트랜잭션을 사용하는 파일시스템에서 저널 충돌 분석

이원종[0] 박성순 원유집
한양대학교 글루시스/안양대학교 한양대학교
lee20807@gmail.com sspark@gluesys.com yjwon@hanyang.ac.kr

Analysis of Journal Conflict in the Transactional Filesystem

WonJong Lee[0] SungSoon Park Youjip Won
Hanyang Univ. Gluesys/Anyang Univ. Hanyang Univ.

요 약

운영체제는 컴퓨터의 비정상적인 종료를 신속하고 정확하게 해결하기 위해 저널링 기법을 사용한다. EXT4와 Barrier-Enabled IO Stack(이하 BFS)는 트랜잭션을 사용하여 저널링을 한다. 트랜잭션을 사용하여 저널링을 하는 파일시스템은 메타데이터를 수정할 때 저널 충돌이 발생할 수 있다. 저널 충돌은 트랜잭션을 사용하는 파일시스템의 확장성을 감소시키는데 심각한 경우 스레드 개수가 증가해도 성능이 전혀 증가하지 않을 수도 있다.

1. 서 론

저널링[1]은 컴퓨터가 비정상적으로 종료되었을 때 신속하게 디스크의 일관성을 보장하기 위해서 반드시 필요하다. EXT4[2]와 BFS[3]는 트랜잭션을 이용하여 저널링을 한다. 트랜잭션은 디스크의 저널 영역에 파일시스템의 수정사항을 기록하는 단위로 원자성과 지속성을 보장한다.

EXT4는 한번에 하나의 트랜잭션만을 커밋하기 때문에 멀티코어 환경에서 확장성이 떨어진다. 반면 BFS는 다수의 트랜잭션을 커밋할 수 있기 때문에 확장성이 좋다.

하지만 BFS에서 메타데이터가 수정되면 확장성이 떨어진다. 이유는 저널 충돌로 인해 트랜잭션의 병렬성이 떨어지기 때문이다. 본 논문에서는 저널 충돌에 대해 자세히 조사하고 저널 충돌의 영향을 실험을 통해 분석한다.

2. 배 경

2.1 트랜잭션

트랜잭션은 원자성과 지속성을 보장하는 단위로 EXT4와 BFS 등 많은 파일시스템이 트랜잭션을 이용하여 저널링을 한다. 트랜잭션을 스토리지의 로그 영역에 영구적으로 쓰는 동작을 커밋이라고 한다. 커밋이 완료된 트랜잭션의 로그 블록들은 추후에 스토리지의 원래 위치에 쓰여지는데 이것을 체크포인트라고 한다.

트랜잭션은 저널 디스크립터 블록, 로그 블록들 그리고 저널 커밋 블록으로 이루어진다. 저널 디스크립터 블록은 트랜잭션에 대한 정보와 로그 블록들이 실제로 써져야 하는 위치 등의 정보를 갖는다. 로그 블록들은 수정할 메타데이터들이다. 저널 커밋 블록은 해당 트랜잭션이 완벽히 스토리지의 저널 영역에 쓰였다는 것을 나타내는 블록이다. 저널 커밋 블록이 쓰여야만 해당 트랜잭션이 커밋되었다고 한다.

파일시스템에는 여러 상태의 트랜잭션이 존재한다. 파일시스템의 메타데이터에 대한 수정으로 인하여 저널링이 필요한 블록들이 삽입되는 트랜잭션을 running 트랜잭션이라 한다. 주기적으로 또는 호스트가 fsync()함수를 호출하면 running 트랜잭션은 committing 트랜잭션이 된다. Committing 트랜잭션 상태에서 트랜잭션에 포함된 블록들이 저널 영역에 써지고 마지막으로 저널 커밋 블록이 저널 영역에 써진다. 저널 커밋 블록까지 스토리지에 써지면 해당 트랜잭션은 committed(checkpoint) 트랜잭션이 된다. committed 트랜잭션은 바로 체크포인트 되지 않고 저널 영역이 가득차면 한번에 체크포인트 된다. 따라서 저널 영역에 여러개의 committed 트랜잭션이 존재할 수 있다.

2.2 EXT4 저널링

EXT4는 ordered-mode를 기본 저널링 모드로 한다. Ordered-mode의 저널링 과정은 다음과 같다. 1) 쓰기 요청과 관련한 데이터 블록들을 디스패치 시킨 후 2) 저널 디스크립터 블록과 로그 블록에 대한 쓰기 요청을 디스패치시킨다. 3) 호스트가 쓰기 요청을 디스패치를 시키면 스토리지는 쓰기 요청받은 블록들을 DMA를 통해 스토리지 캐시까지 이동시킨다. 4) 마지막으로 트랜잭션의 커밋이 완료되었다는 것을 표시하기위해 저널 커밋 블록에 대한 쓰기 요청을 디스패치 시킨다. 이때 저널 커밋 블록에 대한 쓰기 요청에 REQ_FLUSH와 REQ_FUA 플래그를 붙인다. REQ_FLUSH 플래그는 저널 커밋 블록을 스토리지 캐시로 이동시키기 전에 스토리지 캐시를 flush하여 저널 디스크립터 블록과 로그 블록이 항상 저널 커밋 블록보다 먼저 스토리지에 써지도록 한다. REQ_FUA 플래그는 저널 커밋 블록이 스토리지 캐시를 거치지 않고 바로 스토리지에 써지도록 한다.

EXT4에서는 커밋중인 트랜잭션의 커밋이 완료되야 다음 트랜잭션을 커밋시킬 수 있다. 따라서 running 트랜잭션과 commiitting 트랜잭션은 각각 하나씩 존재한다.

2.3 BFS 저널링

BFS는 블록을 영구적으로 쓰지 않고도 쓰기 요청 간의 순서를 지킬 수 있다. 따라서 선행된 쓰기 요청이 스토리지에 써지기 전에 다음 쓰기 요청을 발생하여 파이프

라이닝 방식으로 저널링을 할 수 있다.

BFS에서도 default 저널링 모드로 ordered-mode를 사용하지만 그 방식이 다르다. 쓰기 요청이 스토리지에 써지는 순서를 보장할 수 있다는 장점을 활용하기 위해 BFS는 듀얼 모드 저널링을 사용한다. 듀얼 모드 저널링은 commit 스레드가 디스패치를 맡고 flush 스레드가 스토리지 캐시를 flush하는 역할을 맡아 두 작업이 파이프라이닝 방식으로 진행되도록 하는 저널링 방식이다. BFS에서는 DMA를 통해 블록이 스토리지 캐시까지 써지는 것을 기다릴 필요가 없다. 따라서 저널 디스크립터 블록, 로그 블록, 저널 커밋 블록에 대한 쓰기 요청을 DMA를 통한 블록 이동을 기다리지 않고 순서대로 디스패치 시킨다. 또한 BFS에서는 committing 트랜잭션의 커밋을 기다린 후 running 트랜잭션을 커밋시킬 필요가 없다. 선행하는 트랜잭션이 저널 커밋 블록에 대한 쓰기 요청을 디스패치 했다면 해당 트랜잭션이 스토리지에 쓰여지지 않아도 다음 트랜잭션을 커밋시킬 수 있다. 즉 여러 트랜잭션을 병렬적으로 커밋시킬 수 있다는 장점이 있다.

BFS는 호스트가 쓰기 요청 간의 순서만을 보장하도록 요청하면 flush 스레드를 깨우지 않고, commit 스레드가 스토리지에 디스패치만 요청한 후 return한다. 호스트가 쓰기 요청이 디스크에 써지도록 보장하도록 요청한 경우에만 flush 스레드를 깨워 트랜잭션이 스토리지까지 써지는 것을 기다리고 return한다.

3. 본 문

3.1 Journal Conflict(저널 충돌)

트래잭션을 이용한 파일시스템에서 creat(), delete(), fsync()와 같은 함수들을 호출하면 변경할 메타데이터의 저널헤드를 러닝 트랜잭션에 삽입한다. 이때 삽입하고자 하는 저널 헤드가 이미 커밋 중인 트랜잭션에 포함된 상황을 저널 충돌이라고 한다.

트랜잭션은 저널 헤드들을 연결리스트로 관리하고 저널 헤드의 prev와 next 포인터는 하나밖에 없다. 따라서 저널 충돌을 무시하고 저널 헤더를 러닝 트랜잭션에 삽입하면 연결리스트에 문제가 생겨 로그 블록이 상실된다.

3.2 EXT4에서의 저널 충돌

대표적인 트랜잭션을 이용한 파일시스템인 EXT4에서 저널충돌이 발생했을 때 가능한 상황과 처리는 다음과 같다.

1) 저널 헤드의 b_next_transaction이 running 트랜잭션을 가리킨다.: 이 경우는 저널 헤드가 이미 러닝 트랜잭션의 일부이므로 추가적인 작업을 할 필요가 없다.

2) 1번 경우가 아니지만 frozen_data가 존재한다.: frozen_data는 저널 헤드의 데이터를 백업하기 위한 버퍼이다. 이 경우 저널 헤드의 b_next_transaction 포인터가 running transaction을 가리키도록 한다.

3) 1번과 2번에 해당하지 않고 저널헤드의 frozen_data 없이 관련한 블록이 스토리지에 써지고있다.: 디스크 IO가 완료되기를 기다린다.

4) 1,2,3번에 포함되지 않는다.: 이 경우에는 저널 헤드에 대한 frozen_data를 만든 후 2번의 경우처럼 행동한다.

3번과 4번의 경우 러닝 트랜잭션에 저널 헤드를 삽입하려던 스레드가 기다리거나 추가적인 작업을 해야한다. 따라서 EXT4에서 저널 충돌이 많이 발생하면 성능이 감소한다.

3.3 BFS에서의 저널 충돌

트랜잭션을 병렬적으로 커밋시키는 파일시스템인 BFS에서 저널 충돌을 생각해보면 문제가 더 심각해진다. BFS에서는 쓰기 요청 간의 순서를 보장할 수 있기 때문에 여러개의 트랜잭션을 병렬적으로 커밋시킬 수 있다. 따라서 running 트랜잭션은 여러 committing 트랜잭션과 저널 충돌이 발생할 수 있다. BFS는 저널 충돌이 발생하면 저널 헤드를 running 트랜잭션의 t_jh_wait_list에 삽입하고 이 리스트가 비기 전까지 running 트랜잭션의 커밋을 미룬다. 이때 최악의 경우 선행하는 모든 트랜잭션이 커밋될 때까지 기다려야 한다. 이에따라 BFS의 장점인 트랜잭션의 병렬성이 사라지기 때문에 저널 충돌은 멀티 트랜잭션 파일 시스템인 BFS의 확장성에 심각한 악영향을 끼친다.

4. 실 험

4.1 실험환경

표 1: 논문의 실험환경

CPU	I7-3770(8 core)
실험용 ssd	850PRO(256GB)
리눅스 버전	3.10.61(BarrierFS, EXT4)

4.2 저널 충돌 블록의 타입 분석

표 2: 기존 varmail 워크로드에서 60초 동안 저널 충돌 횟수 분석. T*i*: 스레드, S: 수퍼블록, G: 그룹 디스크립터 테이블, DB: 데이터 비트맵, IB: 아이노드 비트맵, IT: 아이노드 테이블, D: 데이터 블록.

	S	G	DB	IB	IT	D
T1	4	946	4443	4003	17279	1998
T2	6	918	4580	3992	17214	1819
T3	0	923	4674	3953	17045	1929
T4	8	939	4543	4138	17047	1930
T5	2	948	4631	3980	17004	1785
T6	4	900	4687	4002	16932	1954
T7	0	976	4562	3963	17166	1886
T8	6	970	4693	4067	17194	1872
Total	30	7520	36814	32098	136881	15173

표 3: varmail를 수정한 워크로드에서 60초 동안 저널 충돌 횟수 분석. T*i*, S, G, DB, IB, IT, D는 <표 2>와 같다..

	S	G	DB	IB	IT	D
T1	0	2477	4664	0	8491	0
T2	0	2516	4695	0	8260	0
T3	0	2469	4787	0	8633	0
T4	0	2491	4659	0	8453	0
T5	0	2486	4727	0	8443	0
T6	0	2554	4544	0	8703	0
T7	0	2589	4706	0	8470	0
T8	0	2446	4728	0	8422	0
Total	0	20027	37510	0	67875	0

BFS에서 워크로드에 따라 저널 충돌이 발생한 블록의 유형과 스레드별 저널 충돌의 횟수를 분석하였다. 실험은 filebench의 varmail 워크로드를 사용했다. varmail 워크로드는 하나의 디렉토리 내부의 1000개의 16KB 파일에 대해서 60초 동안 여러개의 스레드가 delete, creat, appendwrite, read, fsync를 반복하기 때문에 메타데이터의 수정이 많다.

<표 2>는 varmail 워크로드를 그대로 사용하여 8개의 스레드로 실험했을 때 저널 충돌 횟수이다. 전체 저널 충돌 횟수는 228516번 발생하였다. varmail 워크로드에는 creat와 delete 그리고 appendwrite가 있기 때문에 모든 블록의 타입에서 저널 충돌이 발생한다는 것을 확인 할 수 있었다.

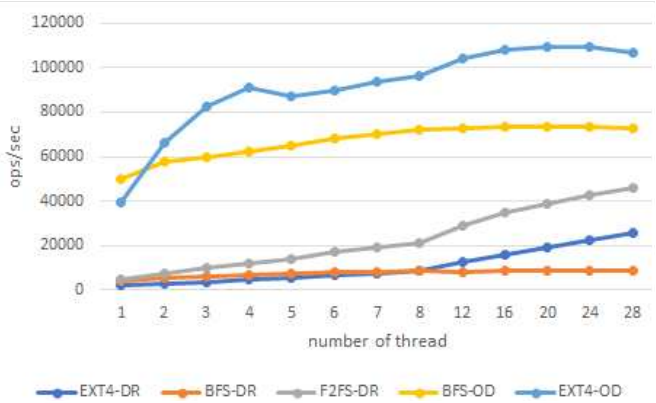
<표 3>은 varmail 워크로드를 수정하여 creat와 delete를 제거하였다. 그 결과 전체 저널 충돌 횟수는 125412번 발생하였고, 아이노드의 생성과 삭제가 발생하지 않기 때문에 슈퍼블록과 아이노드 비트맵에서 저널 충돌이 발생하지 않았다. 또한 데이터 블록에서의 저널 충돌도 발생하지 않았다.

두 실험의 성능을 비교해보면 수정하지 않은 varmail 워크로드로 실험했을 때 8260ops/s 의 성능을 보였고, 수정한 varmail 워크로드로 실험했을 때 12637ops/s의 성능을 보였다. 이를 통해 각 실험에서 operation의 저널 충돌 비율을 (전체 저널 충돌 횟수)/(60초 동안 실행된 operation의 수)로 계산해보면 수정하지 않은 varmail 워크로드는 44.5%의 operation에서 저널 충돌이 발생했으며, 수정한 varmail 워크로드는 16.5%의 operation에서 저널 충돌이 발생하였다. 이를 통해 저널 충돌의 횟수와 파일시스템의 성능에 반비례 관계가 있다는 것을 확인할 수 있다.

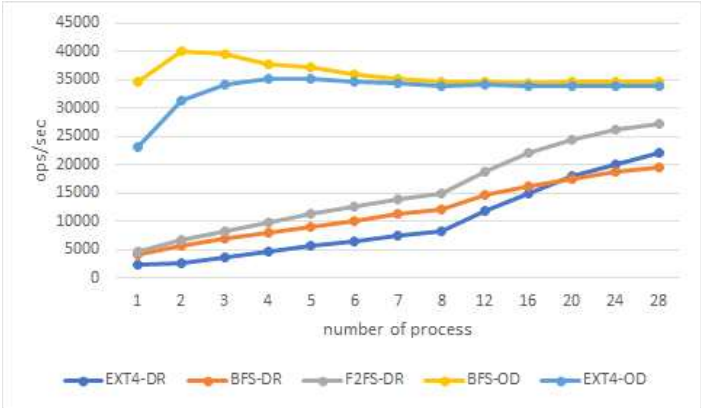
4.3 저널 충돌이 확장성에 미치는 영향 분석

<그래프 1>과 <그래프 2>는 각각 varmail워크로드와 수정된 varmail 워크로드를 스레드 개수를 증가시키면서 실험한 결과이다.

그래프 1: 기존 varmail 워크로드에서 스레드 개수 증가에 따른 성능분석. DR: durability를 보장, BFS-OD: 쓰기 요청의 순서만 보장, EXT4-OD: 마운트할 때 nobarrier 옵션을 사용하여 BFS-OD와 비슷하게 만들었다.



그래프 2: 수정한 varmail 워크로드에서 스레드 개수 증가에 따른 성능분석. DR, BFS-OD, EXT4-OD: <그래프 1>과 같다.



기존 varmail 워크로드는 메타데이터의 수정이 많으므로 저널 충돌이 많이 발생한다. 이에따라 <그래프 1>에서 BFS의 확장성이 전혀 없음을 확인할 수 있다. 저널충돌은 EXT4에서도 발생하기 때문에 F2FS에 비해서 성능이 떨어지는 것을 확인할 수 있다.

반면에 <그래프2>의 실험결과는 수정한 varmail 워크로드에서 메타데이터의 수정이 적으므로 저널 충돌이 적게 발생한다는 것을 보인다. 위의 그래프에서 가장 눈에 띄는 점은 BFS의 확장성이 좋아졌다는 것이다. 또한 BFS와 EXT4의 성능 증가했음을 확인할 수 있다. 하지만 여전히 저널 충돌이 발생하기 때문에 F2FS에 비해 성능이 떨어진다.

5. 결 론

저널 충돌이 모든 블록의 유형에서 발생할 수 있으며 저널 충돌이 많이 발생할수록 트랜잭션을 이용한 파일시스템의 확장성과 성능이 감소한다는 것을 확인하였다. 또한 저널 충돌은 트랜잭션을 병렬적으로 사용하는 BFS에서 더 심각한 성능저하를 가져온다. 따라서 멀티 코어 환경에서 파일시스템의 확장성과 성능을 위해 저널 충돌을 해결해야할 필요가 있다.

6. 사 사

본 연구는 한국연구재단 기초연구실 지원사업(No. 2017R1A4A1015498), 정보통신기술진흥센터 지원사업 [R7117-16-0232, 32Gbps 데이터 서비스를 위한 익스트림 스토리지 입출력 기술 개발], 그리고 IITP 전문연구실 지원사업(2018-0-00549)의 지원을 받아 수행되었음.

7. 참 조

[1] Stephen C.Tweedie. "Journaling the Linux ext2fs filesystem". Proceedings of the 4th Annual LinuxExpo, 1998.

[2] Cao, M., Bhattacharya, S., and Tso, T. "EXT4: The next generation of ext2/3 filesystem". In proc. Of Linux Storage & Filesystem Workshop(2007).

[3] Youjip Won, "Barrier Enabled IO Stack for Flash Storage", in Proc. of USENIX FAST 2018, Oakland, CA, USA, Feb. 12-15, 2018