F2FS에서 NVM 기반 노드 로깅을 통한 *fsync* 응답시간 향상 기법 연구

변현기°, 이창규, 노성현, 김영재
서강대학교 컴퓨터공학과
{bhyunki, changgyu, nsh0249, youkim}@sogang.ac.kr

Improving *fsync* Response Time with NVM-based Node Logging in F2FS

Hyunki Byun°, Changgyu Lee, Sunghyun Noh, Youngjae Kim
Department of Computer Science and Engineering
Sogang University, Seoul, Republic of Korea

요 약

데이터베이스는 *fsync* 시스템 콜을 이용하여 메모리에 있는 변경사항을 파일시스템을 통해 저장 장치에 보관한다. 하지만 이러한 과정은 상당한 오버헤드가 발생하며 응용프로그램의 I/O 성능을 저하시킨다. F2FS는 플래시 저장 장치에 최적화된 로그 기반 파일 시스템으로 *fsync*를 자주 호출하는 경우 체크포인팅으로 인한 오버헤드가 증가한다. 이 때, 다른 I/O 요청들을 임시 중단하게 되며 심각한 I/O 성능 저하를 발생시킨다. 본 논문에서는 비휘발성 메모리를 활용하여 F2FS에서 노드 로그를 가속화함으로써 *fsync*의 응답 속도를 개선하는 연구를 진행하였다. FxMark 벤치마크를 이용하여 실험한 결과 기존 F2FS에 비하여 *fsync*의 I/O 지연시간은 85% 감소하였고 I/O 처리율이 최대 1.7배 향상되었다.

1 서 론

클라우드(Cloud) 환경에서는 데이터를 저장하기 위하여 SQLite, MySQL, LevelDB, RocksDB 같은 데이터베이스를 사용한다. 데이터베이스는 질의를 이용하여 데이터를 관리하고 파일시스템을 통하여 데이터를 저장 장치에 보관한다. NAND Flash가 보급되면서 SSD가 주 저장 장치로 사용되고 있으며 SSD의 특성을 고려하여 설계된 F2FS [1]가 등장하였다. F2FS는 로그 기반 파일 시스템으로 쓰기 전파(Write Propagation) 문제인 Wandering Tree Problem을 해결하였다. 또한, F2FS는 SSD를 저장 장치로 사용하는 데이터베이스 환경에서 기존 파일시스템들 보다 좋은 성능을 보인다.

파일시스템은 메모리의 변경 사항들을 *fsync* 시스템콜을 통하여 저장 장치로 보관하고 일관성을 지킨다. 대부분의 파일시스템들은 일관성을 보장하기 위하여 저널링(Journaling), COW(Copy-on-Write) 또는 로그 기반 구조(Log-Strucutred) 방식들을 채택한다. 하지만 이러한 방식들은 *fsync*가 호출되었을 때 단순히 데이터만 저장하는 것이 아니라 파일시스템 내부 구조에 따라서 추가적인 I/O를 발생시킨다.

F2FS는 *fsync*가 불렸을 때 파일의 데이터 뿐만 아니라 파일의 Inode와 파일시스템 메타데이터를 저장하기 위해 추가적인 I/O를 수행한다. 또한 세그먼트 클리닝과 체크포인팅(Checkpointing)을 위한 추가적인 I/O를 생성한다. 세그먼트 클리닝은 로그 기반 파일시스템에서 새로운 로그 영역을 만들기 위한 과정으로 반드시 체크포인팅을 수반한다. 체크포인팅은 메모리에 존재하는 파일시스템 메타데이터들을 SSD에 저장하여 일관성을 보장하는 과정이다. 특히 체크포인팅은 파일시스템의 모든 I/O 요청을 블록킹(Blocking)하기 때문에 전체적인 성능에 치명적이다.

위와 같은 추가적인 I/O와 체크포인팅으로 인한 I/O 블록킹 문제를 개선하기 위하여 본 논문에서는 비휘발성 메모리(NVM)를 이용한 노드(Node) 로깅 기법을 제안한다. NVM 노드 로깅 기법은 데이터 로그는 SSD에 저장하며 노드 로그와 파일시스템 메타데이터는 NVM에 저장한다. 이를 통하여 체크포인팅 시간을 줄이고 I/O 블록킹 시간도 최소화할 수 있다. 제안된 기법을 FxMark 벤치마크를 이용하여 실험한 결과, 기존 F2FS에 비하여 *fsync*의 지연시간은 85% 감소하였고 I/O 처리량은 최대 1.7배 향상되었다.

2 관련 연구

저널링 파일 시스템에서 *fsync*로 인한 오버헤드를 개선하기 위해서 아래와 같은 연구들이 진행되었다. Fine grained metadata journaling 기법 [2]은 저널링 파일 시스템에서 NVM을 저널 영역으로 활용하는 방식을 제안하였다. NVM의 Byte-addressable 특성을 이용한 Fine-grained 저널을 수행함으로써 쓰기 증폭 문제를 해결하였다. iJournaling [3]은 Ext4에서의 컴파운드(Compound) 트랜잭션(Transaction)으로 발생하는 성능 저하를 해결하기 위해 파일 단위의 트랜잭션을 제안하였다. 하지만 파일 단위의 트랜잭션은 여러 코어에서 동시에 *fsync*를 하는 상황에서 처리해야 할 트랜잭션의 수를 증가시키는 문제가 있다. Tinca [4]는 저널링 파일 시스템에서 NVM을 블록 캐시(Block Cache)로 사용하는 방식을 제안하였다. Tinca는 커밋(Commit)과 체크포인트에서 발생하는 이중쓰기 문제를 해결하였으며 링 버퍼(Ring Buffer)를 이용하여 블록 캐시 수준에서의 트랜잭션(Transaction)을 구현하였다.

이외에도 SSD 장치 수준에서 *fsync*를 개선하는 연구도 진행되었다. RFLUSH [5]는 Flush 명령을 수행할 때 SSD 내부 메모리에 존재하는 모든 데이터를 플래시에 저장하는 *lump-sum* 문제를 해결하기 위하여 새로운 RFLUSH 명령어를 제안하였다. RFLUSH 명령어는 SSD 내부 버퍼(Buffer)에서 특정 파일만 플래시에 저장하여 필요하지 않은 Flush를 줄일 수 있다.

앞서 제시한 연구들은 파일시스템 저널로 인한 *fsync* 오버헤드를 개선하거나 SSD 수준에서의 최적화를 연구하였다. 본 연구는 로그 기반 파일시스템에서 *fsync*를 개선하기 위한 연구로서 NVM을 활용하여 노드 로깅과 체크포인팅을 가속화하는 연구이다.

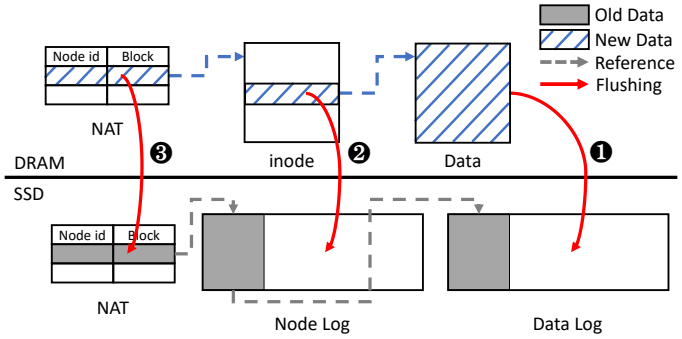

그림 1: F2FS의 On-disk 자료 구조와 Flush 과정

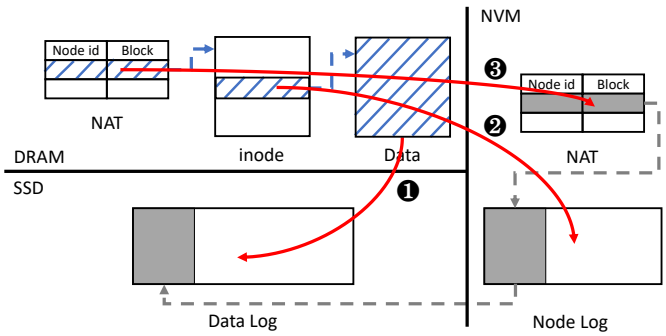

그림 2: NVM 노드 로그의 On-disk 자료 구조와 Flush 과정

3 NVM 기반 노드 로깅 설계 및 구현

3.1 F2FS

F2FS [1]는 SSD의 특성 고려하여 설계된 로그 기반 파일시스템으로 멀티 헤드(Multi-Head) 로그를 사용하여 SSD 내부의 병렬성을 높인다. F2FS의 로그들은 노드(Node) 로그와 데이터(Data) 로그로 구성된다. 노드 로그는 파일의 Inode와 Direct 노드, Indirect 노드로 구성되며 노드 ID를 통해 접근한다. 데이터 로그는 사용자의 데이터와 디렉토리 정보들로 구성되며 블록 주소로 접근한다. 파일시스템의 메타데이터로는 NAT(Node Address Table)와 SIT(Segment Information Table), SSA(Segment Summary Area), CP(Checkpoint Pack)이 존재한다. NAT는 노드의 ID와 해당 노드의 실제 저장위치를 관리하는 테이블로, 메모리 관리에서 페이지 테이블과 같은 역할을 한다. SIT는 각 세그먼트에 대하여 유효 블록을 비트맵으로 관리한다. NAT와 SIT는 각각 Shadow Copy 기법을 사용하여 체크포인트 과정에서의 시스템 장애에 대비한다.

그림 1은 F2FS에서 파일의 변경사항들을 Flush하는 과정을 보여준다. 파일의 Inode 번호가 노드의 ID가 되며 NAT를 통해 해당 Inode가 저장되어 있는 블록 주소를 알아내고 이를 이용하여 Inode를 참조한다. 파일의 Inode에는 파일의 메타데이터들과 데이터 블록에 대한 주소들이 있어 이를 이용하여 데이터를 참조할 수 있다. 이러한 파일의 데이터와 노드 로그, NAT는 *fsync* 시스템 콜이 호출되었을 때 SSD로 Flush 되어야 한다.

이 때, 파일시스템의 일관성을 지켜주기 위하여 체크포인팅(Checkpointing)을 수행한다. 체크포인팅은 ❶ 데이터 로그, ❷ 노드 로그, ❸ 파일시스템 메타데이터의 순서로 Flush를 하게 된다. 이러한 순서를 지키며 쓰게 되면 로그 기반 파일시스템의 특성상 이전의 데이터들이 보존되므로 시스템 장애시에도 쉽게 복구할 수 있다.

F2FS는 Roll-forward 복구 방식도 사용하기 때문에 모든 *fsync*에서 체크포인팅이 발생하지 않는다. 하지만 Roll-forward 복구를 위해서 NAT와 같은 파일시스템의 메타데이터만 SSD로 Flush 하지 않을뿐, 데이터 로그와 노드 로그는 SSD로 Flush를 해야한다. 또한 *fsync*가 자주 불리거나 세그먼트 클리닝이 필요한 상황이 발생하면 체크포인팅이 발생하며 데이터 로그, 노드 로그와 함께 파일시스템 메타데이터도 SSD로 Flush 해야 한다.

3.2 NVM 기반 노드 로깅 기법

그림 1에서 설명한 Flush 과정들은 파일시스템의 다른 I/O 요청들을 블록킹하고 진행되기 때문에 전체적인 성능을 저하시킨다. 특히 *fsync*가 자주 호출될수록 데이터와 노드 로그가 많이 쓰이며 세그먼트 클리닝도 자주 호출되므로 체크포인팅도 자주 발생하게 된다. 또한 메모리에 비하여 느린 블록 디바이스에 저장해야 하므로 *fsync*는 다른 쓰기들에 대한 Latency 증가에도 영향을 미친다.

본 논문에서는 *fsync*로 인한 성능 저하와 오버헤드를 해결하기 위하여 NVM 기반 노드 로깅 기법을 제안한다. 기존 F2FS에서 노드 로그와 파일시스템 메타데이터를 NVM으로 저장함으로써 쓰기 속도와 체크포인팅 시간을 개선하여 *fsync*의 지연시간을 감소시킬 수 있다. 결과적으로 *fsync*가 많은 응용프로그램에서 I/O 성능을 향상시킬 수 있다.

그림 2는 NVM 기반 노드 로깅 기법에서 데이터 로그, 노드 로그와 NAT를 Flush할 때 NVM에 기록하는 과정을 보여준다. 데이터 로그의 경우 기존 F2FS와 같이 SSD에 존재하며 NVM에는 노드 로그와 NAT가 존재하게 된다. NVM에 노드 로그와 파일시스템 메타데이터를 쓸 때에도 일관성을 보장해야 하므로 기존의 Flush와 같이 ❶ 데이터 로그, ❷ 노드 로그, ❸ 파일시스템 메타데이터의 순서로 저장하게 된다. 이 때, 노드 로그와 NAT를 저장하는 순서는 일관성에 치명적이므로 *clflush*와 *mfence*와 같은 명령들을 이용하여 캐시 라인을 Flush하고 명령어들간의 재정렬을 방지해야 한다.

NVM 노드 로깅 기법에서는 NVM 영역에 대한 Free 블록 리스트를 관리한다. 노드 로그 엔트리를 쓰기 위해서 Free 블록 리스트에서 새로운 Free 블록을 할당 받는다. 노드 로그 엔트리가 업데이트 되면 기존의 엔트리는 Invalid가 되며 Free 리스트에 다시 추가된다. 따라서 NVM 영역에 대해서는 Garbage Collection이 필요 없다. 또한 SSD는 데이터 로그만 저장하므로 데이터 로그에 대한 세그먼트 클리닝만 수행하게 된다.

4 실험 결과

4.1 실험 환경

표 1은 본 논문에서 제안한 NVM 노드 로깅 기법을 평가하기 위한 실험 환경이다. *fsync*가 많은 환경을 고려하여 FxMark [6] 벤치마크의 DWSL 워크로드를 이용하였고 기존 F2FS와 I/O 지연시간과 처리량을 비교하였다. FxMark는 파일시스템의 코어 수에 따른 확장성을 확인하기

CPU	Intel Xeon E7-8870 v2 2.3GHz (15cores) x 8
RAM	740GB
SSD	Intel Optane SSD 900p
NVM	128GB Emulated with Intel PMEM
Kernel	Linux 4.14.11
OS	Ubuntu 14.04

표 1: 실험 환경 설정

위한 벤치마크이며, DWSL은 각 코어가 서로 다른 파일에 4KB의 데이터를 쓰고 *fsync*를 호출한다.

F2FS는 세그먼트 클리닝 오버헤드를 개선하기 위하여 Threaded 로깅 기법 [7]을 사용한다. Threaded 로깅은 로그 기반 파일시스템에서 빈 로그 공간이 부족할 때 Invalid인 로그 공간에 덮어 쓰기를 함으로써 세그먼트 클리닝으로 인한 성능 저하를 줄인다. 본 실험에서는 Threaded 로깅을 활성화/비활성화 했을 경우에 대하여 모두 실험하였다.

4.2 결과 및 분석

Latency	Baseline w/o Threaded Logging	NVM_Node w/o Threaded Logging
fsync (ms)	596.8	88.2
write (ns)	4057	3788

표 2: 15코어에서 *fsync*와 *write*에 대한 평균 지연시간 비교

표 2는 Threaded 로깅을 비활성화 했을 때 15코어에서 기존 F2FS(Baseline)와 NVM 노드 로깅 기법의 *fsync*와 *write*의 평균 지연시간을 보여준다. *fsync*의 지연시간은 NVM 노드 로깅 기법에서 기존 F2FS에 비하여 약 85% 감소하였다. 이는 NVM 노드 로깅 기법으로 인하여 Flush와 체크포인팅 시간이 감소함에 따라 *fsync*의 지연시간이 감소하였기 때문이다. *write*의 지연시간은 기존 F2FS에 비하여 NVM 노드 로깅 기법에서 6.6% 감소하였다.

그림 3은 DWSL에 대한 기존 F2FS(Baseline)와 NVM 노드 로깅 기법의 I/O 처리량을 보여준다. Threaded 로깅을 비활성화 했을 때, NVM 노드 로깅 기법은 기존 F2FS보다 15코어에서 최대 약 1.7배의 성능 향상이 있었다. 이는 Sequential 로깅의 경우 새로운 로그 공간을 위해서는 세그먼트 클리닝과 체크포인팅이 수행되어야 하기 때문이다. Threaded 로깅 활성화 했을 때에는 NVM 노드 로깅 기법이 기존 F2FS보다 15코어에서 약 11%의 성능 향상을 보였다. 이는 새로운 로그 공간이 필요해도 Threaded 로깅은 세그먼트 클리닝과 체크포인팅을 하지 않기 때문이다.

또한 15코어 이후로 모든 워크로드에서 처리량이 감소하는 것을 볼 수 있다. 이는 CPU가 15코어 단위의 NUMA 구조이기 때문에 발생하는 원거리 메모리 접근에 의한 성능 하락으로 추정된다.

5 결 론

본 논문에서 제안하는 NVM 노드 로깅 기법은 비휘발성 메모리의 특성을 활용하여 F2FS에서 *fsync*의 응답 속도를 개선하고 *fsync*가 많은 워크로드의 I/O 성능을 향상시켰다. NVM 노드 로깅 기법은 노드 로그와

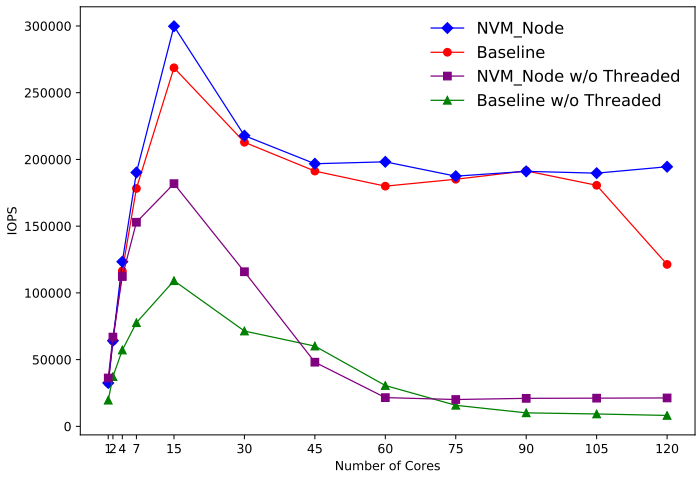

그림 3: DWSL에 대한 기존 F2FS와 NVM 노드 로깅의 처리량

파일시스템 메타데이터를 NVM에 저장하여 SSD로 쓰이는 데이터의 양을 줄여 *fsync* 오버헤드를 감소시켰다. 그 결과 제안된 기법은 기존 F2FS보다 *fsync*의 I/O 지연시간을 약 85% 줄었으며 I/O 처리율은 최대 약 1.7배 향상되었다.

감사의 글

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. Lee, D. Sim, J. Hwang, and S. Cho, "F2FS: A new file system for flash storage," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, (Santa Clara, CA), pp. 273–286, USENIX Association, 2015.

[2] C. Chen, J. Yang, Q. Wei, C. Wang, and M. Xue, "Fine-grained metadata journaling on nvm," in *2016 32nd Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–13, May 2016.

[3] D. Park and D. Shin, "iJournaling: Fine-grained journaling for improving the latency of fsync system call," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, (Santa Clara, CA), pp. 787–798, USENIX Association, 2017.

[4] Q. Wei, C. Wang, C. Chen, Y. Yang, J. Yang, and M. Xue, "Transactional nvm cache with high performance and crash consistency," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC 17)*, (New York, NY, USA), pp. 56:1–56:12, ACM, 2017.

[5] J. Yeon, M. Jeong, S. Lee, and E. Lee, "RFLUSH: Rethink the flush," in *16th USENIX Conference on File and Storage Technologies (FAST 18)*, (Oakland, CA), pp. 201–210, USENIX Association, 2018.

[6] C. Min, S. Kashyap, S. Maass, and T. Kim, "Understanding manycore scalability of file systems," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, (Denver, CO), pp. 71–85, USENIX Association, 2016.

[7] Y. Oh, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Optimizations of LFS with slack space recycling and lazy indirect block update," in *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, SYSTOR '10, (New York, NY, USA), pp. 2:1–2:9, ACM, 2010.