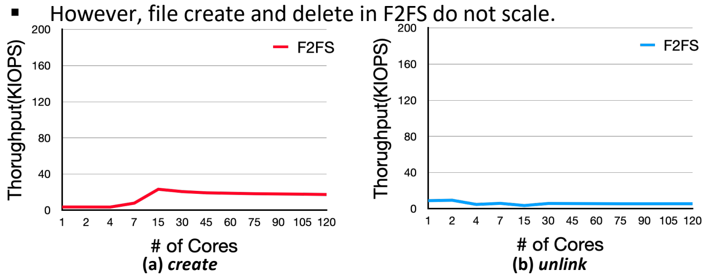
Enabling Manycore Scalability in F2FS Metadata for *unlink()* Operation

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

F2FS Metadata Scalability Limitations

- Manycore servers are expected to bring great scalability in file systems due to their large number of cores.
- However, file create and delete in F2FS do not scale.



(a) *create* (b) *unlink*

Figure 1. < Manycore scalability in file *create* & *unlink* in F2FS>

- We tested file create & unlink scalability with F2FS using FxMark.

Unlink Metadata Operation in F2FS

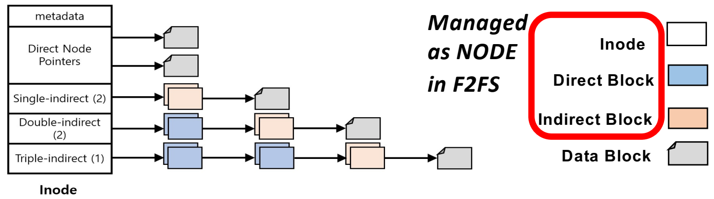


Figure 2. < F2FS File Structure>

- Figure 2 shows the F2FS File Structure with Node.
- Node is identified via *nid* and stored in an on-disk structure, Node Address Table (NAT).

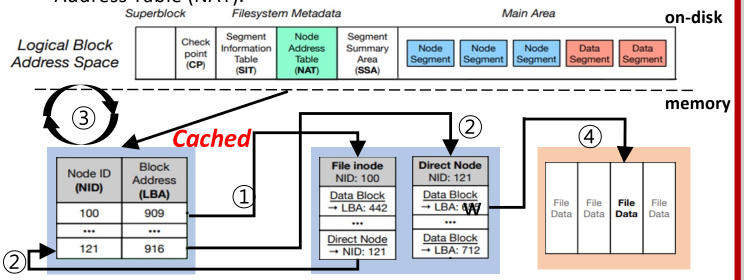


Figure 3. <*unlink()* process in F2FS>

- Figure 3 shows how F2FS processes *unlink()* with on-disk & in-memory data structures including NAT.

Step ①. Find inode by checking cached node in memory.
Step ②. Find the Direct Node's id in cached NAT.
Step ③. Check the number of *Free nids*. If it is not sufficient, refill *Free nid.*
Step ④. Delete Direct node's link to File data that will be deleted.

- F2FS should maintain enough *Free nid*s for future *create().*
- *Free nid* Bitmap to check *Free nid* is used to obtain *Free nid* fast in memory.
- If *Free nid* is below the threshold, F2FS will run *Free nid* Scan, which scans the *Free nid* Bitmap and checks the NAT directly.

Metadata Scalability Bottlenecks

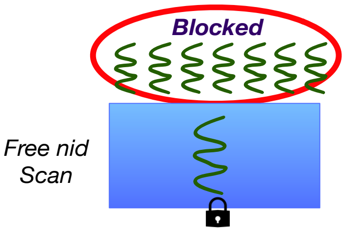


Figure 4. < Blocking in *Free nid Scan* >

- The main cause of the scalability bottleneck of F2FS for parallel *unlink* operations in the Manycore system is a large critical section (CS).
- Figure 4 shows that parallel processing efficiency of threads executing *unlink()* is highly limited by a large CS in *Free nid Scan.*

Proposed Design and Implementation

We propose two techniques(Optimistic *Free nid* Scan, *Heuristic Free nid* Bitmap Scan) to mitigate thread execution efficiency in parallel ***unlink()*** and unnecessary search in *Free nid Scan* **.**

1. Optimistic *Free nid Scan* divides the *Free nid Scan* into two parts to increase the thread execution efficiency.

Step ① . Scanning *Free nid* Bitmap.
Step ②. Only case for Step 1 fails, fill *Free nid* from NAT in the SSD.

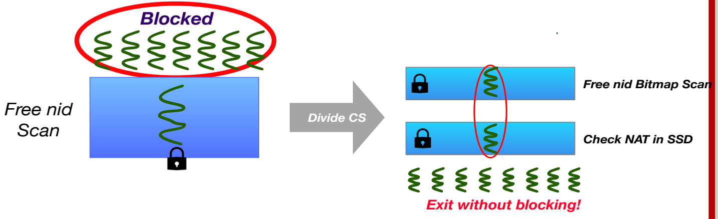
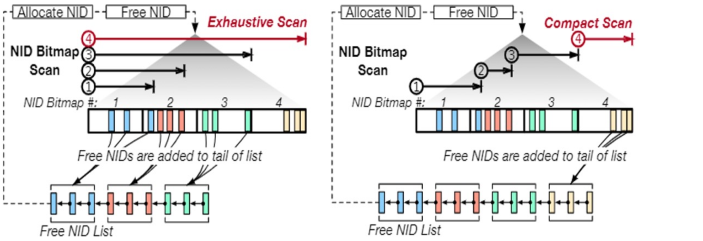


Figure 5. < Improved Parallel I/O by Optimistic *Free nid Scan* >

- In vanilla F2FS, most of the threads are blocked until the preceding one finishes *Free nid* Scan.
- By the *Optimistic Free nid Scan*, threads that were previously blocked will not be blocked any longer, increasing thread's parallel execution efficiency.

2. Heuristic *Free nid* Bitmap Scan starts scanning from point where previous Bitmap Scan ended.

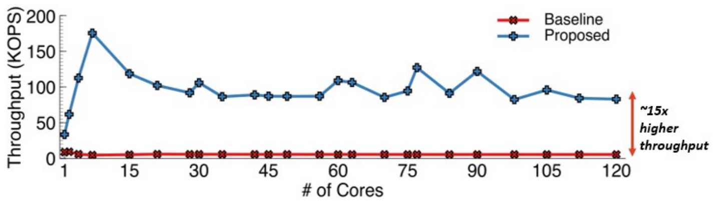


(a) Exhaustive Scan in vanilla F2FS **(b) Compact Scan by Heuristic *Free nid* Bitmap Scan**

Figure 6. < Scan length reduction by Heuristic *Free nid* Bitmap Scan>

- In vanilla F2FS, *Free nid* Bitmap Scan starts from the beginning of the bitmap, increasing the latency of *Free nid* Bitmap Scan.
- Figure 6 shows by Heuristic *Free nid* Bitmap Scan, the total bitmap scanning time is reduced, and the blocking time of threads is minimized.

Evaluation



- We evaluated our proposed design on 120-core manycore server equipped with 740GB memory and Samsung 970 EVO SSD.
- We compared the proposed approach with Baseline (Vanilla F2FS version).
- We tested using MWUL workload in FxMark, where multiple threads perform unlink in their private directory in MWUL workload.
- The proposed approach outperforms Baseline F2FS and improved manycore scalability to 15 cores.
- Throughput of proposed design sustains after 15 core due to the mutex lock in the call path. We identified this is the fundamental limiting factor.

Acknowledgement

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. 2014-0-00035, Research on High Performance and Scalable Manycore Operating System).