CLUE : Compact and Lazy UpdatE를 통한 F2FS에서 `Unlink` 처리 최적화

황순, 이창규, 김영재
서강대학교 컴퓨터공학과
{hs950826, changgyu, youkim}@sogang.ac.kr

CLUE: Optimizing `Unlink` Operation in F2FS by Compact and Lazy UpdatE

Soon Hwang, Chang-Gyu Lee, Youngjae Kim
Department of Computer Science and Engineering, Sogang University, Seoul, Republic of Korea

요 약

매니코어 서버는 수백개의 코어를 가지고 있어 이전 보다 많은 Thread들을 동원해 병렬 I/O Throughput을 크게 높일 것으로 기대 된다. 그러나 실제 Linux 파일 시스템들은 일관성 유지를 위한 Lock의 사용으로 인해 동원된 코어 수 증가에 따른 확장성 (Scalability)있는 성능 향상을 보여주지 않는다. 기존 연구들은 매니코어 서버 환경에서 파일 데이터에 대한 병렬 I/O 성능을 높이는 방안을 제시하였으나 메타데이터 I/O의 Scalability에 대해서는 연구가 부족하다. 본 논문에서는 Linux 파일 시스템인 F2FS에서 메타데이터 I/O 중 파일 삭제인 `unlink`의 Scalability를 개선하고자 한다. F2FS의 `unlink`은 파일 시스템 범위의 Mutex Lock과 Coarse-grained Critical Section으로 인해 병렬성이 크게 떨어진다. 이는 F2FS의 Free NID를 관리하기 위한 것으로 아직 회수하지 않은 Free NID의 Scan Latency가 높아짐에 따라 `unlink`의 Blocking time도 길어지기 때문이다. 이를 해결 하기 위하여 본 논문에서는 *CLUE*를 제안한다. *CLUE*는 Free NID를 찾기위한 Search Space의 중복을 제거하여 효율 적인 Scan을 수행하고 찾은 Free NID의 Lazy Update를 통해 Blocking Time를 줄인다. *CLUE*는 기존 F2FS에 비해 개별 디렉토리에서의 `unlink`는 약 15배, 공유 디렉토리에서의 *unlink*는 약 3배의 성능 향상을 보였다.

1. 서 론

매니코어 서버는 수십에서 수백개에 달하는 코어를 가지고 있어 단일 서버에서도 높은 병렬처리 성능을 기대할 수 있다. I/O 병렬화는 I/O-intensive Application의 성능 가속화의 기본적인 기법으로 매니코어 서버는 더 많은 Thread들을 병렬 I/O에 동원할 수 있다. 그러나 최근 연구는 매니코어 서버에서 더 많은 Thread를 병렬 I/O에 할당하여도 전체 I/O Throughput이 반드시 이에 비례하여 증가하지 못함을 보여주고 있다 [1]. 이는 Linux 파일 시스템들이 병렬 I/O를 효과적으로 처리하지 못해 발생하는 문제로 코어 증가에 따른 성능 확장성 (Scalability)를 획득하기 위해서는 매니코어 환경을 고려한 파일 시스템 설계 및 최적화가 필수적임을 시사한다.

매니코어 환경에서 파일 시스템의 Scalability 병목은 크게 두 가지 측면으로 분석된다. 첫째는 파일 데이터에 대한 병렬 I/O의 성능 병목이다. Linux 파일 시스템 들은 대체로 파일 단위의 Mutex Lock을 사용하여 해당 Lock이 단일 파일에 대한 병렬 I/O의 성능 병목이 된다. 기존 연구들은 파일 단위 Mutex Lock으로 인한 병렬 I/O 성능 저하를 개선하기 위하여 범위 기반 Lock(Range Lock)을 이용해 파일의 겹치지 않는 범위에 대한 병렬 I/O를 가능하게 하여 매니코어 환경에서의 Scalability를 확보하였다 [2, 3].

두번째 측면은 파일 메타데이터에 대한 병렬 I/O의 성능 병목이다. 대다수의 Application들은 작은 파일들을 주로 생성하여 파일 시스템은 수 많은 작은 파일들로 채워진다. 이에 따른 메타데이터 I/O의 증가로 인해 전체 성능 병목이 발생한다 [4]. 분산 파일 시스템에서는 메타데이터 I/O 성능 병목 개선을 위하여 각 서버별 Local Copy를 두고 Lazy Synchronization을 통하여 중앙 메타데이터 서버의 병목을 해결하였다 [4, 5]. 그러나 매니코어 서버 환경에서 Linux 파일 시스템들은 메타데이터 I/O에 대해 주로 파일 및 디렉토리 단위의 Lock을 사용하고 그 구현 또한 VFS (Virtual File System) 레이어과 밀접하게 연동하여 성능 개선이 쉽지않다 [1].

메타데이터 I/O의 중 Lock으로 인한 확장성 저하의 예로 파일 삭제를 위한 `unlink`이 있다. `unlink`은 VFS 레이어에서 해당 파일의 부모 디렉토리에 Mutex Lock을 획득한다. 이는 해당 디렉토리에 대한 병렬 `unlink`을 차단한다. 그 후 파일 시스템 고유의 `unlink`가 실행되고 이에 수반되는 Mutex Lock등으로 인해 병렬 `unlink`의 Scalability는 더욱 저해된다. 본 연구에서는 매니코어 서버와 맞물려 높은 병렬성을 제공해 줄 수 있는 NVMe SSD와 이에 최적화된 F2FS를 통해 병렬 `unlink`을 분석하고 병렬 메타데이터 I/O의 Scalability 병목을 해결하고자 한다.

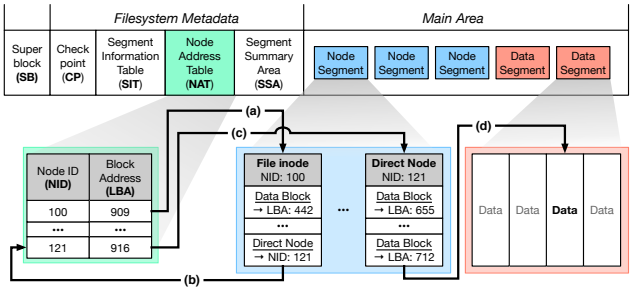


그림 1: F2FS On-disk Layout.

F2FS의 `unlink`은 *Free NID Scan*을 포함한다. NID는 F2FS에서 inode 등 파일 메타데이터에 부여하는 ID로 전체 Free NID 중 일부만을 List로 관리한다. *Free NID Scan*은 파일 생성으로 소모된 Free NID를 List에 보충하는 작업으로 일부 NID의 validity를 저장하는 Free NID Bitmap과 모든 NID의 상태를 저장하는 On-disk NAT (Node Address Table)을 확인하여 Free NID를 List에 추가한다. 이를 통해 삭제된 파일의 NID를 다시 사용할 수 있다. 그러나 *Free NID Scan*은 파일 시스템 범위의 Mutex Lock을 사용하고 Free NID Bitmap과 NAT에 대한 Scan을 하나의 Coarse-grained Critical Section으로 보호해 같은 디렉토리와 다른 디렉토리의 `unlink` 모두 직렬화 되고 매니코어 환경에서의 Scalability를 저해한다.

본 논문에서는 F2FS에서 `unlink`에 수반되는 Free NID Scan의 파일 시스템 범위의 Mutex Lock과 Coarse-grained Critical Section으로 인한 Blocking Time을 최소화 하여 병렬 `unlink`의 성능을 개선하고자 *CLUE* (Compact and Lazy UpdatE)를 제안한다. *CLUE*는 다음과 같은 세 가지 기술을 포함한다. 첫째, 최대 하나의 코어만 *Free NID Scan*을 수행하는 Optimistic Free NID Scan. 둘째, 최근 Free NID Scan 정보를 이용해 Scan length를 단축하는 Heuristic NID Bitmap Scan. 마지막으로 On-disk NAT Scan의 지연을 통한 Lazy Update이다. *CLUE*는 NVMe SSD를 장착한 120 Core 서버에서 개별 및 공유 디렉토리에서의 `unlink`에 대해

이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No. 2014-3-00035, 매니코어 기반 초고성능 스케일러블 OS 기초연구 (차세대 OS 기초연구센터)).

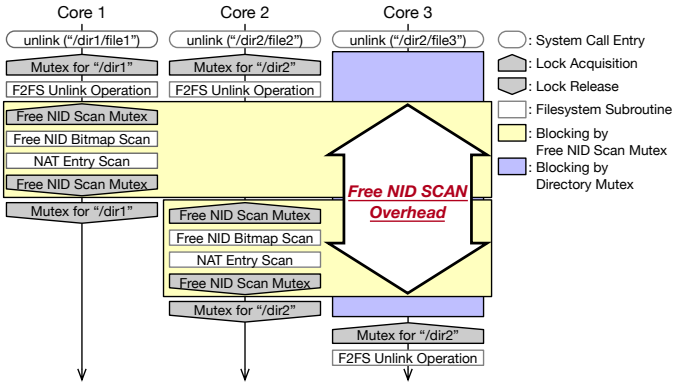


그림 2: Filesystem-wide blocking of *unlink*.

각각 기존 F2FS보다 약 15배, 3배 성능 향상을 보였다.

2. 배경 지식

2.1 F2FS

그림 1는 F2FS의 On-disk Layout을 보여준다. F2FS의 On-disk Layout은 크게 Superblock과 파일 시스템 메타데이터를 위한 Filesystem Metadata 영역 그리고 inode 등 파일 메타데이터와 파일 데이터를 저장하는 Main Area로 나뉜다. F2FS는 inode와 Indirect node, Direct node 등의 메타데이터를 모두 *Node*로써 관리하며 모든 *Node*는 Uniqueg한 NID (Node ID)를 부여받고 LBA (Logical Block Address)가 아닌 NID로 참조된다. 파일 데이터는 모두 *Data*로써 관리되며 *Node*와 *Data*는 Main Area에 Log로 쓰여진다. Log는 Segment로 불리는 고정 크기 영역 단위로 관리된다. Filesystem Metadata에는 CP, SIT, NAT 그리고 SSA를 포함한다. 이들 중 NAT는 NID로 참조되는 *Node*의 LBA 주소를 저장한다. 그림 1은 NID와 NAT를 통한 파일 데이터의 LBA를 찾는 과정을 보여준다. 우선 읽고자 하는 파일의 NID (100)를 사용해 NAT에서 LBA (909)를 얻고 *Node*를 읽는다 (그림 1(a)). 데이터 Offset을 통해 Direct Node를 특정하고 해당 NID (121)사용해 NAT로부터 LBA (916)를 획득한다 (그림 1(b)). 다시한번 NID에 해당하는 *Node*를 읽고 (그림 1(c)) Data Block의 LBA (712)를 획득한다 (그림 1(d)).

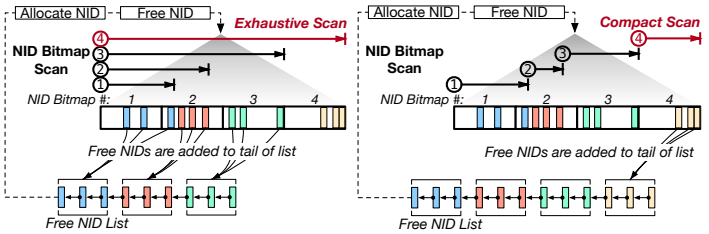
2.2 *Free NID Scan*

F2FS는 파일 생성 및 쓰기 중 *Node*를 생성할 때 부여할 NID를 Free NID List에서 할당받는다. Free NID List는 전체 가용한 NID 중 일부만을 저장하고 남은 Free NID가 Threshold보다 적을때 *Free NID Scan*을 통해 List를 보충한다. *Free NID Scan*은 두 단계로 이루어진다. 첫번째로 Free NID Bitmap을 확인하여 Free NID를 확보하는 *Free NID Bitmap Scan*이다. 이 Bitmap 또한 전체 NID를 추적하지 않아 Bitmap을 통해 Threshold 이상의 Free NID를 확보하지 못하는 경우 두번째 단계로 이행된다. 두번째 단계는 On-disk Layout의 NAT를 확인하는 *NAT Entry Scan*이다. *NAT Entry Scan*는 NAT의 맵핑 정보 중 유효하지 않은 LBA를 가진 NID를 Free NID List에 추가하며 NAT를 저장하는 Block이 Pagecache에 존재하지 않을 경우 I/O가 발생할 수 있다. *Free NID Scan*은 파일 시스템 일관성을 위하여 전체 파일 시스템 범위의 Mutex Lock사용하며 두 단계 모두 하나의 Critical Section으로 보호된다.

3. Design

3.1 Filesystem-wide Blocking by *Free NID Scan*

그림 2는 3개 코어가 `unlink`를 병렬로 처리할때 발생하는 Blocking Time을 보여준다. Core 1과 Core 2는 서로 다른 디렉토리에서 `unlink`를 수행하며 Core 2와 Core 3는 공유 디렉토리에서 수행한다. `unlink`는 VFS 레이어에서 부모 디렉토리의 Mutex Lock 획득을 강제한다. 이에 따라 서로 다른 디렉토리를 사용하는 Core 1과 Core 2는 Mutex를 획득하지만 Core 3는 Core 2의 완료까지 block된다. 부모 디렉토리의 Mutex Lock 획득 후 Core 1와 Core 2는 F2FS-specific `unlink` operation을 수행한다. 그러나 Free NID List의 Free NID 수가 Threshold 이하인지와 관계 없이 *Free NID Scan*을 위한 파일 시스템 범위의 Mutex Lock 획득을 시도한다. 그림 2과 같이 Core 1이 *Free NID Scan* Mutex Lock을 획득하여 Core 2는 Core 1의 *Free NID Scan* 완료까지 block된다. 다음으로 Core 2의 `unlink` 완료 후 마침내 Core 3가 `unlink`를 수행한다. 이 과정에서 `unlink`의 Blocking Time을 높이고 병렬성을 저해하는 요소는 두가지가 존재한다. 첫째는 VFS 레이어에서 강제하는 부모 디렉토리에 대한 Mutex Lock이다. 이는 원천적으로 같은 디렉토리에서의 `unlink` 병렬 수행을 막아 Scalability를 저해한다. 두번째는 *Free NID Scan*으로 인한 파일 시스템 범위의 Mutex Lock이다. 이는 그림 2에서 볼 수 있듯 *Free NID Scan* Mutex가 직접적으로 Core 2의 `unlink` Blocking Time을 증가시킨다. 그러나 두 가지 중 더 큰 영향력을 가지는 요소는 파일 시스템 범위의 Mutex으로 인한 전체 `unlink`의 Blocking이다. 이는 디렉토리의 공유 여부와 관계 없이 모든 `unlink`의 병렬 수행을 막고 따라서 Scalability를 저해한다. 그 예로 Core 3가 부모 디렉토리로 인해 발생하는 Blocking Time은 Core 1과 Core 2 모두의 *Free NID Scan* 수행 시간을 포함한다. 본 논문에서는 공유 및 개별 디렉토리에서의 `unlink` 모두에 영향을 끼치는 공통 요소인 *Free NID Scan*의 수행시간을 줄임으로서 Blocking Time을 최소화 하고 따라서 Scalability 개선하기 위한 *CLUE*를 제안한다.



(a) Exhaustive Scan in current F2FS. (b) Heuristic Bitmap Scan.

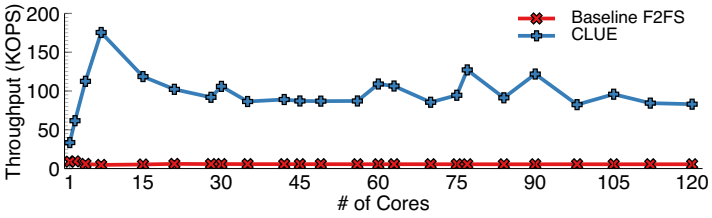
그림 3: Scan length reduction using Heuristic Bitmap Scan.

3.2 Optimistic *Free NID Scan*

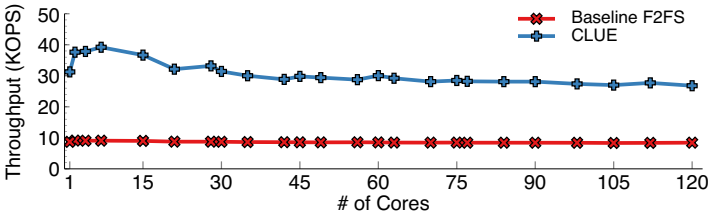
*Free NID Scan*은 *Free NID Bitmap Scan*와 *NAT Entry Scan*를 하나의 Mutex Lock으로 보호한다. 이러한 Coarse-grained Critical Section은 *Free NID Bitmap Scan*와 *NAT Entry Scan*간의 Interleave를 막고 더욱이 *NAT Entry Scan*의 잠재적 Disk I/O로 인해 Blocking Time을 대폭 증가시킬 수 있다. 이러한 문제를 해결하기 위하여 *CLUE*는 *Free NID Scan*의 두 단계를 각기 다른 Mutex Lock으로 보호하여 Critical Section 크기를 줄여 병렬성을 확보하였다. 더불어 *CLUE*는 이미 *Free NID Bitmap Scan*를 수행중인 코어가 있다면 바로 *NAT Entry Scan*로 이행하도록 하였다. 이는 실행중인 *Free NID Bitmap Scan*이 Threshold 이상의 Free NID를 확보할 것으로 기대하고 그럼에도 부족한 Free NID는 *NAT Entry Scan*에서 보충하도록 하여 불필요한 *Free NID Bitmap Scan* 진입을 줄이고 두 단계가 병렬적으로 실행 될 수 있도록 한다.

3.3 Heuristic *Free NID Bitmap Scan*

*Free NID Bitmap Scan*은 Free NID Bitmap을 처음 부터 순회하며 Free NID를 확보한다. 찾아낸 Free NID는 Free NID List의 Tail에 차례로 추가 되고 NID는 List Head에서부터 할당된다. 이후 Free된 NID는 Free NID Bitmap에 다시 셋팅된다. 그림 3a는 *Free NID Bitmap Scan*의 수행을 보여준다. 4번의 *Free NID*



(a) MWUL 성능



(b) MWUM 성능

그림 4: Scalability comparison between *CLUE* and Baseline F2FS

*Bitmap Scan*은 모두 첫번째 Bitmap 부터 탐색을 시작한다. 따라서 ④는 ①, ②, ③이 Scan한 범위를 중복 Scan한다. 연속된 *Free NID Bitmap Scan*에서 이전 Scan이 찾은 Free NID는 Free NID List의 Tail에 위치한다. 따라서 이전 Scan의 구간에서 찾아낸 Free NID는 이미 List에 존재하는 Free NID 이후에 할당되고 Free 될 것이다. 또한 이전 Scan에서 이미 할당되었던 NID는 현재 Scan과 이전 Scan 사이에 발생한 `unlink`의 수에 비례해 Free 될 것이다.

*CLUE*는 높은 `unlink` Rate 상황에서는 앞선 Scan 구간의 Free NID가 빠르게 할당되고 Free 될 확률은 높지만 그만큼 연이은 두 *Free NID Bitmap Scan*의 간격이 짧아 이전 Scan 구간 내의 Free NID들은 이미 List에 있을 것으로 기대한다. 더불어 낮은 `unlink` Rate 상황에서는 이전 Scan 구간 내의 이미 할당된 NID가 Free 될 확률이 낮을 것으로 기대한다. 따라서 *CLUE*는 직전 *Free NID Bitmap Scan*이 끝난 지점부터 Scan을 시작하는 Heuristic *Free NID Bitmap Scan*을 제안한다. 그림 3b와 같이 *CLUE*는 ①에서 Scan한 구간을 ②, ③, ④가 차례로 이어서 Scan하여 중복된 구간을 Scan하지 않는다. 이로써 기존 F2FS보다 짧은 Scan 구간을 탐색하여 *Free NID Bitmap Scan*으로 인한 Blocking Time을 최소화 한다.

3.4 Lazy Free NID List Update

*CLUE*는 Optimistic *Free NID Scan*과 Heuristic *Free NID Bitmap Scan*을 통한 성능 향상에도 불구하고 여전히 *NAT Entry Scan*로 인한 Blocking Time이 존재한다. 이는 뒤따라올 *Free NID Bitmap Scan*을 통해 충분히 Threshold 이상의 Free NID를 확보 할 수 있었음에도 불구하고 현재 실행중인 *Free NID Bitmap Scan*로 인해 *NAT Entry Scan*이 수행되어 NAT Entry에 대한 I/O 수행해야 하기 때문이다. 따라서 *CLUE*에서는 *Free NID Bitmap Scan*이 Threshold 이상의 Free NID 확보에 실패하고 *Node* 할당이 발생하여 즉시 Free NID가 필요한 시점까지 *NAT Entry Scan*를 지연 시킨다. 이러한 Lazy Free NID List Update인해 *Node* 할당의 직렬화를 회피하기 위하여 *CLUE*는 F2FS에서 주기적으로 실행되는 Checkpoint에서 *NAT Entry Scan*를 수행한다.

4. 실험 및 분석

4.1 실험 환경

본 실험은 120 Core (8 Socket) 매니코어 서버에서 740GB의 메모리와 Samsung EVO 970 250GB NVMe SSD를 사용해 수행하였으며 Linux 5.7 F2FS에 *CLUE*를 구현하였다. 실험은 FxMark [1]의 MWUL, MWUM 워크로드를 사용하였다. MWUL는 각 Core가 개별 디렉토리에서 각자 `unlink`를 수행하고 MWUM는 모든 Core가 공유 디렉토리에서 `unlink`를 수행한다.

4.2 개별 디렉토리에서의 `unlink` Scalability

그림 4b는 MWUL 실험 결과이다. 기존 F2FS는 *Free NID Scan*의 파일 시스템 범위 Mutex로 인해 `unlink`의 병렬성이 저해되어 Scalability를 전혀 보이지 못한다. 반면 *CLUE*는 *Free NID Scan*의 Blocking Time을 최소화 하고 *Free NID Scan*와 *NAT Entry Scan* 간의 동시 수행을 가능케 하여 15 Core까지 Scalability을 보인다. 그러나 15 Core 이후 성능이 하락하여 120 Core 까지 유지한다. 이는 *Free NID Scan*의 파일 시스템 범위의 Mutex Lock을 근본적으로 제거한 것이 아닌 Blocking Time의 최소화만을 달성 했기 때문으로 보인다. 따라서 Mutex Lock이 매니코어 환경에서 갖는 성능 한계와 함께 Free NID Bitmap, NAT Entry I/O 그리고 Free NID List의 NUMA (Non-Uniform Memory Access) 환경에서의 Remote Access로 인한 성능 한계가 나타난 것으로 보인다.

4.3 공유 디렉토리에서의 `unlink` Scalability

그림 4a는 MWUM 실험 결과이다. 기존 F2FS는 *Free NID Scan*의 파일 시스템 범위의 Mutex로인한 Blocking Time의 증가와 비효율적인 *Free NID Bitmap Scan*으로 인해 매니코어 환경에서 Scalability를 보여주지 못한다. *CLUE*의 경우 *Free NID Scan*에서 Blocking Time의 최소화로 인해 기존 F2FS에 비해 약 3배 높은 Throughput을 보여준다. 그러나 개별 디렉토리상에서의 `unlink`와 달리 코어 수 증가에 따른 뚜렷한 성능 Scalability를 보이지 않는다. 그 이유는 여러 코어들이 동시에 공유 디렉토리에 대해 `unlink`를 수행할 때 디렉토리에 대한 Mutex가 VFS 레이어에서 강제되어 모든 `unlink`들이 직렬화 되고 *Free NID Scan*와 *NAT Entry Scan*간의 동시 수행이 일어나지 않기 때문이다. 분석 결과 15 Core와 120 Core에서 VFS 레이어 상에서 공유 디렉토리에 대한 Mutex Lock이 전체 CPU cycle 중 91%를 소모하여 디렉토리의 Mutex Lock으로 인한 Scalability 병목임을 확인하였다.

5. 결론

본 논문에서는 매니코어 환경에서 F2FS의 병렬 `unlink`시 성능과 Scalability을 저해하는 원인을 밝혔다. 이는 *Free NID Scan*에서 파일 시스템 범위의 Mutex Lock과 Coarse-grained Critical Section으로 인한 Blocking이고 이를 해결하기 위해서 *CLUE*를 적용하였다. *CLUE*를 적용한 F2FS에서 MWUL에서 약 15배, MWUM에서 약 3배의 성능 향상을 보였다.

참고 문헌

[1] C. Min, S. Kashyap, S. Maass, and T. Kim, "Understanding Manycore Scalability of File Systems," in *Proceedings of the USENIX Conference on Usenix Annual Technical Conference (ATC)*, pp. 71–85, 2016.

[2] C.-G. Lee, H. Byun, S. Noh, H. Kang, and Y. Kim, "Write optimization of log-structured flash file system for parallel i/o on manycore servers," SYSTOR '19, 2019.

[3] J.-H. Kim, J. Kim, H. Kang, C.-G. Lee, S. Park, and Y. Kim, "Pnova: Optimizing shared file i/o operations of nvm file system on manycore servers," in *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '19, 2019.

[4] Q. Zheng, K. Ren, and G. Gibson, "Batchfs: Scaling the file system control plane with client-funded metadata servers," in *2014 9th Parallel Data Storage Workshop*, pp. 1–6, 2014.

[5] K. Ren, Q. Zheng, S. Patil, and G. Gibson, "Indexfs: Scaling file system metadata performance with stateless caching and bulk insertion," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 237–248, IEEE, 2014.