Concurrent File Metadata Structure Using Readers-Writer Lock

<u>Chang-Gyu Lee</u>, Sunghyun Noh, Hyeongu Kang, Soon Hwang, Youngjae Kim

Sogang University, Seoul, Republic of Korea

Manycore Server and Parallel I/O

- Manycore CPU enables a single server with tens to hundreds of core.
- Parallel I/O is the key to improve I/O throughput using the massive number of cores.

2P Intel vs 1P EPYC comparison[7]

	x86 PROCESSOR + x86 PROCESSOR	AMD EPYC
Model	2x6262V	1x7702P
Cores	48	64
Memory Capacity	2TB	4TB
Max Memory Frequency	2400MHz	3200MHz[6]
I/O Lanes	96 PCIe® 3.0	128 PCIe® 4.0[6]
TDP	270Watts	200Watts
SPECrate®2017_int_base	242	319
'Per Socket software' licensing cost	x2	x1
List Price	5800USD	4425USD

* https://www.amd.com/en/processors/epyc-7002-series

Overview of 3rd Gen Intel Xeon Scalable processors

- Up to 28 cores per Intel Xeon Scalable processor
- 6 memo
- Features
  and VNN ce

- Up to 224 cores per node in an 8-socket configuration

* https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/3rd-gen-xeon-scalable-processors-brief.pdf

Manycore Server and Parallel I/O

- However, there are several things we need to check before actively adopts parallel I/O into the manycore system

1. **Fast and highly parallel storage device**
   enough to accommodate parallel I/O requests from the manycore system

   PCIe connected NVMe SSD such as Intel Optane SSD

2. **The file system design**
   has to remove inode mutex, which serializes I/O requests to a shared file

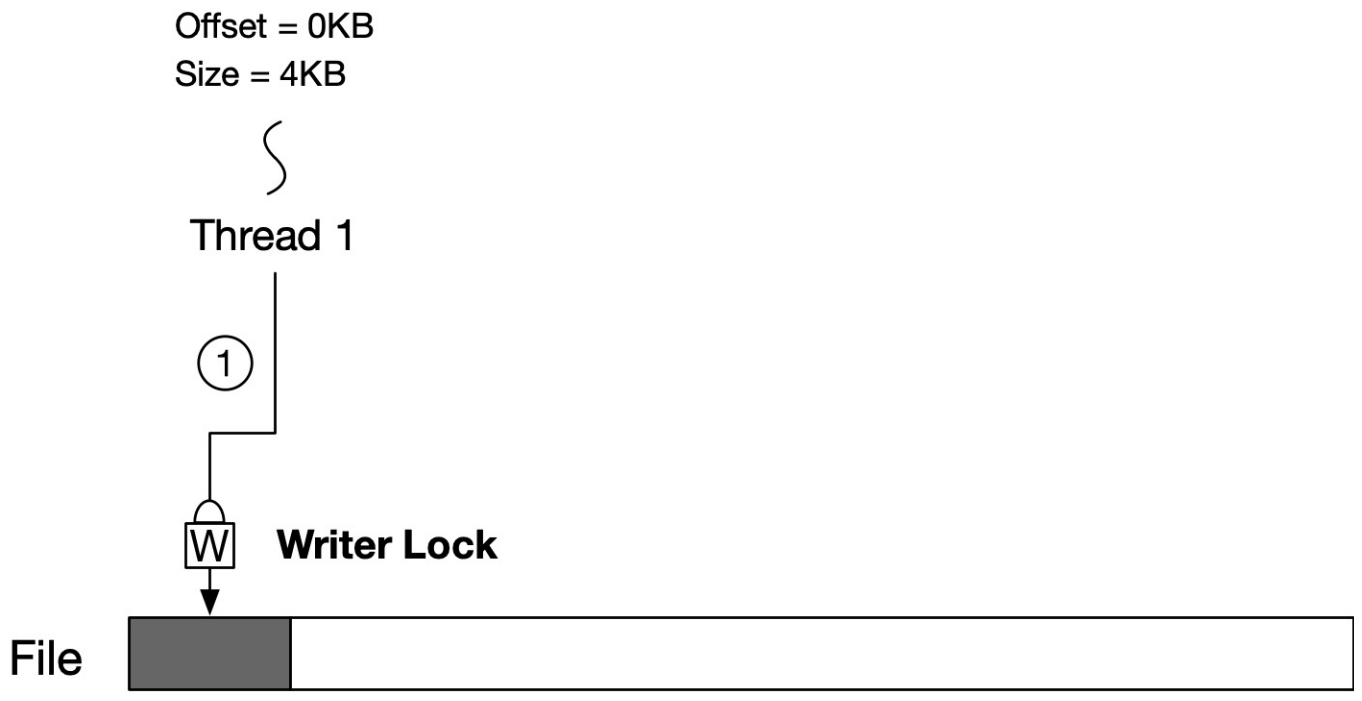
   Fineer-grained locking mechanism based on the extent of I/O operation (Range Lock)

3. **Keep the POSIX requirements**

   Recent studies presented the Range Lock with POSIX-compliant the file system
   i.e., pNOVA (APSYS'19), F2FS-RL (SYSTOR'19), CrossFS (OSDI'20)

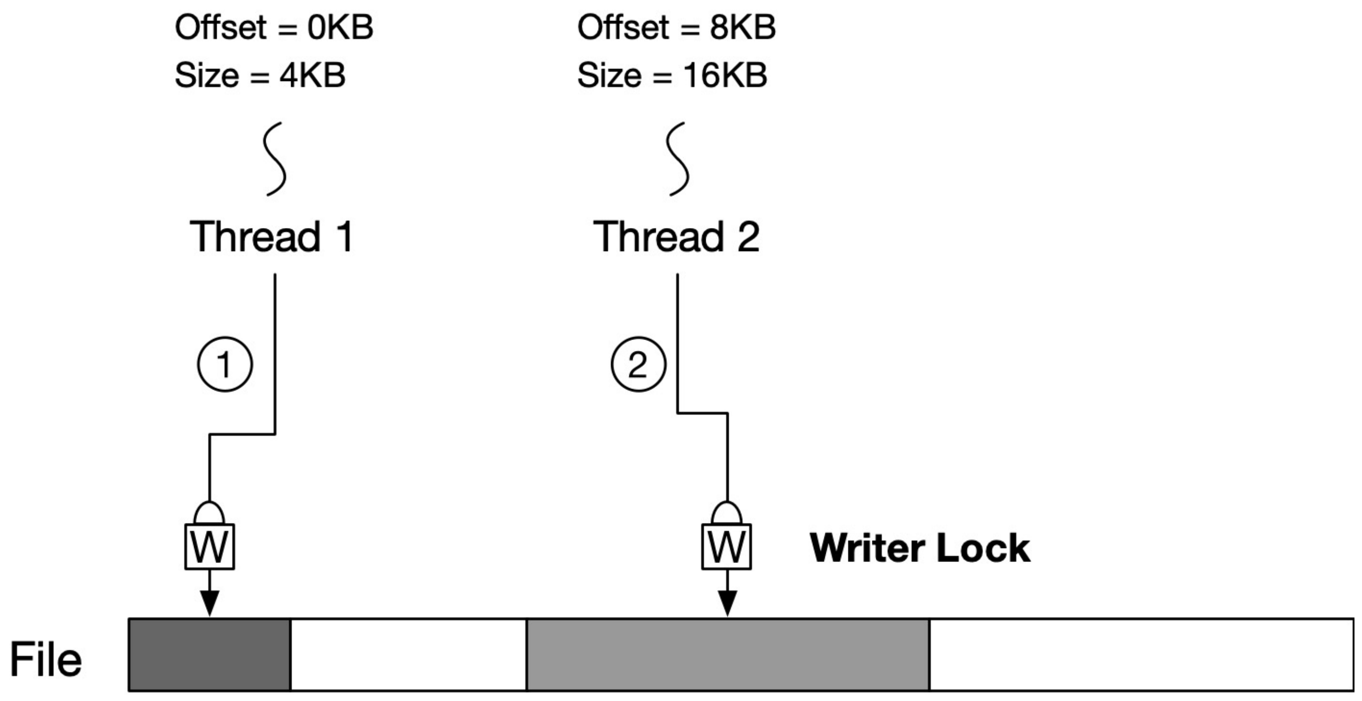
Range Lock to Enable Parallel IO

- Consider three threads are writing the same file.
  - Thread 1 and Thread 2 are writing non-overlapping ranges.
  - Thread 3 is writing the file range overlapping with Thread 2's range.



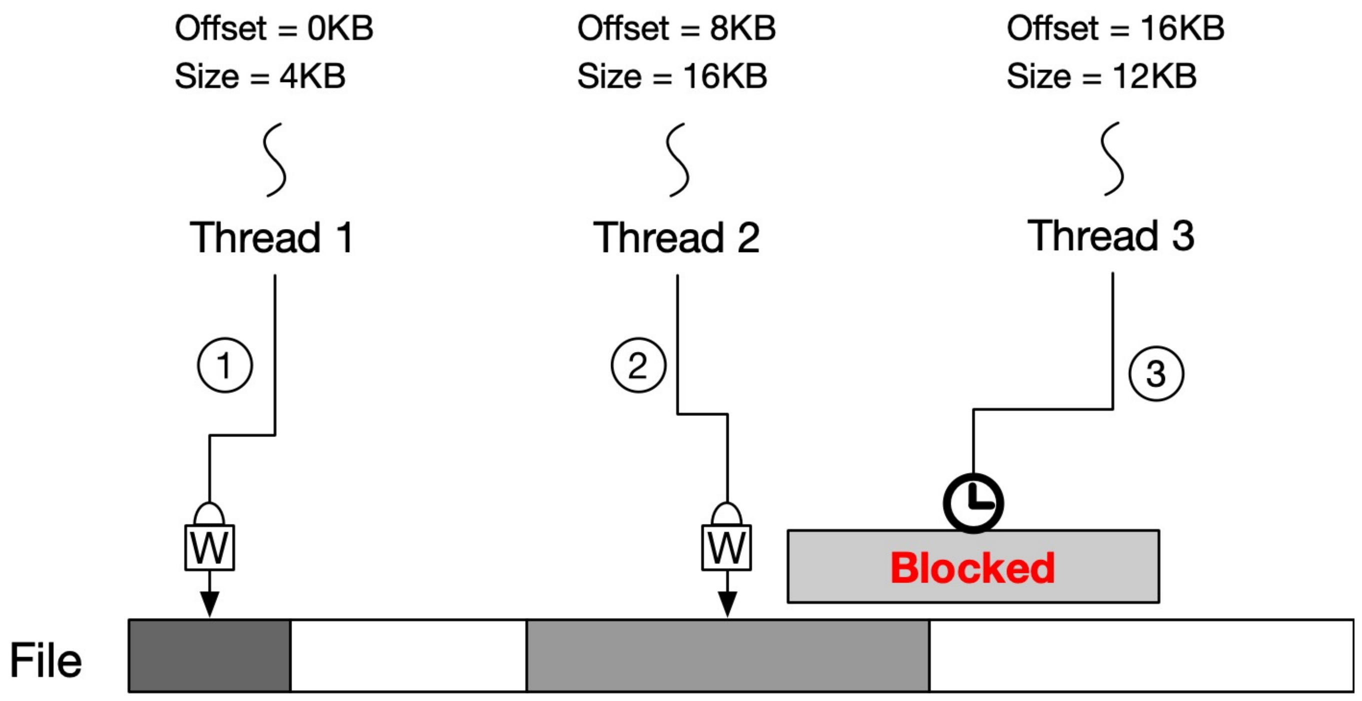
Range Lock to Enable Parallel I/O

- Consider three threads are writing the same file.
- Thread 1 and Thread 2 are writing non-overlapping ranges.
- And Thread 3 is writing overlapping range with Thread 2.



Range Lock to Enable Parallel I/O

- Consider three threads are writing the same file.
- Thread 1 and Thread 2 are writing non-overlapping ranges.
- And Thread 3 is writing overlapping range with Thread 2.



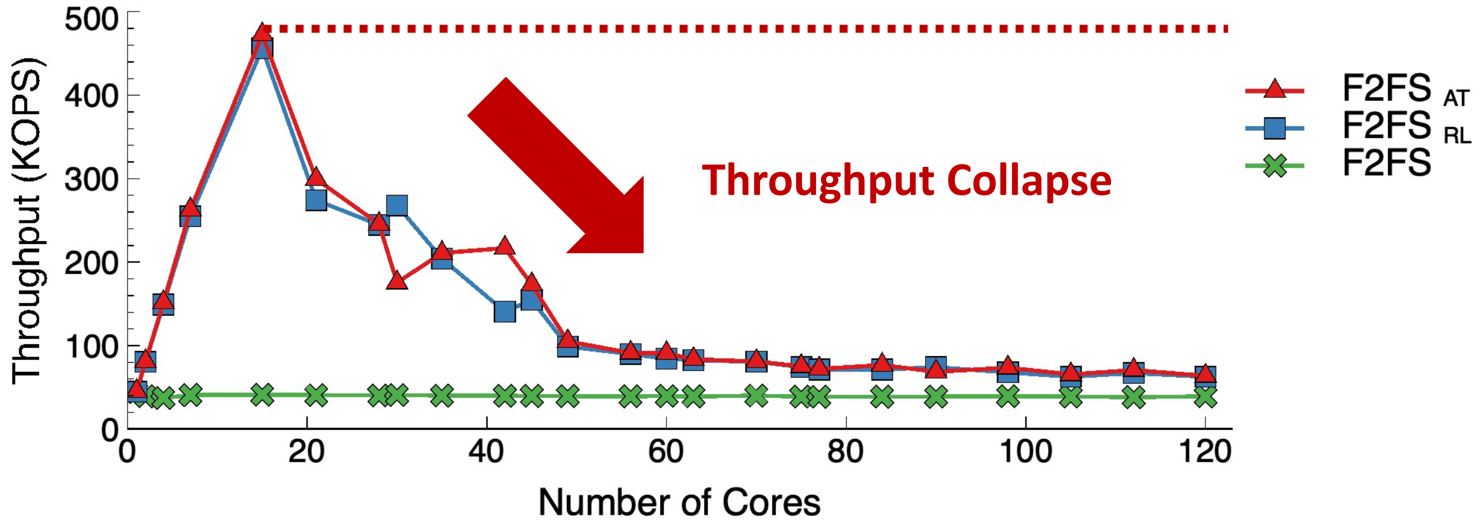
Throughput still Collapses After 15 Cores.

- We tested two range-lock implementations that have different locking overheads.
  - Interval Tree-based(**F2FS$_{RL}$**) and atomic operation-based(**F2FS$_{AT}$**)
- Interval Tree requires the tree-level lock to secure consistency against tree modification.
- In the atomic operation-based approach, the file is divided into fixed-length segments. Thus, it does not lock the entire file but only locks the corresponding segment.

< Samsung EVO 970 >

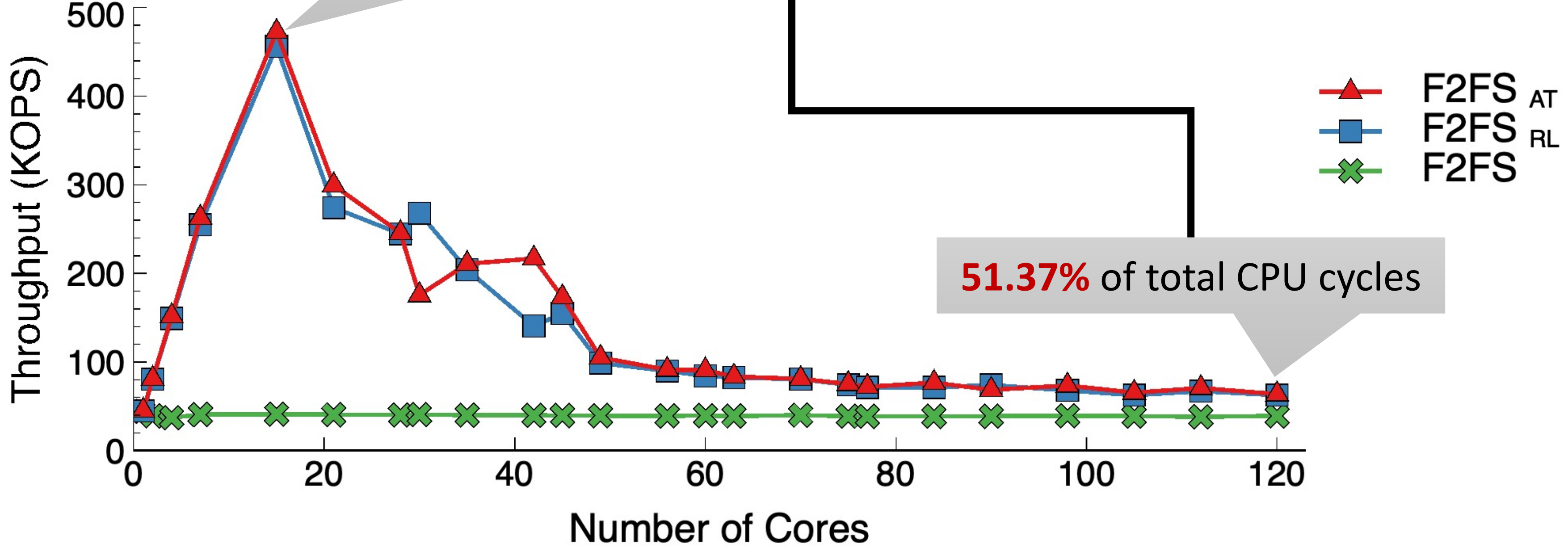
DirectIO

Shared File I/O (DWOM in FxMark)



Problem: Lack of Concurrency in File Metadata

14.54% of total CPU cycles — **Used for searching and modifying the file metadata**



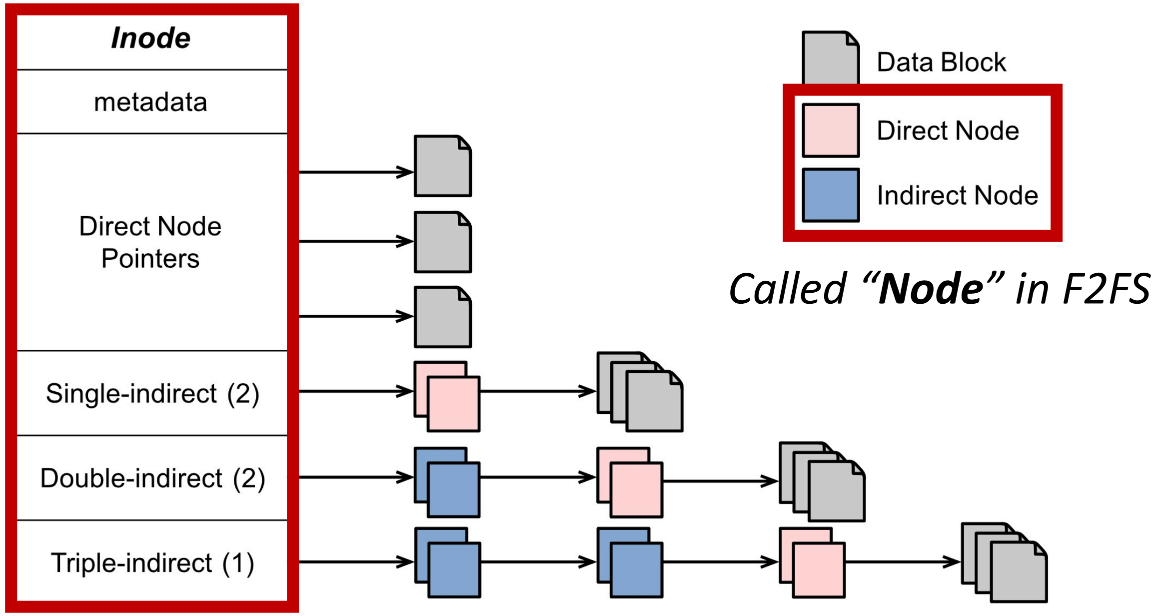
51.37% of total CPU cycles

- Searching block addresses of file data from the file metadata become the bottleneck.

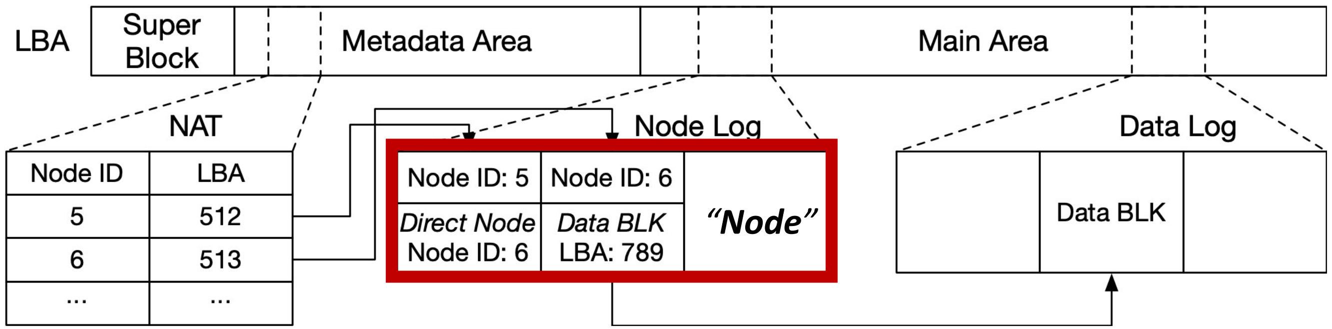
<u>Lack of concurrency in the file metadata</u>

File Metadata in F2FS

- Inode, direct node, and indirect node are called ***Node*** in F2FS
- F2FS stores Node and Data in a log fashion



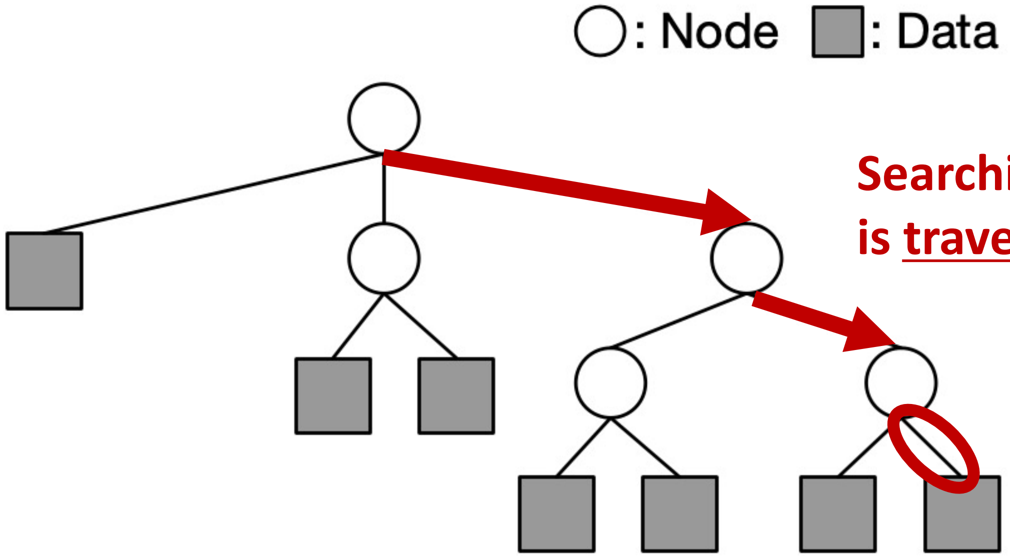
File metadata



F2FS On-disk structure

File Metadata is the Node Tree

- Since Nodes are aligned to the page size in F2FS, blocks that store Nodes are loaded into the page cache.
- When Nodes are in the page cache, the file metadata is simply a tree consist of Nodes.
- We call it **Node Tree**

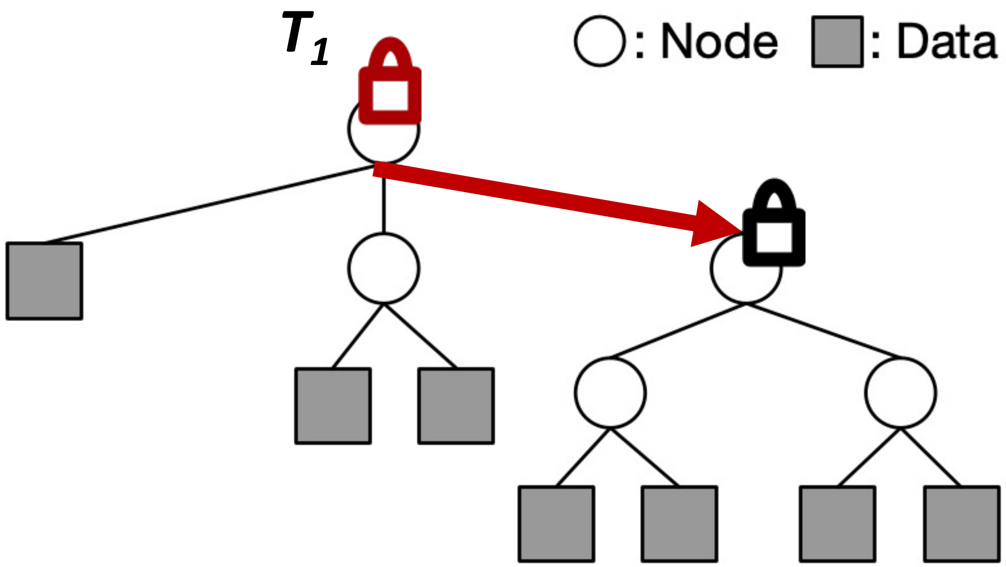


Searching for block addresses of file data is traversing the Node Tree

< Node Tree >

What is happening to node tree in Parallel I/O?

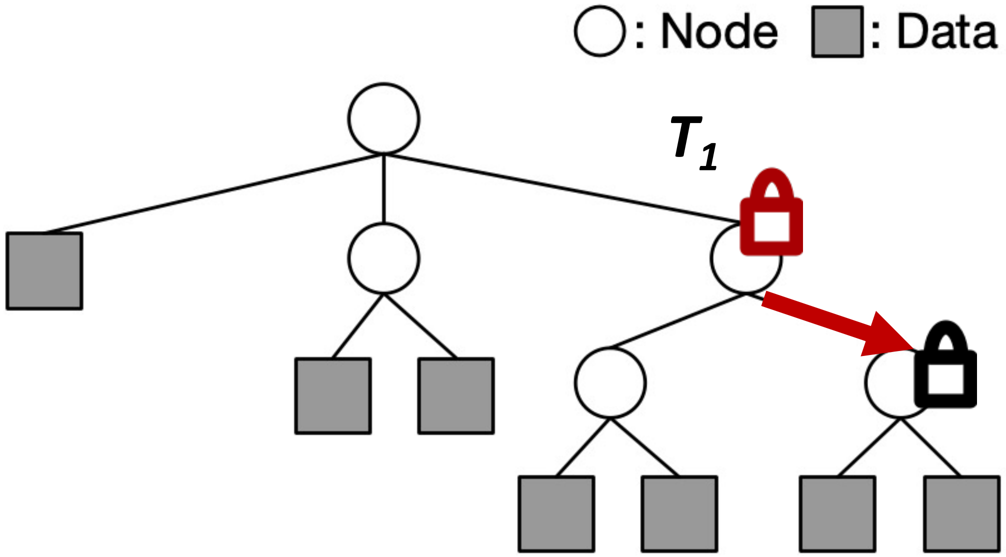
- While traversing the Node Tree, F2FS uses Mutex lock on Node for consistency.
- It only release the lock only when Mutex lock is acquired for its child Node.



< Node Tree >

What is happening to node tree in Parallel I/O?

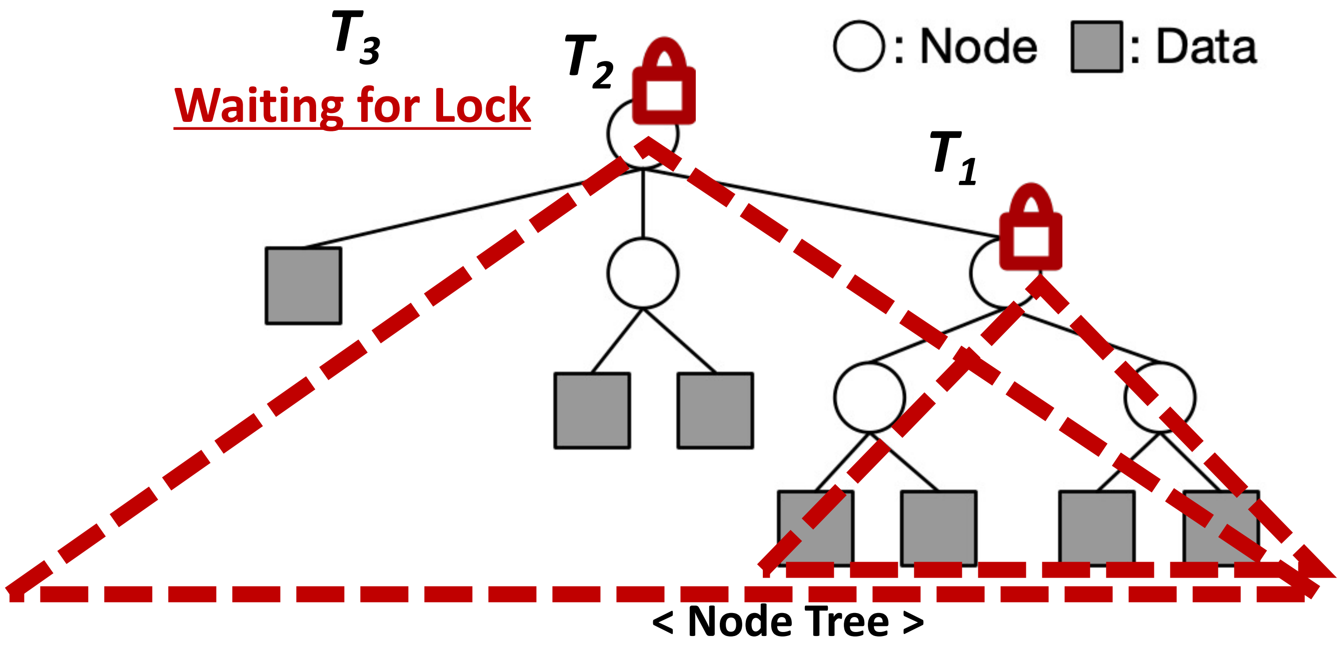
- While traversing the Node Tree, F2FS uses Mutex lock on Node for consistency.
- It only release the lock only when Mutex lock is acquired for its child Node.



< Node Tree >

Problem – Cascading Tree Lock

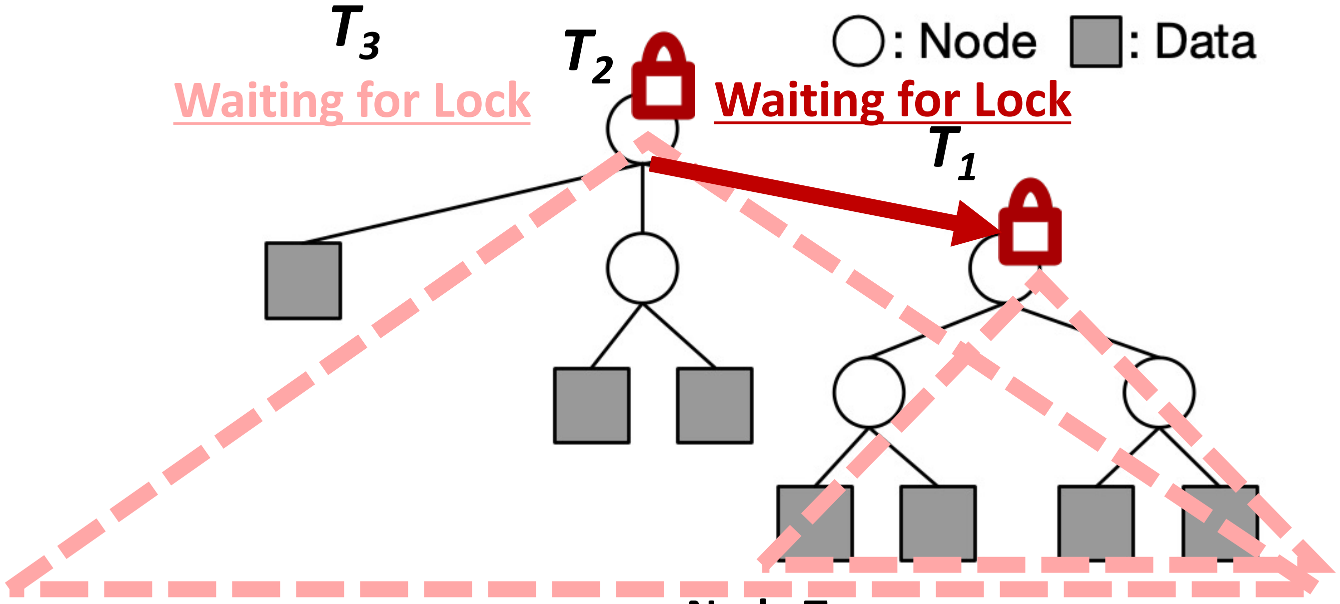
- Mutex lock on a Node blocks any other thread to enter its subtree regardless of read or write.
- As closer to the root node, a larger subtree will be blocked.



Threads are serialized at the Node even though two thread are reading disjoint ranges of file.

Problem – Cascading Tree Lock

- Mutex lock on a Node blocks any other thread to enter its subtree regardless of read or write.
- As closer to the root node, a larger subtree will be blocked.



< Node Tree >

Threads are serialized at the Node even though two thread are reading disjoint ranges of file.

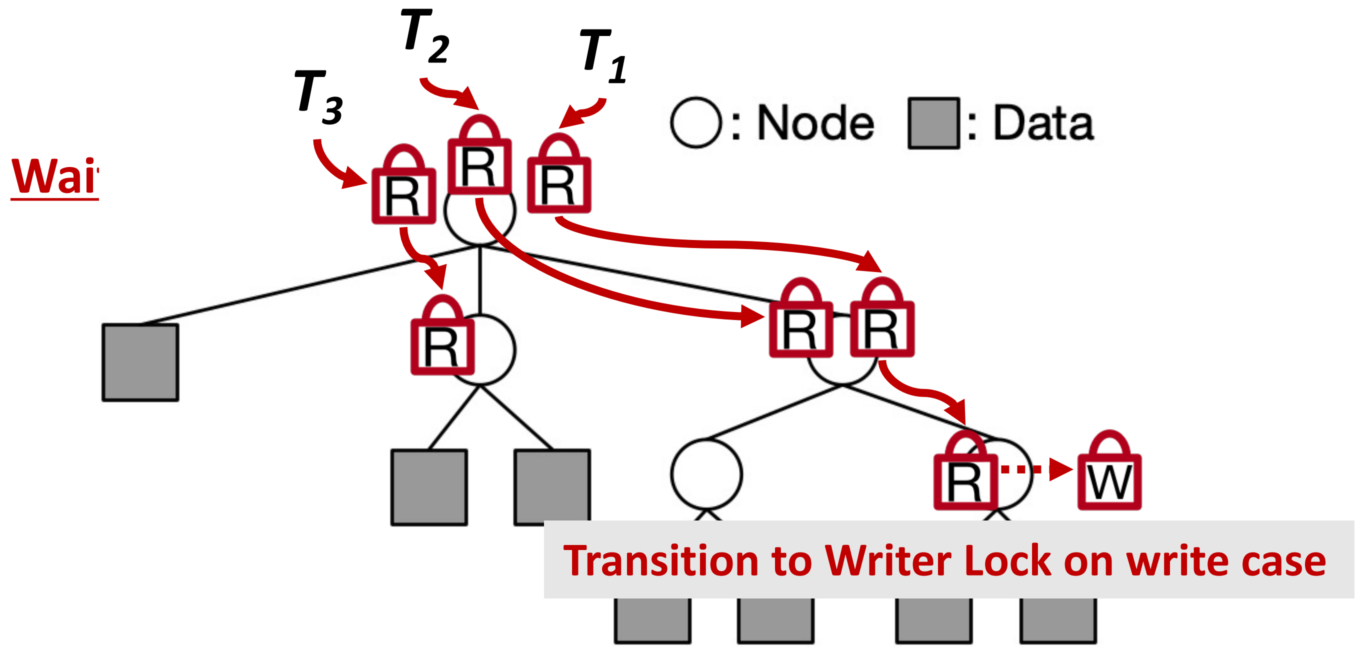
Problem Summary

- Problems
  1. Lack of Concurrency in File Metadata
  2. Cascading Tree Lock

- To solve these problems, we propose ***nCache***.

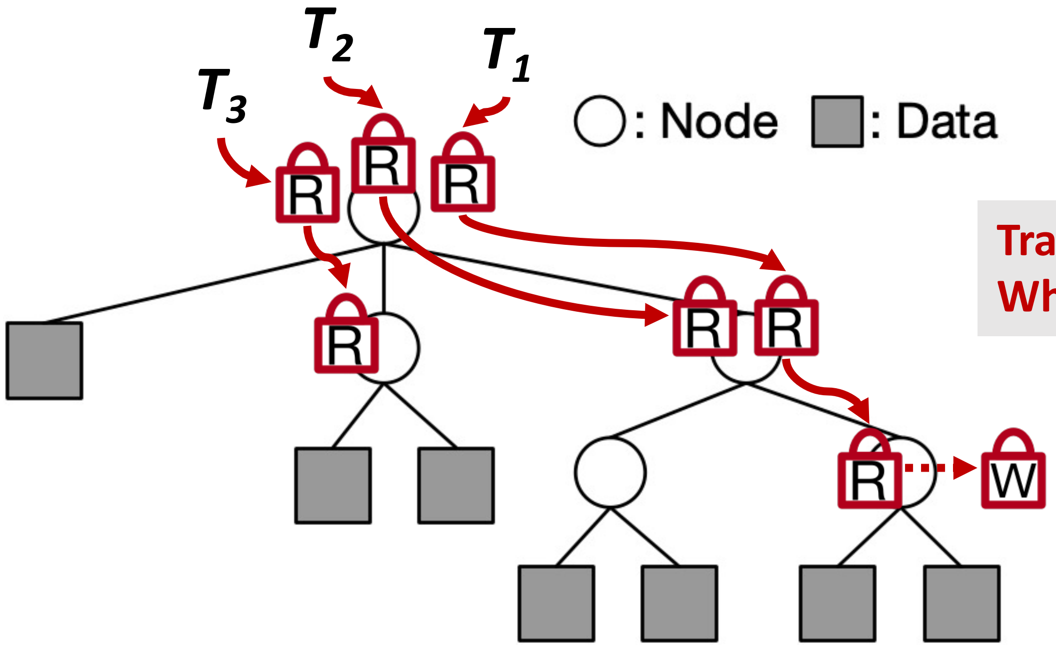
nCache Overview

- ***nCache*** employs *Readers-Writer Lock* to enable parallel accesses to *<u>Node Tree</u>*
  - Allow readers to share the subtree while traversing the Node Tree.
  - Block other threads only required subtree on a write case.



< Node Tree with nCache >

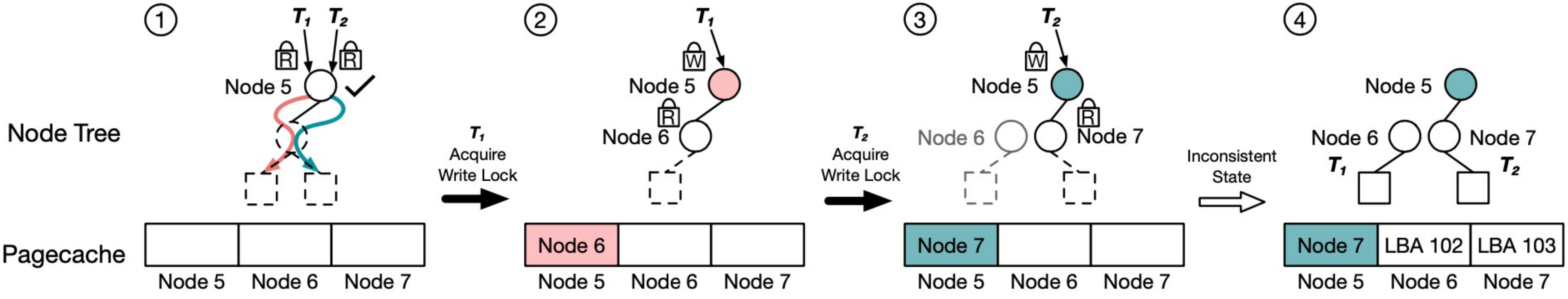
Example of *nCache*



Transition to Writer Lock
When the thread need to modify the Node

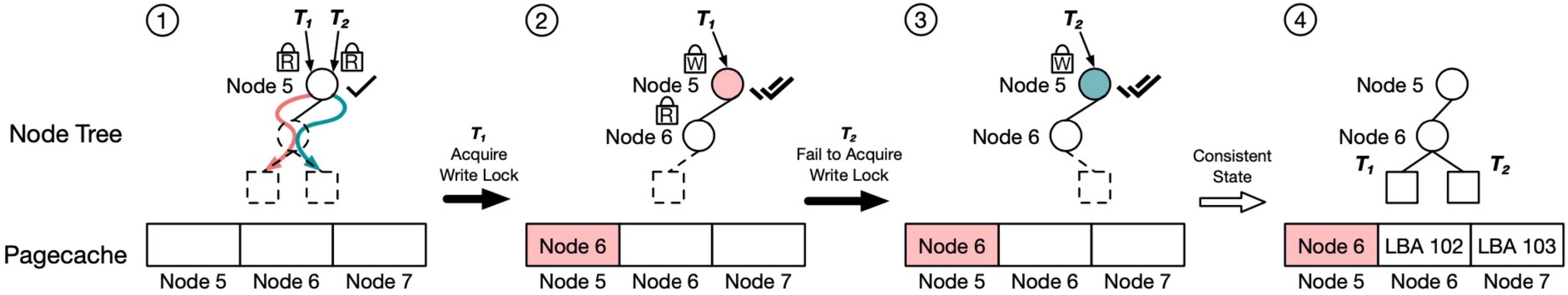
Consistency Problem

- Consider two writer threads sharing subtree.
- Both threads are trying to add a new node to the tree but different extent.
- In this case, simply adopting Readers-Writer Lock results in an inaccessible Node in the tree.



Double-checked Locking

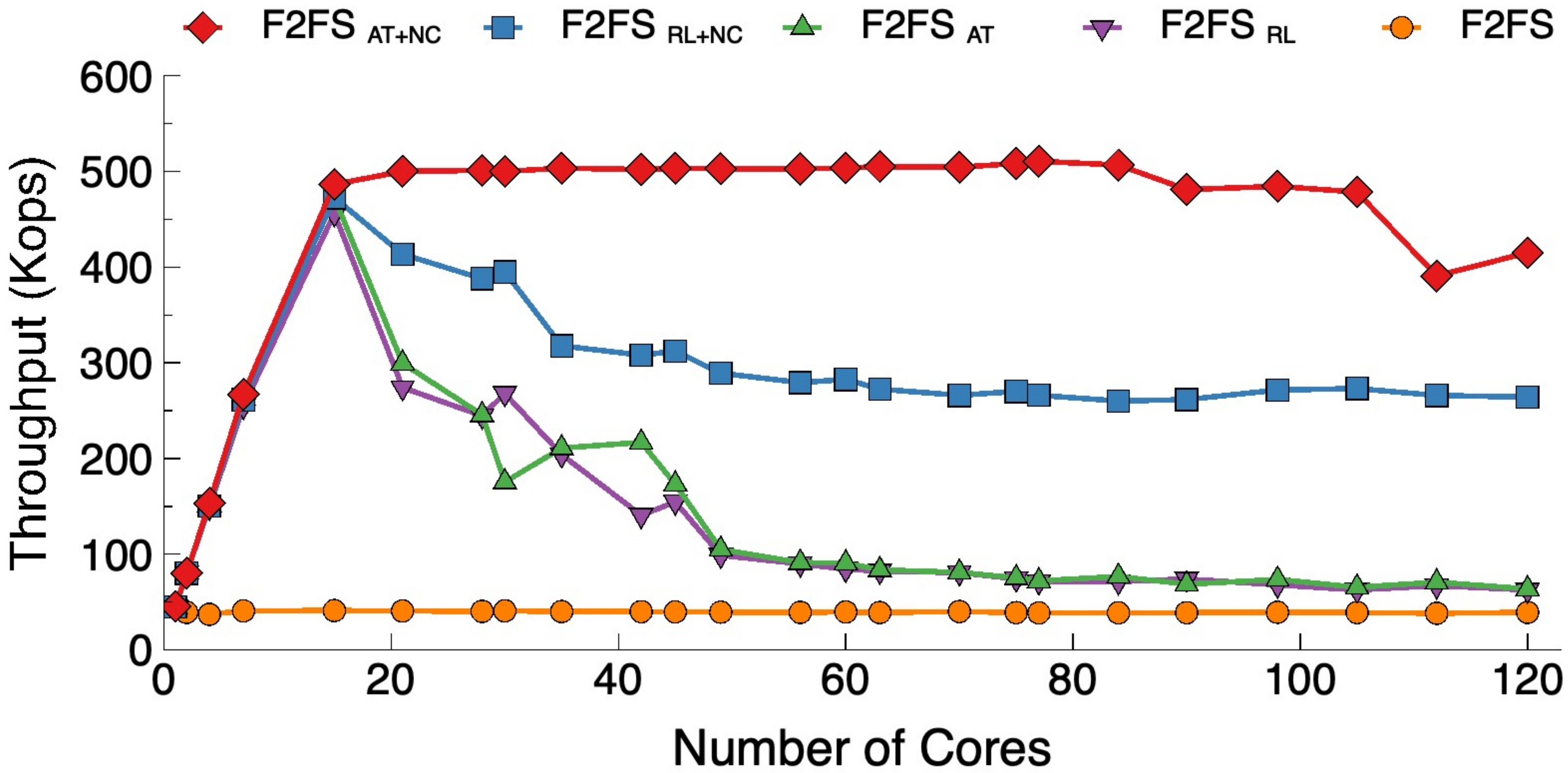
- To solve this, nCache employed double-checked locking.
- nCache releases the reader lock and re-acquires the writer lock when the thread notices it needs Node modification.
- So when a thread acquired the writer lock, it double-checks the condition because the previous writer thread might change the Node.



Evaluation Setup

- IBM 120 Core Machine with 3 different NVMe SSDs
  - Samsung 970 EVO
  - Intel 750 SSD
  - Intel Optane 900P
- Workloads
  - Synthetic Workload (FxMark) – Shared File Write (DWOM) and Shared File Read (DRBM)
  - Realistic Workload
    - HACC-IO : I/O Benchmark for Scientific Simulation Framework
    - RocksDB : LSM-based Key-Value Store
- Configurations
  - F2FS : The baseline F2FS
  - $F2FS_{RL}$: F2FS with the Interval Tree-based Range Lock
  - $F2FS_{AT}$: F2FS with Atomic operation-based Range Lock
  - $F2FS_{RL+NC}$: $F2FS_{RL}$ with nCache
  - $F2fs_{AT+NC}$: $F2FS_{AT}$ with nCache

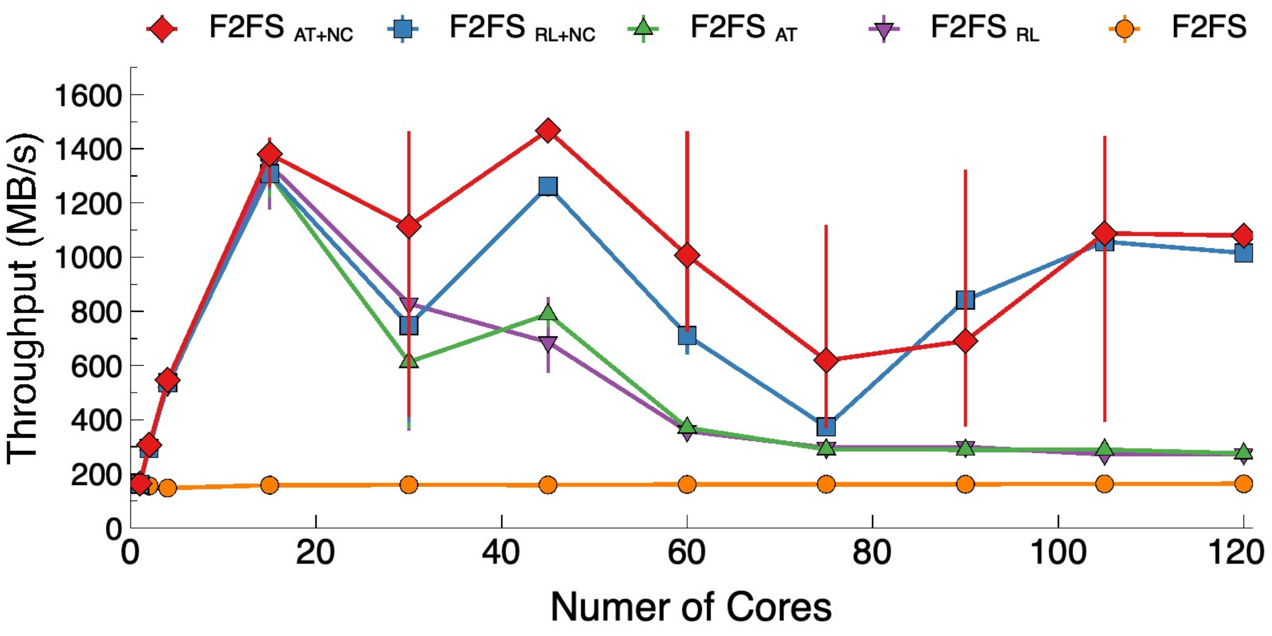
Evaluation - DWOM (Samsung 970 EVO SSD)



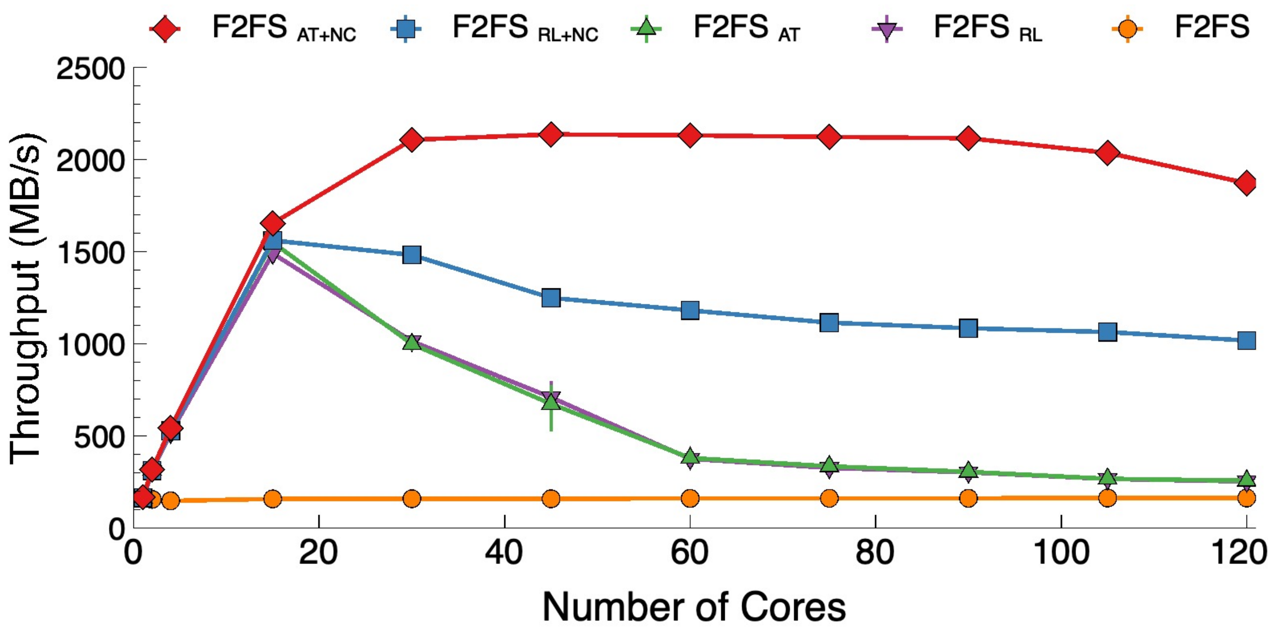
- In FxMark DWOM Workload, each thread writes a private region on a shared file.
- Both of range lock design has improved the manycore scalability.

서강대학교 SOGANG UNIVERSITY

Evaluation – HACC-IO



< Samsung 970 EVO SSD >

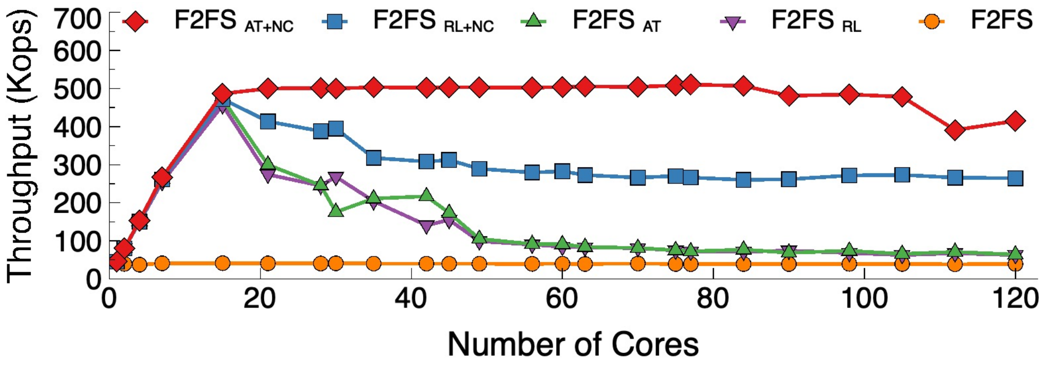


< Intel Optane 900P >

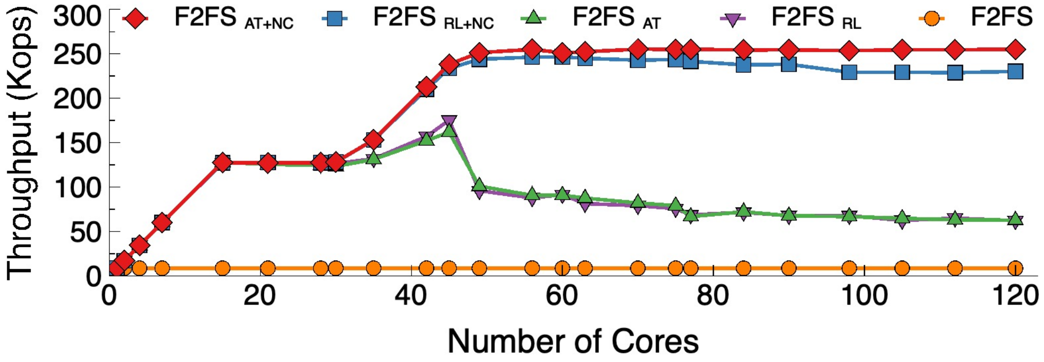
- HACC-IO emulates the checkpoint phase of HACC which is a large cosmological simulation framework for HPC

Evaluation - DWOM (Device comparison)

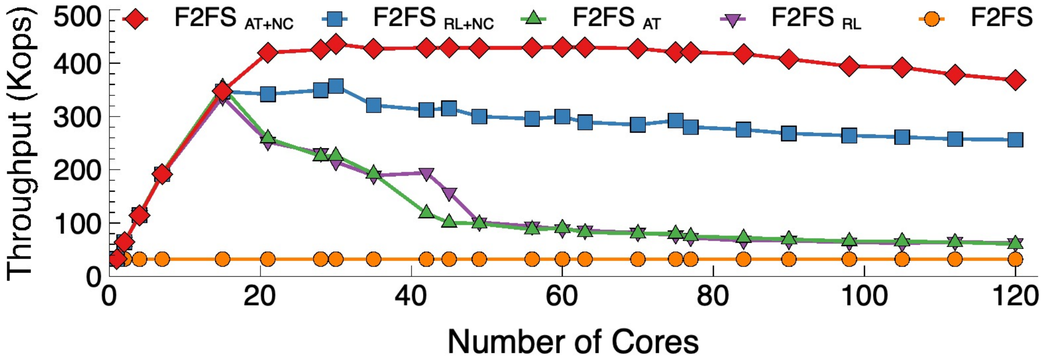
< Samsung EVO SSD >



< Intel 750 SSD >

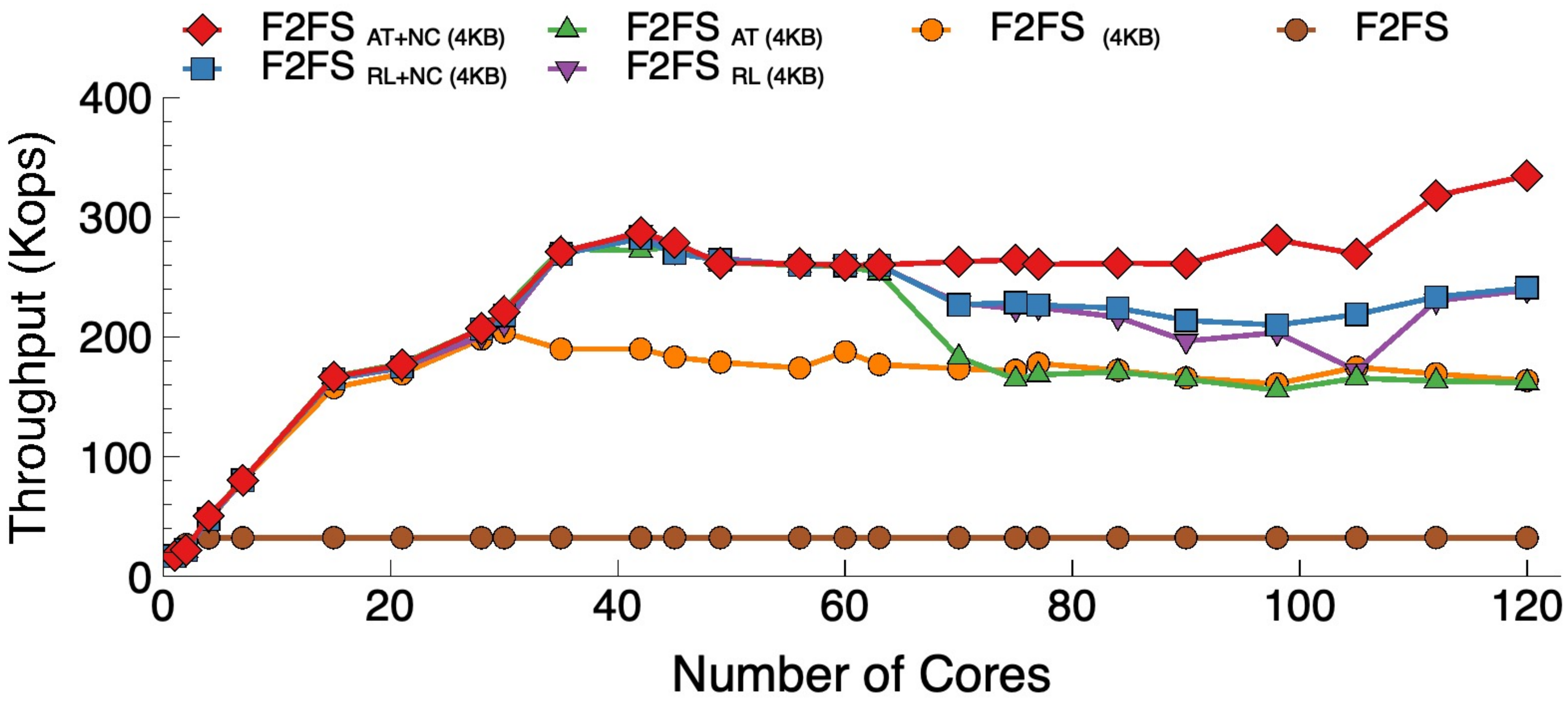


< Intel Optane 900P >



서강대학교 SOGANG UNIVERSITY

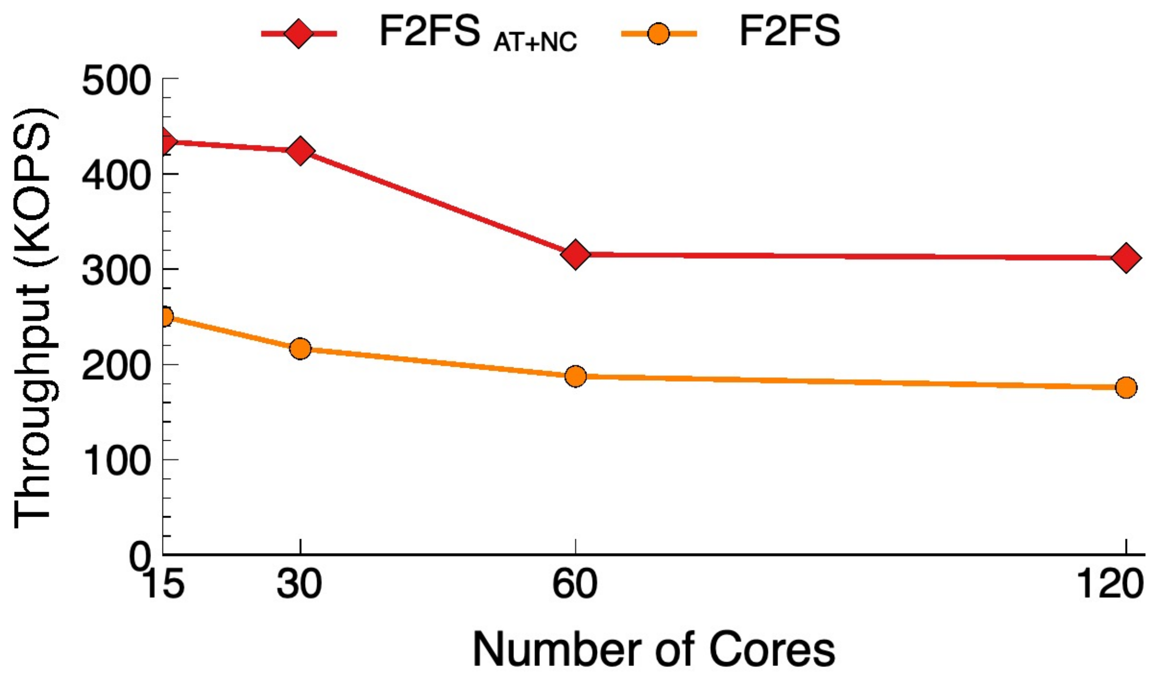
Evaluation - DRBM (stride: 4KB vs 8MB)

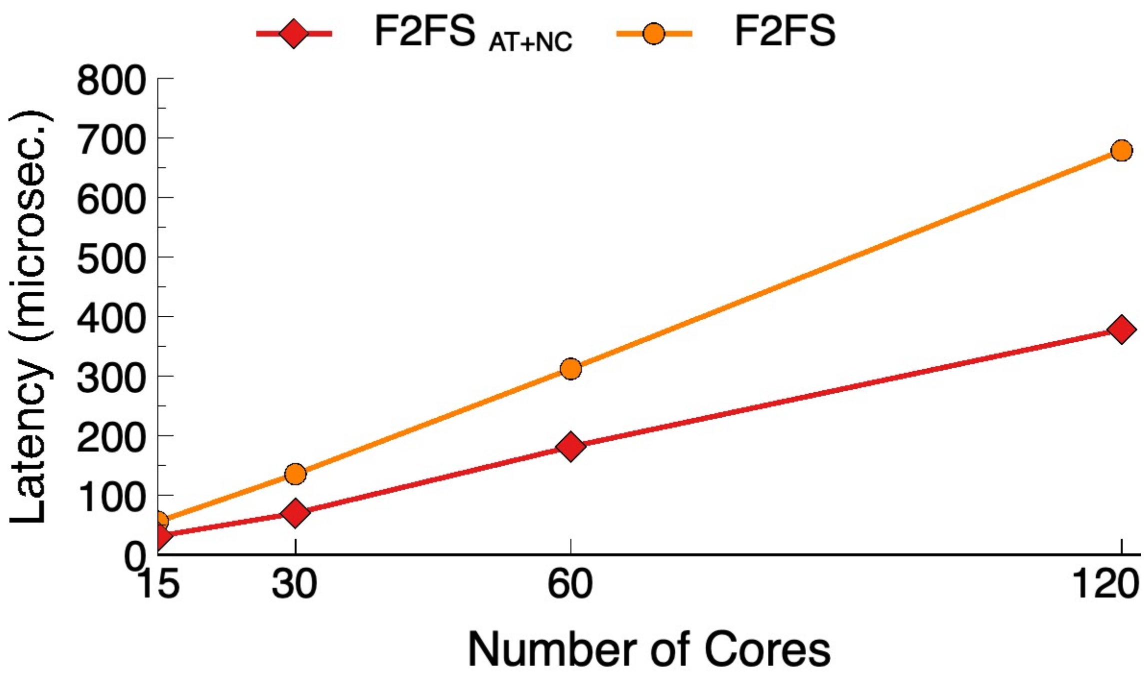


< Samsung EVO SSD >

- Device max throughput changes as IO pattern. (Refer the paper for detail)
- 8MB Stride issues more random IO to the device which leads lower device max throughput
- On the other hand, 4KB Stride make more sequential IO. As a result, the device shows higher max throughput.

Evaluation – RocksDB





- Tested via *db_bench* in RocsDB, varying number of issue thread bounded to each CPU.
- Random Read Random Write workload with 16B key and 100B Value.
- With Intel Optane 900P, nCache outperformed the baseline F2FS.
- However, the performance does not increase as the number of core increases. We see this is because of the contention in RocksDB, since LSM-tree has high compaction overhead and serialization at the memory table.

Conclusion

- Parallel I/O throughput in the manycore system had improved with the Range Lock.
- However, Range Lock solely cannot sustain the throughput till hundreds of cores.
- The main cause of the scalability bottleneck is the lack of concurrency in the file metadata structure.
- The tree data structure of file metadata has to allow concurrent accesses with considering the consistent updates to mitigate this.
- ***nCache*** enabled parallel accesses to the tree with consistency via readers-writer lock and double-checked locking.

Thanks!