로그 기반 파일 시스템의 매니 코어 성능 및 확장성 분석

노성현°, 이창규, 변현기, 김장웅, 김영재
서강대학교 컴퓨터공학과
{nsh0249, bhyunki, changgyu, ski6812, youkim}@sogang.ac.kr

Manycore Performance and Scalability Analysis of A Log-Structured File System

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

요 약

매니 코어 환경은 병렬처리를 통해 작업량을 높일 수 있기 때문에 고성능 시스템에서 널리 사용되고 있다. 특히 I/O 연산이 많은 데이터베이스 시스템에서는 매니 코어 환경을 이용한 I/O 성능 증가를 기대할 수 있다. 하지만, 현재 파일시스템에서 사용하고 있는 파일 단위의 아이노드 락(Inode Lock)은 매니 코어 환경에서의 병렬성을 저해한다. 따라서 본 논문에서는 I/O 성능을 높이기 위해서 아이노드 락을 범위 락(Ragne Lock)으로 대체하는 기법을 소개한다. 범위 락(Range Lock)은 파일 전체가 아닌 접근하고자 하는 일부분 만을 Lock함으로써 매니 코어 환경에서의 병렬성을 극대화 할 수 있다. 높은 내부 병렬성을 갖는 SSD와 이에 최적화 되어 있는 대표적인 로그 기반 파일 시스템인 F2FS를 사용하여 범위 락을 구현하였으며 다양한 파일 시스템 성능 테스트를 위한 합성 워크로드 벤치마크(Synthetic Workload Benchmark)인 FxMark를 사용하여 성능을 측정하였다. 그 결과 병렬 단일 파일 쓰기 워크로드인 DWOM에서 범위 락을 적용한 F2FS가 기존 F2FS보다 최대 40.5배 높은 IOPS를 보였다.

1 서 론

매니 코어 환경은 높은 병렬성을 활용하여 I/O 성능을 증가시킬 수 있기 때문에 많은 고성능 시스템에서 사용되고 있다. 데이터베이스 시스템은 다수의 질의를 동시에 처리할 수 있으며 그 결과를 저장장치에 저장한다. 따라서 I/O 병렬화를 이용하기에 적합하다. 또한 데이터베이스 시스템에서는 빠른 I/O 처리를 위하여 SSD(Solid-State Drive)를 주로 사용한다 [1]. F2FS [2]는 SSD에 최적화된 로그 구조 파일 시스템으로 임의 쓰기 성능이 순차 쓰기 성능보다 비교적 낮은 SSD에서 좋은 성능을 보인다. 또한 멀티 헤드 로깅 기법을 적용하여 SSD의 내부 병렬성을 극대화한다. 그러나 데이터베이스 시스템은 이러한 매니 코어 환경의 I/O 성능을 제대로 활용하지 못한다. 그 예시로, TPC-C [3] 워크로드를 처리할 때 발생하는 쓰기 명령들의 직렬화가 있다.

TPC-C는 OLTP(On-Line Transaction Processing)를 시뮬레이션하는 워크로드이다. TPC-C 워크로드를 사용한 데이터베이스의 I/O 패턴은 여러 스레드(Thread)가 같은 파일에 다른 영역을 쓰는 병렬 단일 파일 쓰기가 대부분을 차지한다. 그러나 현재 파일 시스템에서는 병렬 단일 파일 쓰기가 발생했을 때 파일 단위의 아이노드 락(Inode Lock)을 걸기 때문에 여러 쓰기 명령이 직렬화된다. 따라서 매니 코어 환경에서의 I/O 병렬화를 저해하여 성능을 향상시키지 못한다.

본 논문에서는 병렬 단일 파일 쓰기의 직렬화를 해결하기 위하여 파일 시스템 단계에서 범위 락(Range Lock)[4]을 제안한다. 범위 락은 파일 전체가 아닌 쓰고자 하는 범위에만 락을 건다. 따라서 병렬 단일 파일 쓰기의 경우, 범위가 다르기 때문에 동시에 수행 할 수 있어 I/O 병렬성을 활용할 수 있다. 제안한 기법의 성능 평가를 위하여 F2FS에 범위 락을 적용하여 기존 F2FS와의 I/O 처리량을 FxMark [5] 벤치마크를 사용하여 비교하였다. FxMark는 합성 워크로드(Synthetic Workload)를 사용하는 파일 시스템 벤치마크로 다양한 I/O 패턴에 대하여 성능을 측정할 수 있다. 그 중 병렬 단일 파일 쓰기에 해당하는 DWOM 워크로드에 대하여 범위 락을 적용한 F2FS는 기존 F2FS와 비교하여 IOPS가 평균 14.8배 증가하였고 4코어에서 최대 40.5배 증가하였다.

2 배경 지식

2.1 F2FS

F2FS는 최첨단 로그 구조 파일 시스템으로 플래시 저장 장치에서 좋은 성능을 보여준다. 로그 구조 파일 시스템에서는 데이터가 로그에 순차적으로 저장된다. 이는 상대적으로 임의 쓰기의 속도가 느린 SSD에서 좋은 성능을 보여준다. 그림 1는 F2FS의 On-disk 자료구조로 NAT(Node Address Table), SB(Super Block), CP(Checkpoint), SIT(Segment Information Table), SSA(Segment Summary Table)로 구성되어 있다.

기존의 로그 구조 파일 시스템에서는 블록이 업데이트 되면 블록을 가리키는 다이렉트 블록, 다이렉트 블록을 가리키는 인다이렉트 블록, 인다이렉트 블록을 가리키는 아이노드가 순차적으로 업데이트 되는 Wandering tree 문제가 발생한다. F2FS에서는 이러한 문제를 해결하기 위해서 노드의 주소를 저장하는 NAT라는 자료구조를 구성하였다.

F2FS는 멀티 헤드 로깅을 사용하여 성능을 높인다. 로깅을 할 때 여러 개의 로그에 동시에 로깅해서 병렬성을 높혔다. 또한 데이터를 쓸 때 해당 데이터의 업데이트 빈도를 예상하여 Hot/Cold로 데이터를 구분해서 각자 다른 로그에 할당한다. 이는 SSD의 가비지 콜렉션(Garbage Collection) 성능 향상에 큰 도움이 된다. F2FS에서는 로깅 할때 두가지 방법을 사용한다. 클린 세그먼트에 로깅

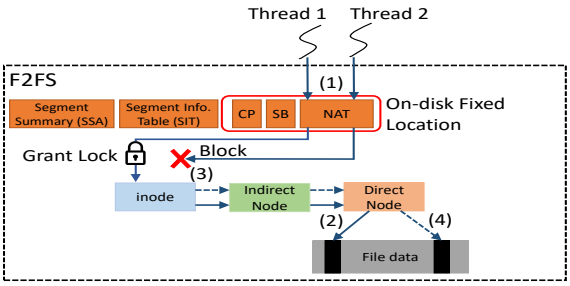


그림 1: F2FS 파일 시스템의 구조.

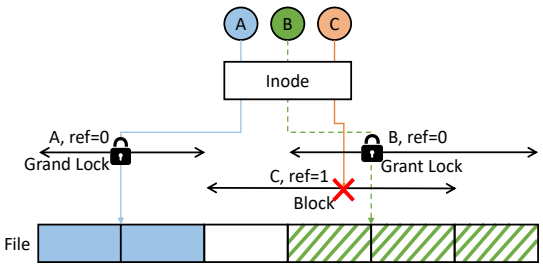


그림 2: 범위 락을 이용한 파일의 접근.

하는 방법(Append-only logging)과 더티 세그먼트의 무효화된 블록에 로깅(Threaded logging)하는 방법이다.

F2FS에서는 갑작스러운 시스템 장애 발생시 일관성 있는 회복을 위하여 체크포인트 기법을 사용한다. 또한 시스템 오류 발생시 Roll-forward와 Roll-back 두가지 회복 기법을 사용한다. Roll-back 회복 기법은 시스템 오류 발생시 가장 최근 버전의 체크포인트로 회복한다. 작은 크기의 쓰기와 fsync가 자주 호출되는 경우 데이터 블록과 다이렉트 노드만 기록한 후 시스템 오류 발생시 Roll-forward 기법으로 회복할 수 있다.

2.2 병렬 단일 파일 쓰기

F2FS에서 병렬 단일 파일 쓰기의 실행 과정은 그림 1에서 볼 수 있다. (1) 스레드 2개가 같은 파일에 동시에 쓰기를 시작한다. 먼저 도착한 스레드1은 쓰려는 파일의 아이노드에 락이 걸려 있지 않기 때문에 파일의 아이노드에 락을 걸고 쓰기를 진행한다. 이후 아이노드 락에 의해 같은 파일에 대한 쓰기가 블록되기 때문에 스레드2는 쓰기를 일시 정지한다. (2) 스레드1은 데이터를 해당 위치의 블록에 쓰고 메타데이터를 수정한 후 파일의 락을 해제하고 쓰기를 종료한다. (3) 스레드1이 쓰기를 종료하면 스레드2가 파일의 아이노드에 락을 걸고 쓰기 명령을 재개한다. (4) 스레드2가 쓰기를 수행 후 파일의 락을 풀고 종료된다. 결국 F2FS에서 여러 스레드의 병렬 단일 파일 쓰기는 순차적으로 실행된다.

F2FS에서 병렬 단일 파일 쓰기는 아이노드 락 때문에 동시에 실행되지 않는다. 만약 병렬 단일 파일 쓰기에서 각 스레드들이 파일의 다른 위치에 쓴다면 파일 단위 아이노드 락은 필요하지 않는다. 이 경우 쓰기를 하는 범위에만 락을 사용하여 여러 스레드가 데이터를 동시에 쓸 수 있다. 이러한 Fine-grained 한 락을 범위 락(Range-lock) 이라고 한다. 범위 락을 사용하면 병렬 단일 파일 쓰기의 병렬성을 높여서 F2FS의 I/O 성능을 증가시킬 수 있다.

3 범위 락 기반 F2FS 설계 및 구현

3.1 범위 락

범위 락은 인터벌 트리로 구현을 할 수 있다. 인터벌 트리는 구간을 가지는 노드들로 구성된 트리형 자료구조이다. 인터벌 트리는 겹치는 구간을 가지는 다른 노드를 감지할 수 있으며 각 노드들은 자신의 노드의 구간과 겹치는 구간을 가지는 노드의 갯수를 나타내는 레퍼런스 카운터(Reference counter)를 가진다. 범위 락은 범위에 락을 걸때 인터벌 트리에 노드를 삽입하고 노드의 레퍼런스 카운터가 0일 때 락을 풀어준다. 언락 함수가 호출되면 트리에서 노드를 제거한다.

인터벌 트리를 이용한 범위 락의 동작 매커니즘는 그림 2에서 볼 수 있다. 이 그림에서 병렬 단일 파일 쓰기를 실행하는 스레드 A, B, C가 순서대로 실행된다. ref는 레퍼런스 카운터를 나타낸다. 스레드 A가 파일에 쓸 때 A와 구간이 겹치는 다른 노드가 없기 때문에 ref값은 0이 되고 노드는 트리에 삽입된다. A의 ref값이 0이기 때문에 락이 걸리지 않고 실행된다. 그 다음 실행되는 스레드 B 또한 구간이 겹치는 다른 노드가 없기 때문에 ref값은 0이 되고 노드는 트리에 삽입된 후 락이 걸리지 않고 바로 실행된다. 스레드 C의 구간은 B의 구간과 겹치기 때문에 노드가 트리에 삽입된 후 락이 걸리고 쓰기는 실행되지 않는다. B의 쓰기가 끝나면 B는 노드에서 제거되고 C의 ref값은 0이 되어서 쓰기가 시작된다.

3.2 범위 락 설계 및 구현

본 논문에서는 인터벌 트리로 구현된 범위 락 패치를 F2FS에 적용하였다. 패치를 리눅스 커널 4.14.11에 적용하기 위해서 리눅스 커널과 F2FS의 코드를 적절히 수정하였다. 락의 범위를 구하기 위해서 F2FS의 쓰기 함수에서 파일의 오프셋과 쓰는 길이 정보를 이용하여 범위 락을 걸 구간을 구한 후 F2FS의 쓰기 함수의 시작과 끝에 호출되는 아이노드 락 및 아이노드 언락을 범위 락 및 범위 언락으로 대치하였다.

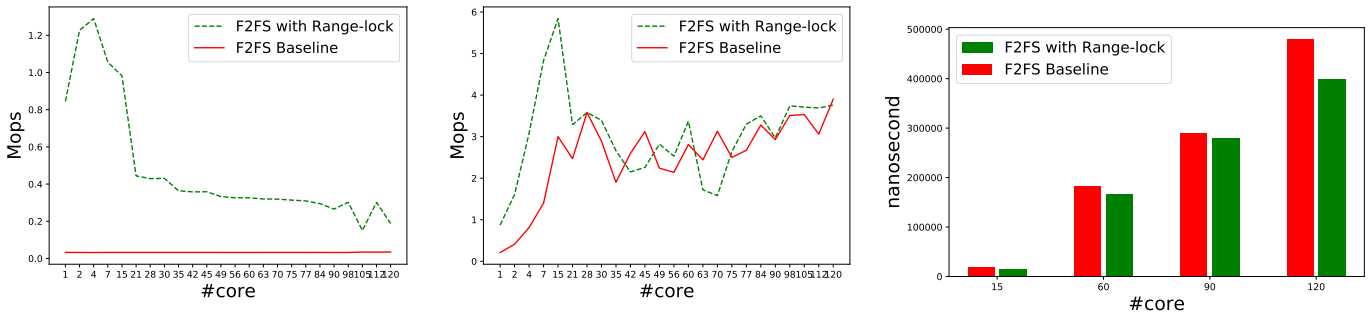
4 실험 및 분석

4.1 실험 환경

CPU	Intel Xeon E7-8870 v2 2.3GHz (15cores) x 8
RAM	740GB
SSD	Intel Optane SSD 900p
Kernel	Linux 4.14.11
OS	Ubuntu 16.04

표 1: 실험 환경 설정.

표 1는 본 논문에서 제안한 범위 락 기법을 평가하기 위한 실험 환경이다. 병렬 단일 파일 쓰기 환경에서 범위 락 기법을 통한 성능 향상을 보기 위하여 FxMark 벤치마크의 DWOM 워크로드를 사용



(a) DWOM 처리율 (b) DWOL 처리율 (c) DWOM 지연 시간

그림 3: (a) 병렬 단일 파일 쓰기(DWOM)과 (b) 병렬 다른 파일 쓰기(DWOL) 워크로드에 대한 기본 F2FS 및 Range-lock 적용 F2FS의 처리율 확장성 실험. (c) 병렬 단일 파일 쓰기(DWOM) 워크로드에 대해 기본 F2FS 및 Range-lock 적용 F2FS가 한 페이지를 쓸 때 걸리는 평균 지연 시간 측정 실험.

하였다. 그리고 병렬 다른 파일 쓰기인 DWOL 워크로드를 사용하여 실험하였다.

먼저, 매니 코어를 사용하는 환경에서 F2FS의 확장성을 분석하기 위하여 FxMark를 이용하여 DWOM, DWOL 워크로드에서 기존 F2FS와 범위 락이 구현된 F2FS의 I/O 처리량을 비교하였다. 그 다음, 기존 F2FS와 범위 락이 구현된 F2FS가 병렬 단일 파일 쓰기에서 한 페이지를 쓸 때 걸리는 평균 지연 시간을 측정하였다. 지연 시간은 15, 60, 90, 120 코어를 사용할 때 측정하였다.

4.2 실험 결과 및 분석

그림 3에서는 두가지 워크로드를 사용하여 기존 F2FS와 범위 락을 적용한 F2FS의 확장성을 비교하였다. 그림 3(a)는 DWOM 워크로드로 기존 F2FS는 코어의 갯수에 관계 없이 성능에 변화가 거의 없음을 볼 수 있다. 이는 동작하는 코어의 개수를 늘려서 병렬적으로 처리할 수 있는 I/O를 증가시켜도 아이노드 락 때문에 쓰기 요청이 동시에 실행되지 않고 순차적으로 실행되기 때문이다. 반면, 범위 락을 적용한 F2FS에서는 기존 F2FS보다 최대 13.6배의 성능 증가를 보였다. 범위 락 F2FS의 경우 4개의 코어까지는 성능의 확장성을 보였지만 그 이후로는 성능이 감소하였다. 하지만 기존 F2FS보다는 훨씬 더 좋은 성능을 보였다.

그림 3(b)는 DWOL 워크로드의 실험 결과로 기존 F2FS와 범위 락을 적용한 F2FS가 비슷한 성능을 보였다. 두 경우 모두 성능의 확장성을 보였다. 이는 DWOL 워크로드의 경우 여러 스레드가 각각 다른 파일에 쓰기를 수행하기 때문에 아이노드 락에 의해 블록되는 스레드가 없다. 즉, 여러 쓰기가 동시에 수행될 수 있기 때문에 성능의 확장성을 볼 수 있었다. 범위 락을 적용한 F2FS의 경우 28코어 까지 기존 F2FS보다 좋은 성능이 나왔다. 이는 범위 락을 수행할 때 발생하는 오버헤드와 아이노드 락을 수행할 때 발생하는 오버헤드가 다르기 때문에 성능의 차이가 발생한다.

그림 3(c)는 DWOM 워크로드에서 한 페이지에 쓰기를 할 때 발생하는 지연 시간을 비교한 결과이다. 모든 경우에서 범위 락을 적용한 F2FS가 기존 F2FS보다 한 페이지를 쓸 때 걸리는 지연 시간이 평균 14% 감소하였고 120 코어에서 최대 17% 감소하였다. 또한 코어의 수가 증가하면 지연 시간 또한 증가함을 볼 수 있었다. 이는 매니코어 환경에서 여러 프로세서를 사용할 때 발생하는 NUMA현상 때문이다.

5 결론

본 논문에서는 매니 코어 환경에서 I/O 성능이 확장하지 않는 원인을 파일 시스템 관점에서 분석하였다. 병렬 단일 파일 쓰기 워크로드를 처리할 때 파일 시스템에서 호출하는 파일 단위의 아이노드 락 때문에 I/O가 순차적으로 처리된다. 이를 해결하기 위하여 블록 단위의 범위 락을 F2FS에 적용하고 성능을 평가하였다. 병렬 단일 파일 쓰기에서 범위 락이 구현된 F2FS의 경우 기존 F2FS보다 평균 14.8배, 최대 40.5배의 성능 증가를 보였다. 또한 범위 락이 구현된 F2FS가 기존 F2FS보다 한 페이지를 쓸 때 걸리는 지연 시간이 평균 14%, 최대 17% 감소하였다.

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] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim, “A case for flash memory ssd in enterprise database applications,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1075–1086, ACM, 2008.

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

[3] TPC-C contributors, “TPC-C,” 1992. [Online; accessed 31-October-2018].

[4] J. Kara., “Range reader/writer locks for the kernel.” [Online; accessed 31-October-2018].

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