비휘발성 메모리 파일시스템의 단일 파일에서의 쓰기 병렬화

김장웅O, 김준형, 김영재, 박성용
서강대학교 컴퓨터공학과
{ski6812, whekf1234, youkim, parksy}@sogang.ac.kr

Parallelizing a Single File Write Operation of Non-Volatile Memory File Systems

Jangwoong KimO, Junehyung Kim, Youngjae Kim, Sungyong Park
Department of Computer Science and Engineering
Sogang University, Seoul, Republic of Korea

요 약

비휘발성 메모리는 빠른 지연시간, 높은 가격 당 용량, 바이트 단위의 접근성으로 인하여 파일을 저장하는 매체로써 사용될 수 있다. 하지만, 파일시스템 계층에서의 파일에 대한 획일적인 락은 병렬적 쓰기를 제한하는 요소로 작용하고 있다. [1]. 이러한 제약은 멀티코어 환경을 고려해 I/O를 병렬화하는 최근 고성능 데이터베이스 애플리케이션들의 성능을 제한하는 요소가 된다. 따라서 본 논문은 한 파일에 동시에 쓸 수 있도록 범위 기반 락을 적용하는 방법을 소개한다. 특히, 비휘발성 메모리 파일시스템 중 최신 연구인 NOVA에 범위 기반 락을 구현하여 파일시스템 확장성을 테스트 할 수 있는 벤치마크인 FxMark를 이용하여 성능을 측정하였다. 제안한 방식은 기존 NOVA 대비 최대 170%의 성능 향상을 보였다.

1 서 론

최근 서버가 100개 이상의 많은 코어를 갖게 되고, 각각의 코어가 많은 I/O를 발생시킴으로써 멀티/매니코어 환경에서의 고성능 I/O에 대한 관심이 높아지고 있다. 특히 고성능 데이터베이스나 키-밸류 애플리케이션들의 I/O는 멀티코어 환경을 염두하여 설계되기 시작하였다.

한편, 인텔의 3D-Xpoint, ReRAM (Resistive RAM), PRAM (Phase change RAM) [2] 등의 비휘발성 메모리 (NVM, Non-Volatile Memory)는 메모리 버스에 장착되어 바이트 단위로 접근이 가능하며 DRAM과 비슷한 읽기/쓰기 지연시간을 보인다. 따라서 이러한 빠른 비휘발성 메모리는 최근 요구되기 시작하는 고성능 I/O를 위한 스토리지 매체로 사용 될 수 있으며, 이에 대한 연구들이 활발히 진행되고 있다.

비휘발성 메모리 기반의 파일시스템들은 여러 가지 요인들로 인하여 SSD와 HDD같은 기존의 블락 기반 장치들보다 높은 성능을 보인다. 우선 초당 메가바이트의 스루풋을 보이는 블락 기반 장치에 비하여, 비휘발성 메모리는 초당 기가바이트의 스루풋을 보인다. 또한 기존 블락 장치 기반 파일시스템에서는 디스크 접근을 최소화 하기 위해서 데이터를 접근할 때 페이지 캐시부터 접근하여 오버헤드가 발생하지만, 대부분의 메모리 파일시스템은 페이지 캐시를 사용하지 않으므로 해당 오버헤드가 없어진다.

또한 메모리 파일시스템은 장치의 바이트 단위의 접근성을 이용하여 적은 오버헤드로 파일시스템의 일관성을 지킬 수 있다 [3]. 기존의 블락 장치 기반 파일시스템들은 장치의 읽기/쓰기 최소 단위가 4KB 등의 블락이기 때문에 로깅 [4], 저널링 [5], Copy-On-Write [6] 등의 방식으로 일관성을 보장한다. 하지만 이러한 일관성 보장 방식은 하나의 트랜잭션에 대한 일관성을 보장하기 위하여 하나 이상의 블락을 써야하므로 큰 오버헤드를 유발한다. 하지만 비휘발성 메모리 기반 파일시스템들은 메모리의 바이트 단위의 접근성을 이용하여 적은 양의 쓰기로 하나의 트랜잭션의 일관성을 보장하므로 일관성 보장의 오버헤드가 적다 [3].

하지만, 이러한 고속의 스토리지 매체의 특징에도 불구하고 여러 쓰레드들이 같은 파일에 쓰기 연산을 하는 경우 병렬적인 쓰기가 불가능하다. 이는 파일시스템 계층에서 쓰기 연산시 파일 전체에 락을 거므로 모든 쓰기가 직렬화되기 때문이다. 이렇게 파일에 걸리는 획일적인 락은 여러 쓰레드가 같은 파일에 쓰기 연산을 하는 데이터베이스같은 애플리케이션의 확장성을 저해한다.

이를 해결하기 위하여 본 논문에서는 획일적인 락이 아닌 파일 쓰기 범위에 기반한 범위 락의 이용을 제안한다. 또한 이를 비휘발성 메모리 파일시스템 중 하나인 NOVA에 적용하여 개선된 NOVA 파일시스템을 구현하고 성능을 평가한다. 파일시스템의 확장성을 평가하는 도구인 FxMark [1]를 이용하여 성능을 측정하였을때, 한 파일에 동시에 쓰는 워크로드에서 개선된 NOVA는 기존의 NOVA 대비 최대 170%의 성능 향상을 보였다.

2 NOVA 파일 시스템

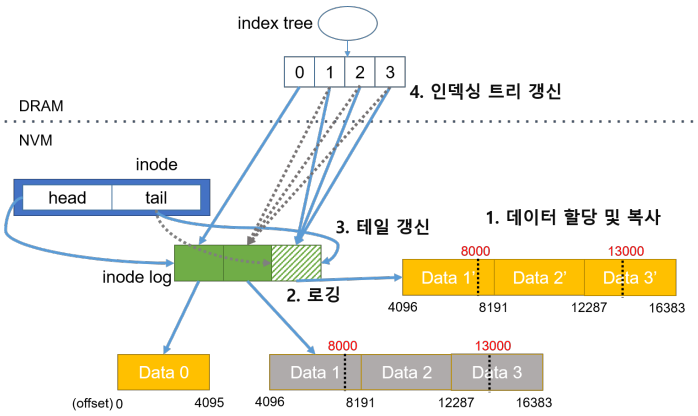


그림 1: NOVA 파일 시스템 구조 및 단일 파일에 쓰기 흐름 설명

NOVA [7] 파일시스템은 비휘발성 메모리와 DRAM을 모두 사용하는 하이브리드 메모리 기반 로깅 파일시스템이다. NOVA는 파일시스템의 일관성을 위하여 데이터에 대해서는 COW (Copy-On-Write)를, 로그에 대해서는 로그 구조를 사용한다. 그림 1은 NOVA 의 구조를 나타낸다. 쓰기 연산은 COW 방식을 이용한다. 즉 데이터를 기존의 위치에 쓰는게 아니라

새로운 곳에 쓴 후 포인터를 바꾸게 된다. 초록색의 각 로그 엔트리는 해당하는 데이터를 가르키며 순차적으로 써진다. 인덱싱을 위한 트리는 파일마다 할당되며 DRAM에 존재한다. 유저 데이터, 일관성 유지를 위한 로그, 아이노드 등은 NVM에 존재한다.

쉬운 설명을 위하여 페이지 사이즈가 4KB 라고 가정하고, 파일이 초기에 4 페이지로 구성되어 있다고 가정하자. 데이터 페이지의 번호는 각 데이터의 페이지 오프셋을 나타낸다. 그림 1을 이용하여 8000 번째 바이트를 시작으로 5000 바이트 크기의 쓰기 시스템이 요청되었을 때 NOVA의 쓰기 과정을 설명한다. 파일이 동시에 여러 쓰레드에 의해 쓰여지는 것을 방지하기 위하여 NOVA는 쓰기 연산 이전에 우선 파일 전체에 락을 요청한다. 쓰기 쓰레드가 락을 획득한 후, NOVA의 프리 블록 매니저는 3개의 데이터 페이지 (data1', data2', data3') 를 NVM으로부터 할당한다. 데이터 페이지 할당이 끝나면 유저버퍼에서 데이터를 복사 한 후, data1과 data3 의 기존 페이지의 일부를 data1'과 data3' 으로 복사한다. (1. 데이터 할당 및 복사) 이제 새로운 페이지가 바르게 구성되었으므로 해당하는 로그 엔트리를 만들고 해당 로그 엔트리가 새로운 데이터 페이지의 시작 주소를 가르키게 한다. (2. 로깅) 로그 엔트리는 쓰기 연산의 위치와 크기등 해당 쓰기 연산에 대한 정보를 담고있다. 로그 엔트리가 써지고 나면 아이노드의 테일 포인터를 새로운 로그 엔트리의 주소로 변경해 준다. (3. 테일 갱신) 테일 포인터까지 업데이트가 되고 나면 해당 쓰기는 유효한 것이 된다. 왜냐하면 해당 시점에서 전원이 차단되면 NOVA는 로그의 처음부터 테일이 가르키는 로그 엔트리까지 탐색하여 파일의 인덱스 구조를 복구할 수 있기 때문이다. 만약 테일을 업데이트 하기 전에 전원이 차단된다면 해당 로그가 복구 대상에서 제외되므로 쓰기 연산은 없던 것이 되므로 역시 일관성이 보장된다. 마지막으로 파일의 인덱스 트리의 1, 2, 3 번째 노드가 새로운 로그 엔트리를 가르키게 변경함으로써 쓰기 연산은 완료된다. (4. 인덱싱 트리 갱신)

NOVA는 쓰기 연산 시 아이노드 전체에 락을 건다. 이러한 획일적인 락은 같은 파일에 병렬적인 쓰기를 제한한다 [1]. 그림 3은 파일시스템의 확장성을 측정하는 도구인 fxmark 벤치마크를 이용하여 쓰레드의 갯수에 따른 NOVA의 I/O 출력량을 보여준다. DWOL 워크로드는 각 쓰레드가 각자의 파일에 쓰기를 수행하며 DWOM 워크로드는 쓰레드들이 같은 파일의 다른 부분에 쓰기를 수행 한다. DWOL의 경우 15 쓰레드까지 쓰기 연산이 확장성을 보이는 것을 알 수 있다. 하지만 같은 파일에 쓰기를 하는 DWOM의 경우 쓰기를 하는 쓰레드가 해당 파일 전체에 락을 걸기 때문에 여러 쓰레드가 쓰기를 진행해도 전혀 확장성이 없다.

3 범위 락 기반 NOVA 파일 시스템 설계 및 구현

본 장에서는 NOVA의 획일적인 락이 한 파일에 병렬적인 쓰기를 제한한다는 점에 착안하여 파일에 쓰는 부분에만 락을 거는 범위 기반 락이 구현된 개선된 NOVA의 구현과 이로 인한 디자인 이슈에 대해 설명한다.

Interval tree [8]는 겹치는 범위를 탐지할 수 있는 유일한 자료구조이다. 범위는 시작과 끝으로 구성되며, 각 범위는 red black 트리에 노드로 삽입되게 된다. 이 자료구조를 이용하면 어떤 범위에 대하여 이것과 겹치는 범위가 있는지 여부를 트리 탐색을 하여 알 수 있다. 이러한 interval tree를 이용하여 NOVA에서 범위 기반 락을 설계할 수 있다. 즉, 쓰기 연산 전에 NOVA는 해당 쓰기 범위와 겹치는 쓰기가 진행되어 있는지 여부를 확인하여, 만약 겹치는 쓰기가 수행되고 있다면 기존의 쓰기가 끝날 때 까지 새로운 쓰기를 대기시킨다. 만약 겹치는 쓰기가 없다면 해당 쓰기는 대기 없이 수행되며, I/O 쓰레드는 해당하는 범위에만 락을 걸어 겹치는 쓰기가 동시에 수행 되는 것을 방지한다. 이렇게 겹치는 범위를 가지는 쓰기만 대기시키기 때문에 기존의 NOVA와 달리 개선된 NOVA에서는 겹치지 않는 쓰기에 대해서는 병렬적으로 쓰기를 할 수 있다. 따라서 우리는 interval tree가 구현된 리눅스의 패치 [9]를 리눅스 커널에 추가하여 NOVA의 코드에 맞게 적용하고 기존의 파일 전체에 걸리는 락을 범위 기반 락으로 대체하여 NOVA를 개선하였다. 확장성을 위하여 범위 확인을 위한 red black 트리를 파일마다 정의하였다.

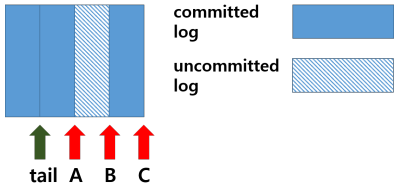


그림 2: 병렬적 로깅으로 인한 일관성 문제

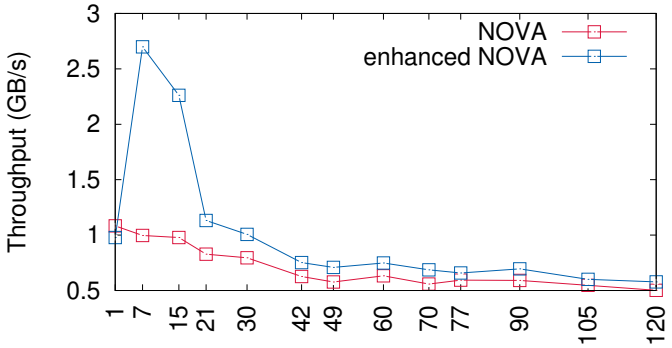
이제 NOVA에서 겹치지 않는 쓰기들은 동시에 진행되므로 로그 또한 병렬적으로 쓰여지는데, 이는 로그의 일관성을 깰 수 있다. 그림 2은 이러한 로그 일관성 문제를 보여준다. 3개의 쓰레드가 겹치지 않는 쓰기를 병렬적으로 수행한다고 가정하자. 해당 쓰레드들은 로그를 쓴 후 각각 테일을 A, B 그리고 C로 업데이트 하려하는데, 테일 업데이트의 순서가 보장되지 않는다. 따라서 만약 B로 테일을 업데이트 하는 쓰레드의 로그는 쓰여지지 않은 상태에서 C로 테일이 변경된다면 쓰여지지 않은 로그가 커밋되어 파일시스템의 일관성이 깨지게 된다. 이를 해결하기 위하여 테일을 업데이트할 때 스핀락을 이용하여 테일 업데이트를 보호한다.

4 실험결과

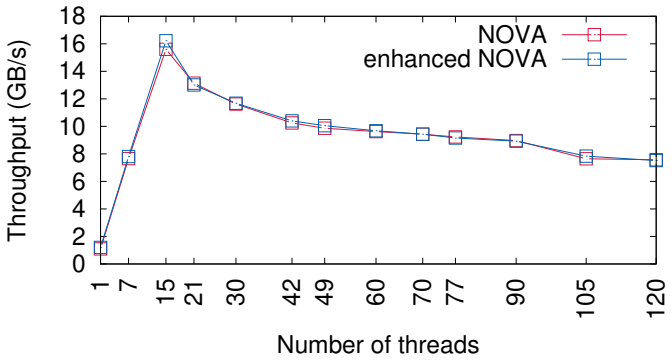
실험을 위하여 15 코어의 Intel Xeon E7-8870 v2 프로세서가 8개의 소켓으로 구성된 120 코어 머신을 사용하였다. 메모리는 768 GB의 DDR3 DRAM을 사용하였으며, 그 중 32GB를 인텔의 Persistent Memory Emulation Platform (PMEP)를 사용하여 비휘발성 메모리로 에뮬레이트 하였다. 성능은 파일시스템의 확장성을 측정하는 벤치마크 도구인 FxMark를 이용하여 측정되었다.

그림 3는 NOVA, 개선된 NOVA에 대하여 FxMark 벤치마크의 DWOM, DWOL의 실험 결과를 보여준다. 그림 3 (a)의 DWOM 워크로드의 경우 여러 쓰레드가 한 파일에 쓰기를 수행하는 N-to-1 쓰기의 실험이며 그림 3 (b)의 DWOL은 각 쓰레드가 각자의 파일에 개별적으로 쓰기를 수행하는 N-to-N 쓰기의 실험이다.

N-to-1 (DWOM) 쓰기의 경우 7 쓰레드까지 확장성을 보이며 최대 170%의 성능 개선을 보인다. 하지만 7에서 15로 쓰레드 수가 늘어날 때는 확장성이 보이지 않는데 이는 락의 오버헤드 때문이다. 그림 4는 DWOM 워크로드에 대하여 perf 툴을 사용하여 얻은 요소별 소요 시간을 타나낸다. 7에서 15쓰레드로 쓰레드 수가 늘어날 때 범위락과 스핀락에 소요하는 시간이 전체 수행 시간의 대부분을 차지한다. 즉 쓰레드의 수가 늘어나서 생기는 락에서의 병목현상 7보다 많은 쓰레드의 실험에서의 확장성을 저해한다. 또한 하나의 NUMA 켓의 코어 수인 15를 넘어가면 리모트 메모리 접근으로 인하여 성능이 급격히 저하된다.



(a) DWOM 워크로드



(b) DWOL 워크로드

그림 3: NOVA, 개선된 NOVA의 DWOL, DWOM 실험 결과

N-to-N (DWOL) 쓰기에서는 기존의 NOVA와 성능 차이가 없다. 즉, 우리가 추가한 범위락에 의한 오버헤드가 존재하지 않음을 볼 수 있는데, 이는 쓰레드들이 각자의 파일에 쓰기를 수행하므로 파일마다 존재하는 범위락을 위한 트리가 병목지점이 되지 않기 때문이다. 또한 단일 파일이 병목 지점이 되는 N-to-1 쓰기 워크로드 보다 전체적인 스루풋이 높다.

N-to-1 쓰기에서 7 쓰레드까지 확장성을 보였지만 락의 오버헤드가 그 이상의 확장성을 저해한다. 그림 4은 N-to-1 쓰기 수행 시간의 비율을 나타낸다. 21 쓰레드에서 스핀락과 범위락이 차지하는 수행 시간이 71.83%이다. 즉 쓰레드 수가 많아질 수록 락이 확장성을 제한하는 병목 지점이 된다. 특히 N-to-1 쓰기의 경우에는 쓰레드들이 범위락을 획득하려 할 때마다 공통된 범위락 트리에 연산을 하기 때문에 범위락이 병목지점이 된다.

5 결론

본 논문에서는 일관성을 보장하는 비휘발성 메모리 기반 파일시스템 중 가장 높은 성능을 보이는 NOVA 파일시스템의 구조와 일관성 보장 방식을 알아보았다. 또한 쓰기시 파일 전체에 걸리는 락을 한 파일에 병렬적 쓰기를 저해하는 요인으로 분석하였고, 이를 해결하기 위하여 쓰기 범위에 기반한 범위 락을 제안하고 성능을 평가하였다. 제안된 기법은 같은 파일에 쓰는 N-to-1 쓰기 워크로드에서 최대 170%의 성능 향상을 보였으며, 다른 파일에 쓰는 N-to-N 쓰기 워크로드에 대해서는 기존의 NOVA와 거의 동일한 성능을 보였으며 추가적인 오버헤드를 보이지 않았다.

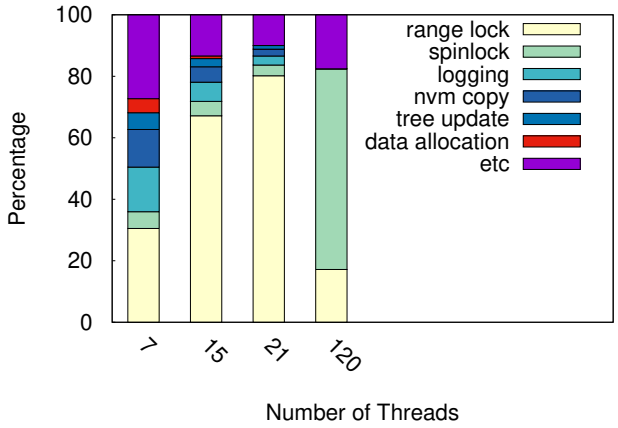


그림 4: 개선된 NOVA의 DWOM에서 작업당 소요시간 분석

6 사사문구

이 논문은 2018년도 정부 (과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2014-3-00035, 매니코어 기반 초고성능 스케일러블 OS 기초연구 (차세대 OS 기초연구센터)).

참고 문헌

[1] C. Min, S. Kashyap, S. Maass, and T. Kim, "Understanding manycore scalability of file systems.," in *USENIX Annual Technical Conference*, pp. 71–85, 2016.

[2] H.-R. Oh, B.-h. Cho, W. Y. Cho, S. Kang, B.-g. Choi, H.-j. Kim, K.-s. Kim, D.-e. Kim, C.-k. Kwak, H.-g. Byun, *et al.*, "Enhanced write performance of a 64-mb phase-change random access memory," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 122–126, 2006.

[3] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *Proceedings of the Ninth European Conference on Computer Systems*, p. 15, ACM, 2014.

[4] C. Lee, D. Sim, J. Y. Hwang, and S. Cho, "F2FS: A new file system for flash storage.," in *FAST*, pp. 273–286, 2015.

[5] M. Cao, S. Bhattacharya, and T. Ts'o, "Ext4: The next generation of ext2/3 filesystem.," in *LSF*, 2007.

[6] O. Rodeh, J. Bacik, and C. Mason, "Btrfs: The linux b-tree filesystem," *ACM Transactions on Storage (TOS)*, vol. 9, no. 3, p. 9, 2013.

[7] J. Xu and S. Swanson, "NOVA: A log-structured file system for hybrid volatile/non-volatile main memories.," in *FAST*, pp. 323–338, 2016.

[8] A. Pal and M. Pal, "Interval tree and its applications," *Advanced Modeling and Optimization*, vol. 11, no. 3, pp. 211–224, 2009.

[9] J. Kara, "Range Reader/Writer Locks for the Kernel." https://lwn.net/Articles/724502/.