Fine-Grained Journaling에서의 fsync 지연시간 개선을 위한 선택적 Grouped fsync 기법 제안

김솜○, 박대준, 신동군
성균관대학교 전자전기컴퓨터공학과
sommkim@skku.edu, pdaejun@skku.edu, dongkun@skku.edu

Selective Grouped fsync for Improving fsync System Call Latency in Fine-Grained Journaling

Somm Kim○, Daejun Park, Dongkun Shin
Department of Electronical and Computer Engineering, Sungkyunkwan University

요 약

대부분의 시스템에서 데이터 내구성을 보장하기 위해 fsync 시스템 콜을 사용한다. fsync 시스템 콜은 현재 수행 중인 프로세스와 동기적으로 동작하기 때문에 fsync 시스템 콜의 지연시간을 개선하는 것은 시스템의 응답시간 측면에서 중요하다. 기존 선행 연구로 Fine-Grained Journaling인 파일 수준 저널링을 사용하여 컴파운드 트랜잭션으로 인한 fsync 시스템 콜의 지연 문제를 해결한 iJournaling[1]이 있다. 하지만 iJournaling의 경우 파일 수준 저널링을 사용하므로 플러시(flush) 동작의 횟수가 증가하고, fsync 시스템 콜이 여러 코어에서 비슷한 시기에 발생한다면 각 코어에서 비슷한 시기에 플러시 동작이 수행되어 플러시로 인한 fsync 지연 문제가 발생한다. 따라서 본 논문에서는 파일 수준 저널링에서의 플러시 횟수 증가로 인한 fsync 시스템 콜의 지연시간 개선을 위해 각 코어에서 수행되는 fsync 동작을 선택적으로 그룹핑하여 플러시 횟수를 줄임으로써 fsync 지연시간을 줄였다.

1. 서 론

대부분의 파일시스템은 시스템에 장애가 발생했을 때 데이터 일관성을 복구하기 위해 저널링을 사용한다. 저널링이란 파일시스템의 변경사항을 실제 스토리지 영역에 기록하기 전에 먼저 저널 영역에 기록하여 시스템 장애가 발생했을 때 저널 영역에 기록된 변경사항을 읽어 실제 스토리지에 기록하는 과정을 통해 데이터 일관성을 복구하는 기법이다.

리눅스 커널의 기본 파일시스템인 ext4에서는 저널링을 위해 JBD2(journaling block device2) 쓰레드를 사용하여 5초 동안 발생하는 파일시스템의 변경사항을 묶어서 컴파운드 트랜잭션(compound transaction)으로 커밋하여 실제 스토리지 영역에 기록한다. 하지만 fsync 시스템 콜에 대해서 컴파운드 트랜잭션을 사용하면 특정 파일에 대한 즉시 내구성을 보장할 수는 있지만, fsync 하려는 파일에 대한 변경사항 뿐만 아니라 관련 없는 다른 파일에 대한 변경사항도 함께 커밋하므로 해당 파일의 fsync 지연시간이 길어지는 문제가 있다.

또한 fsync 시스템 콜은 현재 수행 중인 프로세스와 동기적으로 동작하므로 fsync 지연시간은 시스템의 성능과 밀접한 관련이 있다. 따라서 iJournaling[1]에서는 fsync와 관련된 파일의 변경 사항만을 기록하는 파일 수준 저널링을 통해 컴파운드 트랜잭션으로 인한 fsync 지연시간 문제를 해결하였고 모든 데이터를 저널 영역에 기록하지 않고 최소한의 정보만을 기록하고, 이를 이용해 데이터 일관성을 복구할 수 있는 기법을 제안하였다. 그리고 코어 별로 발생하는 fsync에 대해서 서로 다른 저널 영역을 사용하도록 하여 동시에 저널링 동작이 수행 가능하고 데이터 일관성을 유지한다.

하지만 파일 수준 저널링을 사용하는 iJournaling에서는 fsync에 대한 동작이 컴파운드 트랜잭션으로 처리되지 않으므로 스토리지 내 버퍼에서 실제 스토리지 영역에 기록하는 플러시 동작의 횟수가 기존 ext4 저널링 기법에 비해 증가하는 단점이 있고, fsync 시스템 콜이 여러 코어에서 비슷한 시기에 발생한다면 각 코어에서 비슷한 시기에 플러시 동작이 수행되어 플러시로 인한 fsync 지연 문제가 발생한다. 그림 3을 보면 실제로 플러시로 인한 지연시간은 fsync 동작을 수행하는데 가장 큰 비중을 차지한다.

따라서 본 논문에서는 iJournaling에서 컴파운드 트랜잭션을 수행하지 않아서 과도하게 증가하는 플러시 횟수를 줄이기 위해 파일 수준 저널링에서의 동시에 수행 중인 fsync 동작을 선택적으로 그룹핑하는 Grouped fsync 기법을 제안한다.

2. 관련 연구

파일시스템의 성능 고립화를 통해 컴파운드

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2016R1A2B2008672)

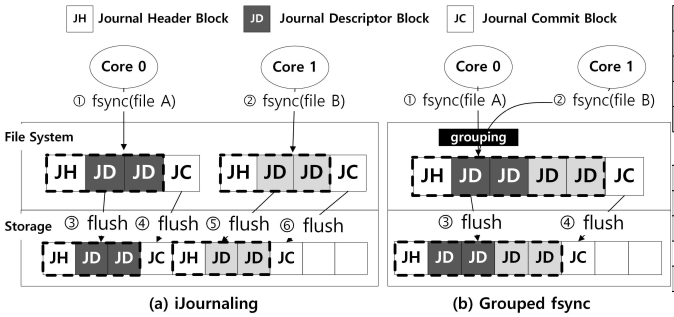

그림 1 iJournaling 기법과 Grouped fsync 기법의 플러시 횟수 비교

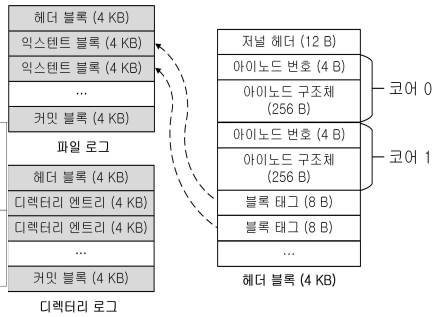

그림 2 Grouped fsync 저널 구조

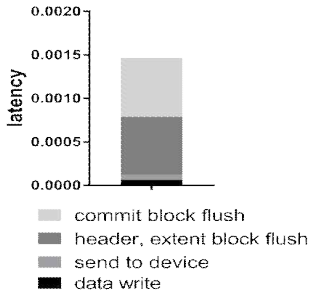

그림 3 fsync 동작 별 지연시간

트랜잭션으로 인한 fsync 지연시간을 개선한 연구로는 PBG(per-block-group)[2]과 IceFS[3], SpanFS[4]가 있다.

PBG에서는 ext4의 블록 그룹을 고립화 단위로 사용하고 fsync가 호출된 파일이 포함된 블록 그룹의 트랜잭션만을 커밋한다. 하지만 fsync가 호출된 파일과 관련 없는 파일들이 같은 블록 그룹을 공유할 경우 여전히 컴파운드 트랜잭션으로 인한 fsync 지연 문제가 발생한다.

IceFS에서는 파일과 디렉터리를 기반으로 파일시스템의 물리적 구조를 큐브(cube)라 불리는 단위를 통해 분리하여 파일시스템의 고립성을 제공한다. 따라서 서로 다른 큐브 간에는 컴파운드 트랜잭션으로 인한 fsync 지연 문제가 발생하지 않는다. 그러나 동일 큐브 내에서는 여전히 컴파운드 트랜잭션으로 인한 fsync 지연 문제가 발생하고, 또한 큐브 간에는 서로 디렉터리 구조를 공유할 수 없어, 사용자가 직접 큐브를 관리해야 하는 등 파일시스템을 사용하는데 제약사항이 발생한다.

SpanFS는 IceFS에서 큐브 간 서로 디렉터리 구조를 공유할 수 없는 문제를 해결하기 위해 도메인(domain)이라는 고립화 단위를 사용한다. 도메인은 큐브와 달리 도메인 간의 디렉터리 구조 연결을 지원하므로 사용자가 파일시스템을 사용하기에 제약사항이 없다. 그러나 SpanFS 또한 같은 도메인 내에서는 컴파운드 트랜잭션으로 인한 fsync 지연 문제가 있다.

3. 선행 연구

iJournaling[1]은 앞선 관련 연구에서 해결하지 못한 컴파운드 트랜잭션으로 인한 fsync 지연 문제를 Fine-Grained Journaling인 파일 수준 저널링을 사용하여 해결하였다. iJournaling에서는 컴파운드 트랜잭션으로 인해 fsync와 관련 없는 파일의 변경 사항이 기록되어 fsync 지연시간이 길어지는 것을 피하기 위해, fsync가 호출된 파일에 해당하는 데이터와 메타데이터만을 기록한다. 파일시스템 메타데이터인 블록 비트맵과 아이노드 비트맵, GDT(group descriptor table)은 여러 파일과 연관되어 있기 때문에 iJournaling에서는 하나의 파일에만 속하는 파일 메타데이터인 아이노드와 익스텐트 블록(extent block)만을 기록한다.

따라서 iJournaling에서 fsync 저널 구조는 헤더 블록, 익스텐트 블록, 커밋 블록으로 구성되어 있고, 헤더 블록에는 저널 헤더, 아이노드 번호, 아이노드 구조체, 블록 태그가 포함된다. 저널 헤더에는 저널 간의 순서를 파악할 수 있는 숫자가 기록되고, 아이노드는 fsync를 발생시킨 파일에 대한 것이고, 블록태그는 헤더 블록 뒤에 기록되는 익스텐트 블록에 대한 주소 정보를 가지고 있다. 그리고 fsync 저널의 마지막에는 커밋 블록을 기록하여 해당 fsync 저널이 유효하다는 것을 나타낸다. 또한 디렉터리 접근성을 위해 fsync에 해당하는 변경 중인 모든 조상 디렉터리에 대해 디렉터리 엔트리를 포함한 디렉터리 로그를 기록한다.

따라서 iJournaling에서는 하나의 fsync에 대해 헤더 블록과 익스텐트 블록에 대한 플러시와 커밋 블록에 대한 플러시, 총 두 번의 플러시가 발생한다.

4. Grouped fsync: 선택적 fsync 그룹핑 기법

선행 연구인 iJournaling에서 fsync 지연시간을 개선하기 위해 컴파운드 트랜잭션으로 처리하지 않음으로 인해 기존 ext4 저널링에 비해 플러시 횟수가 증가하는데, 그림 3을 보면 fsync 동작 수행 중 플러시에 대한 지연시간이 가장 큰 비중을 차지하는 것을 볼 수 있다. 따라서 여러 코어에서 비슷한 시기에 발생하는 fsync를 묶어서 처리하여 최대한 플러시 횟수를 줄여서 과도한 플러시로 인한 fsync 지연시간을 줄이려고 한다.

Grouped fsync 기법에서는 여러 코어에서 발생하는 fsync 시스템 콜에 대해서 아직 플러시를 수행하지 않은 코어에 fsync 처리를 넘김으로써 선택적으로 그룹핑을 한다. 그룹핑을 수행함으로써 헤더 블록과 익스텐트 블록에 대한 플러시와 커밋 블록에 대한 플러시가 그룹당 두 번만 발생하게 된다. 그림 1은 두 개의 코어에서 각각 fsync 시스템 콜이 발생했을 때 iJournaling 기법과 Grouped fsync 기법에서 몇 번의 플러시가 발생하는지를 보여준다. 그림 1(a)를 보면 iJournaling 기법에서는 fsync 시스템 콜 하나 당 헤더 블록과 익스텐트 블록에 대한 플러시와 커밋 블록에 대한 플러시, 총 두 번의 플러시가 발생하여 전체적으로 4번의 플러시가 발생하는 것을 알 수 있다. 반면에 그림 1(b)를 보면 Grouped fsync 기법에서는 비슷한 시기에 코어 1에서 발생한 fsync 시스템 콜에 대한 처리를 코어 0과 그룹핑함으로써 전체적으로 2번의 플러시만 발생하는 것을 알 수 있다.

그림 2와 같이 Grouped fsync에서는 여러 파일에 대해 발생하는 fsync를 묶어서 처리하도록 헤더 블록에 저널 헤더와 코어 개수만큼의 아이노드 번호와 아이노드 구조체가 있고, 블록 태그들을 기록한다.

또한 서로 다른 코어에서 발생하는 fsync를 묶어서 처리하기 위해 코어 간 공유하는 구조체가 있고 해당 구조체에 각 코어에서 진행 중인 fsync에 대한 정보를 기록해둔다. 코어 간 공유하는 구조체는 (1) fsync가 발생한 파일에 대한 아이노드와 (2) 해당 fsync를 실제로 처리할 코어 번호, (3) fsync가 수행 중인 단계를 나타내는 플래그, (4) 현재 코어에서 처리하는 fsync 작업들에 대한 wait queue로 구성되어 있다. fsync가 호출되면 먼저 코어 간 공유하는 구조체를 통해 fsync가 수행 중인 단계를 나타내는 플래그를 확인하여 저널 플러시가 수행되기 이전이라면 해당 코어에 fsync 처리를 위임한다.

또한 데이터 쓰기 도중에도 그룹이 가능하도록 해서 최대한 많은 fsync가 그룹으로 처리되도록 하였고, 그룹으로 묶이는 작업을 wait queue에 넣기 전에 해당 fsync에 대한 데이터 쓰기를 수행하여 메타데이터가 쓰이기 전에 데이터 쓰기가 보장되도록 하였다.

5. 실험

Grouped fsync 기법을 사용했을 때 플러시 횟수가 얼마나 감소하는지와 fsync 지연시간이 얼마나 줄어드는지 알아보기 위해 기존 ext4 저널링 기법, iJournaling 기법과 본 논문에서 제안하는 Grouped fsync 기법에서의 fsync 지연시간과 플러시 횟수를 측정하였다. 실험을 위해 사용된 컴퓨터 사양은 Intel(R) i5-6500(3.20GHz, 쿼드 코어), 16GB DRAM, SSD 256GB이고, 운영체제는 Ubuntu 14.04 LTS, 리눅스 커널은 4.7.3 버전을 사용하였다. 그리고 SSD는 85GB로 파티션을 나누었고 순차 모드를 사용하는 ext4 파일 시스템을 마운트하여 사용하였다.

fsync 지연시간을 측정하기 위해 인위적으로 코어 개수만큼 쓰레드를 생성하여 각 쓰레드에서 1000번의 4KB I/O write와 fsync를 호출하였다. 그리고 각 쓰레드에서 호출하는 fsync가 항상 같은 코어에서 수행되도록 각 쓰레드를 각 코어에 고정(CPU affinity) 시켜서 수행하였다. 또한 실험 수행 전 매번 파일 삭제와 페이지 캐시를 비움으로써 동일한 조건에서 fsync 지연시간이 측정되도록 하였다.

그림 4(a)는 위와 같은 실험 방법에서 기존 ext4 저널링 기법과 iJournaling 기법, Grouped fsync 기법에서 발생되는 전체 플러시 횟수를 나타낸 그래프이고 그림 4(b)는 각 기법 별 fsync 지연시간을 누적분포함수로 나타낸 것이다. 기존 ext4 저널링 기법에서는 플러시 횟수는 적지만 컴파운드 트랜잭션으로 인해 평균 fsync 지연시간이 가장 느린 것을 볼 수 있다. iJournaling 기법에서는 파일 수준 저널링을 수행하므로 fsync 하나 당 두 번의 플러시가 발생하므로 가장 플러시 횟수가 많지만 컴파운드 트랜잭션으로 인한 fsync 지연 문제를 해결하여 기존 ext4 저널링 기법에 비해 평균 fsync 지연시간이 많이 향상된 것을 볼 수 있고, Grouped fsync 기법에서는 파일 수준 저널링을 선택적으로 그룹핑함으로써 iJournaling 기법에 비해 전체 플러시 횟수를 2배 이상 줄였고 그로 인한 fsync 지연 문제를 해결하였다.

그림 4(b)을 보면 Grouped fsync 곡선이 iJournaling

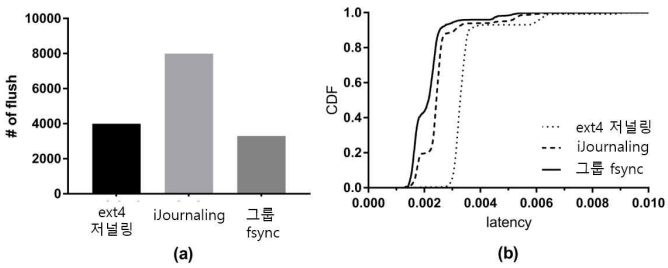

그림 4 각 기법 별 플러시 횟수 및 fsync 지연 시간 비교

곡선이나 기존 ext4 저널링 곡선보다 왼쪽에 위치하는 것을 볼 수 있다. 따라서 Grouped fsync 기법의 fsync 지연시간이 iJournaling 기법과 기존 ext4 저널링 기법의 fsync 지연시간보다 평균적으로 적게 걸리는 것을 알 수 있다.

6. 결론

Grouped fsync 기법에서는 fsync가 발생한 파일과 관련된 메타데이터만을 저널링하므로 컴파운드 트랜잭션으로 인한 fsync 지연시간 문제가 발생하지 않고, 최소한의 정보만을 이용하여 데이터 일관성을 복구할 수 있다. 또한 각 코어마다 사용하는 저널 영역이 분리되어 있으므로 코어 간 또는 백그라운드 JBD2 저널링 간 동기화가 필요하지 않다.

또한 여러 코어에서 비슷한 시기에 발생하는 fsync를 묶어서 처리하여 iJournaling 기법에서 fsync마다 두 번의 저널 플러시가 발생하는 것을 그룹 당 두 번의 저널 플러시만 발생하도록 하여 fsync 지연시간을 줄였다.

참고 문헌

[1] Daejun Park and Dongkun Shin. iJournaling: Fine-Grained Journaling for Improving the Latency of Fsync System Call. In Proceedings of the 2017 USENIX Annual Technical Conference, ATC’17, pages787-798, 2017.

[2] Yunji Kang and Dongkun Shin, Per-block-group journaling for improving fsync response time. In Proceedings of the 18th IEEE International Symposium on Consumer Electronics, pages 22-25, 2014.

[3] Lanyue Lu, Yupu Zhang, Thanh Do, Samer AI-Kiswany, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Physical disentanglement in a container-based file system. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, OSDI’14, pages81-96, 2014.

[4] Junbin Kang, Benlong Zhang, Tianyu Wo, Weiren Yu, Lian Du, Shuai Ma, and Jinpeng Huai. SpanFS: A scalable file system on fast storage devices. In Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference, ATC’15, pages249-261, 2015.

[5] 박대준, 신동군. fsync 지연시간 개선을 위한 저널링 기법 연구, 한국정보과학회2015년 동계학술발표회 논문집, 1219-1221, 2015.12