution efficiency.

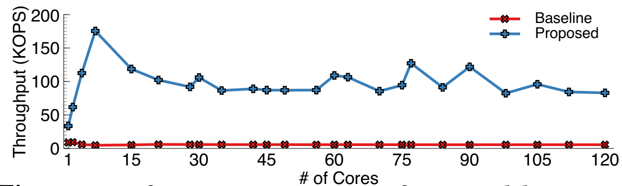

Figure 1: Performance comparison of proposed design with baseline F2FS on a 120 core machine.

Second, in the first step of *Free nid Scan*, every free *nid* bitmap scan begins at the beginning of the bitmap. Thus, a thread has to search for free *nids* in the bitmap already scanned by the preceding thread. This increases the latency of free *nid* bitmap scan unnecessarily. Thus, we propose the *Heuristic Free nid Bitmap Scan* where the bitmap scan starts scanning from the point where the previous bitmap scan ended. By reducing the work performed in the free *nid* bitmap scan, the total bitmap scanning latency is reduced and the blocking time of threads is minimized.

3 EVALUATION

We evaluated the proposed design for F2FS on a 120-core machine with a Samsung 970 EVO SSD using the FxMark MWUL workload [2]. In MWUL, multiple threads each issue *unlink()* to their own directory. As shown in Figure 1, our design outperformed the baseline due to the reduced lock contention in *Free nid Scan*. However, it only scales to 15 cores since it does not fully eliminate the lock contention.

ACKNOWLEDGMENTS

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2014-3-00035, Research on High Performance and Scalable Manycore Operating System).

REFERENCES

[1] C. Lee et. al. 2019. Write Optimization of Log-Structured Flash File System for Parallel I/O on Manycore Servers. In *SYSTOR*. 21–32.
[2] C. Min et. al. 2016. Understanding Manycore Scalability of File Systems. In *USENIX ATC*. 71–85.

SYSTOR '21, June 14–16, 2021, Haifa, Israel

ACM ISBN 978-1-4503-8398-1/21/06…$15.00
https://doi.org/10.1145/3456727.3463832