리눅스 커널 버전 별 ext4 fsync() 동작 분석

손성배○ 노연진 원유집
한양대학교
{afireguy, magyun, yjwon}@hanyang.ac.kr

Analysis on the ext4 fsync() operation according to Linux Kernel version

Seongbae Son○ Yoenjin Noh Youjip Won
School of Computer Software, Hanyang University

요 약

Ext4 파일시스템은 리눅스를 기반으로 하는 데스크톱 및 임베디드 시스템에 널리 사용되고 있고, 동작 분석 및 성능 향상에 대한 방법들이 많이 연구되었다. 하지만, 버퍼를 활용한 입출력을 사용하는 Ext4에서 특정 파일에 대한 즉시 내구성을 요구하는 fsync 시스템 콜을 자세히 분석한 연구는 찾아보기 힘들다. fsync()는 즉시 내구성이 보장 될 때 반환되는 동기적 방식을 사용하는 시스템 콜이므로 fsync의 지연시간은 어플리케이션의 성능과 밀접한 관련이 있다. fsync()의 개선을 위해서는 커널 버전 별 fsync() 동작의 차이점을 이해하는 것이 중요하다. 본 논문에서는 리눅스 커널 버전별 Ext4 fsync() 동작의 차이점과 장단점을 분석한다.

1. 서 론

Ext4[1]는 현재 리눅스 커널의 기본 파일시스템으로 적용되어 있고, 높은 성능으로 안드로이드 기반의 스마트폰과 같은 모바일 기기나 데스크톱 컴퓨터에 널리 사용되고 있다.

Ext4는 파일시스템의 일관성을 위해 저널링을 지원하고[2], JBD2(journaing block device2) 데몬이 저널링을 담당한다. 저널링은 시스템 장애가 발생 했을 때, 일관성을 유지하기 위한 방법이다. 휘발성인 DRAM을 버퍼로 활용한 입출력은 갑작스런 시스템 장애가 발생 시 버퍼링된 쓰기가 스토리지에 적용되지 않을 수 있다. 파일시스템의 메타데이터와 데이터 중 한 정보라도 스토리지에 적용이 누락된다면, 파일 시스템은 잘못된 정보를 가지게 되고 이 상태를 일관성이 깨졌다고 정의한다. 저널링에서는 파일 연산이 일어날 때 발생하는 쓰기를 실제 스토리지의 위치에 기록하기 전에, 쓰기의 복사본을 저널 영역에 먼저 기록한다. 이 기록 과정을 커밋(commit)이라고 한다. 시스템 장애가 일어나게 되면 저널 영역에 커밋된 변경사항이 담긴 기록을 읽어 실제 스토리지의 위치에 기록하는 간단한 복구 과정을 통해 일관성을 유지할 수 있다. 현재의 저널링은 효율적인 쓰기를 위해 여러 파일 연산의 결과를 모아서 한꺼번에 기록하는 컴파운드 트랜잭션을 사용한다.

Ext4 저널링은 write-back, ordered, data 세 개의 모드를 지원한다. Ext4 저널링에서 기본으로 사용되는 ordered 저널 모드는 오직 메타데이터만 저널 영역에 기록하며, 메타데이터가 저널 영역에 기록되기 전에 트랜잭션과 연관된 유저 데이터를 먼저 디스크에 기록한다. 트랜잭션 자료구조에는 여러 속성 중 메타데이터 리스트와 아이노드 리스트가 존재한다. 메타데이터 리스트는 변경된 메타데이터를 리스트로 관리하고, 아이노드 리스트는 트랜잭션에 포함된 파일의 아이노드를 리스트로 관리한다. 순차 모드일 경우 JBD2 데몬은 트랜잭션의 아이노드 리스트를 참조하여 트랜잭션에 포함된 파일의 데이터가 디스크에 기록될 때까지 메타데이터를 저널에 기록하지 않고 기다린다.

버퍼를 활용한 입출력[3]을 사용하는 Ext4에서 fsync 시스템 콜은 특정 파일에 대한 즉시 내구성을 보장해 준다. 하지만 즉시 내구성이 보장 될 때 반환되는 동기적 방식을 사용하는 시스템 콜이므로 fsync()의 지연시간은 어플리케이션의 성능과 밀접한 관련이 있다. 따라서 fsync() 지연시간 개선을 통해 어플리케이션의 응답성을 향상시킬 수 있고, 이러한 개선을 위해서는 커널 버전 별 fsync() 동작의 차이점을 이해하는 것이 중요하다.

본 논문에서는 스마트폰에서 널리 사용된 리눅스 커널 버전인 3.4.0부터 최신 커널인 4.7.4까지 버전 별 Ext4 fsync()의 동작과, 다른 IO 관련 시스템 콜과의 상관관계를 분석한다. 커널 3.4.0 버전 이후 3.8.0과 4.6.2 버전에

서 fsync() 동작에 변화를 주는 패치가 적용되어 본 논문에서는 동작의 차이가 있는 위의 세 개 버전을 중심으로 fsync()를 분석하였다.

본 논문의 구성은 다음과 같다. 2장에서는 fsync() 동작을 분석한다. 3장에서는 커널 버전 별 fsync() 동작의 차이점을 기술한다. 4장에서 커널 버전별 fsync()의 성능 측정 결과를 보인다. 마지막으로 5장에서는 본 논문의 결론을 맺는다.

2. fsync() 동작 분석

Ext4 파일시스템의 경우 fsync 시스템 콜이 호출되면 VFS에 의해 ext4_sync_file() 함수가 호출된다. ext4_sync_file() 함수가 파일의 데이터와 메타데이터를 저널 및 디스크에 기록하는 전반적인 작업을 수행한다.

ext4_sync_file() 함수는 fsync()의 인자로 주어진 파일의 데이터를 디스크에 기록하고, 메타데이터를 현재 동작중인(running) 트랜잭션에 삽입하며, 해당 파일의 아이노드 상태를 JBDDirty로 변경한다. 또한, 해당 파일의 아이노드를 현재 동작중인 트랜잭션의 아이노드 리스트에 삽입하고, 메타데이터를 저널에 기록하기 위해 JBD2 데몬을 깨운다.

JBD2 데몬이 깨어나면 fsync()의 인자로 주어진 파일의 메타데이터가 포함된 트랜잭션의 아이노드 리스트를 순회하여 각 아이노드에 해당하는 파일의 데이터를 디스크에 동기적으로 기록한다. 데이터가 현재 기록 중이거나 이미 디스크에 기록되어 데이터가 클린(clean)한 상태라면 추가로 디스크에 데이터를 기록하지 않는다. 데이터의 기록이 끝나면 트랜잭션에 포함된 메타데이터를 저널에 기록한다. 메타데이터가 모두 기록되면 커밋(commit) 블록을 저널에 기록하여 해당 트랜잭션의 커밋을 완료하고, 트랜젝션에 포함된 메타데이터의 상태를 더티(dirty)로 변경한다. 이렇게 더티 상태로 변경된 메타데이터들은 flush 쓰레드에 의해 실제 디스크에 기록되게 된다.

리눅스 커널 버전에 따라 fsync()된 실행된 결과와 다른 파일 연산의 결과가 동일한 트랜잭션에 포함될 수도 있고, 되지 않을 수도 있다. 또한, write() 동작 시 새로운 블록 할당 유무에 따라 fsync()된 파일이 포함된 트랜잭션의 구성이 달라질 수 있다.

3. 커널 버전 별 fsync() 동작의 차이점

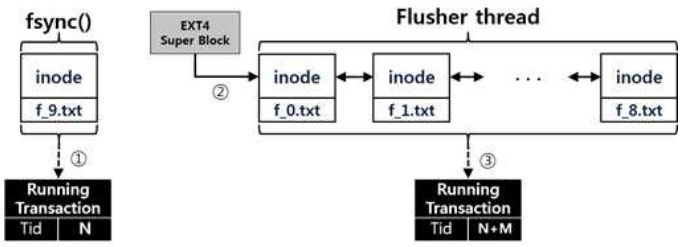
3.1. 커널 3.4.0

리눅스 커널 3.4.0 버전에서는 write() 동작 중에 새로운 블록을 할당 받는 경우 [그림1]과 같이 해당 파일의 아이노드를 현재 running transaction의 아이노드 리스트에 삽입하지 않고, 새로운 블록을 할당 받지 않는 경우에 [그림2]와 같이 아이노드가 추가된다.

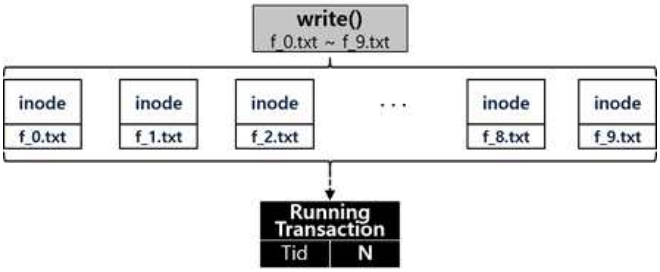
fsync()가 호출되면 이전의 write() 동작 중 새로운 블록을 할당 받은 경우 아이노드를 running transaction의 아이노드 리스트에 추가하지 않는다. fsync() 처리가 끝나면 fsync()된 파일의 데이터는 디스크에 기록되고, 메타데이터가 포함된 트랜잭션은 저널 영역에 기록된다. 이후 flush 쓰레드가 깨어나면 write() 동작에 의해 변경된 더티한 데이터를 디스크에 기록하고, 새로운 running transaction을 생성하여 해당 파일들의 메타데이터와 아이노드를 각각 running transaction의 메타데이터 리스트와 아이노드 리스트에 추가한다. 즉, write()된 파일과 fsync()된 파일이 서로 다른 트랜잭션에 포함되게 되고, 트랜잭션의 구성 순서는 [그림1]의 ①~③과 같다.

반면에 fsync()가 호출될 때 이전의 write() 동작 중 새로운 블록을 할당 받지 않은 경우에는 write() 동작 중에 변경된 파일의 메타데이터와 아이노드를 각각 running transaction의 메타데이터 리스트와 아이노드 리스트에 삽입하기 때문에, [그림2]와 같이 fsync()된 파일이 write()된 파일과 동일한 트랜잭션에 포함되게 된다. 동일한 트랜잭션에 fsync()된 파일과 write()된 파일이 모두 들어 있으므로, JBD2 데몬에 의해 fsync()된 파일의 메타데이터를 저널에 기록하기 전에 write()된 파일의 데이터가 디스크에 동기적으로 기록되고, fsync()된 파일과 write()된 파일의 메타데이터가 한 번에 저널에 기록된다.

따라서, 커널 3.4.0 버전은 write() 동작 중 새로운 블록을 할당 받지 않는 경우 write()된 파일의 데이터도 JBD2 데몬에 의해 동기적으로 디스크에 기록되기 때문에, fsync()의 지연시간이 길어지는 단점이 있다.



[그림1] 커널 3.4.0 fsync()의 아이노드 삽입(블록 할당 O)

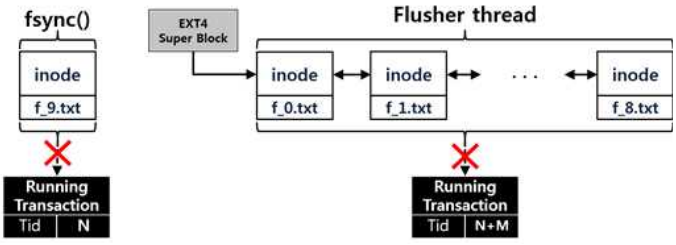


[그림2] 커널 3.4.0 fsync()의 아이노드 삽입(블록 할당 X)

3.2. 커널 3.8.0

커널 3.8.0 버전부터 *"ext4:remove calls to ext4_JBD2_file_inode() from delalloc write path by Jan Kara"*[1] 패치가 적용되었다. 해당 패치에서 write() 동작 또는 flush 쓰레드에 의해 변경된 파일의 아이노드가 트랜잭션의 아이노드 리스트에 삽입되는 함수를 제거하면서, [그림3]과 같이 write()된 파일의 아이노드가 트랜잭션에 추가되지 않는다. 그러므로 fsync()가 처리될 때 이전의 write() 동작에서 새로운 블록을 할당 받지 않은 경우 JBD2 데몬에 의해 write()된 파일이 동기적으로 디스크에 기록되지 않아 fsync() 지연시간이 짧아졌다. 하지만, write()된 파일의 메타데이터는 트랜잭션의 메타데이터 리스트에 삽입되지 않기 때문에 write()된 파일의 메타데이터는 저널 영역에 기록되지 않는다.

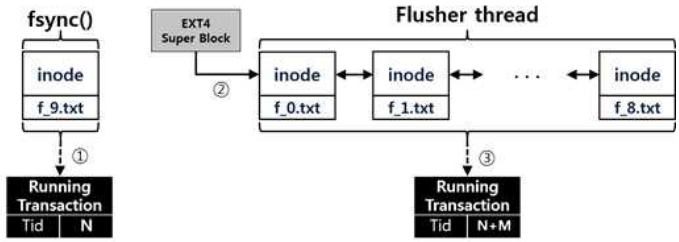
write() 동작 중 새로운 블록을 할당 받는 경우에는 flush 쓰레드에 의해 write()된 파일의 메타데이터가 트랜잭션에 추가되어 저널영역에 기록되지만, 아이노드는 트랜잭션의 아이노드 리스트에 삽입되지 않는다. 따라서 JBD2 쓰레드가 파일의 메타데이터를 저널에 기록하기 전에 데이터가 디스크에 기록 되었는지 여부를 확인할 수 없게 되어 ordered가 보장되지 않을 수 있다.



[그림3] 커널 3.8.0 fsync()의 아이노드 삽입

3.3. 커널 4.6.2

커널 3.8.0 버전부터 4.6.1 버전까지 ordered가 보장되지 않는 문제를 해결한 *"ext4: Fix data exposure after a rash by Jan Kara"*[2] 패치가 4.6.2 버전부터 적용되었다. 파일의 데이터 블록을 할당하는 함수에 변경된 파일의 아이노드를 트랜잭션의 아이노드 리스트에 추가하도록 변경되었고, 블록 할당 함수는 flush 쓰레드에 의해 호출된다. 따라서 [그림4]와 같이 새로운 블록을 할당 받지 않는 경우에도 fsync()된 파일과 write()된 파일은 서로 다른 트랜잭션에 포함되며, JBD2 데몬에 의해 write()된 파일의 데이터가 디스크에 기록되지 않고, 오직 flush 쓰레드에 의해 데이터가 기록된다. 하지만 커널 3.8.0 이전 버전과는 달리 write() 동작 중에 파일의 아이노드를 트랜잭션의 아이노드 리스트에 삽입하는 코드는 추가되지 않았기 때문에, write() 시 새로운 블록을 할당 받지 않는 경우 write()된 파일의 메타데이터는 저널 및 디스크에 기록되지 않는다.



[그림4] 커널 4.6.2 fsync()의 아이노드 삽입

4. 실 험

4.1 실험 환경

실험은 Intel(R) i7-4770(3.4GHz), 12GB DRAM, HDD 500 Gbyte, 운영체제는 Ubuntu 14.04 LTS 환경으로 구성된 시스템에서 상기 기술한 커널 별로 실험을 진행하였다.

4.2 실험 방법

본 실험은 fsync()의 동작을 확인하기 위해 1, 10, 100, 1000개의 파일을 개방한 후 각 파일에 대해 2KB의 write 동작을 수행하고, 마지막으로 개방한 하나의 파일에 대해서만 fsync()를 호출하는 테스트 프로그램을 사용하였다. 각 실험은 블록 할당이 이루어지는 경우와 그렇지 않은 경우를 나누어서 실험하였다. 실험 결과는 fsync() 작업의 지연 시간과 fsync() 작업에 의해 내려가는 Data 블록의 개수를 성능 평가의 기준으로 삼아 실험을 진행하였다.

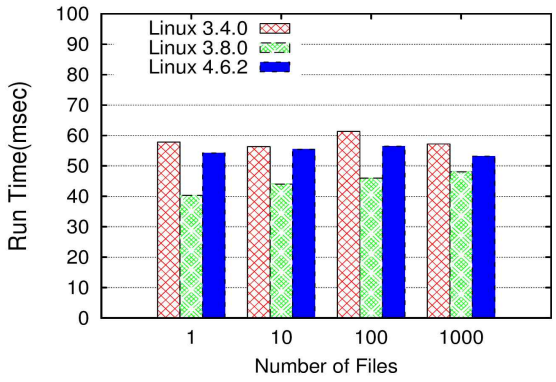
4.3 실험 결과

4.3.1 fsync() 지연시간

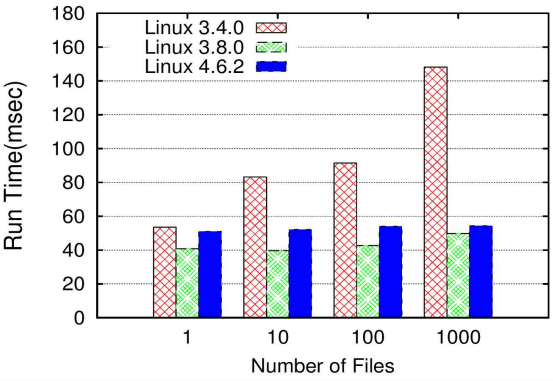
[그림5]는 커널 버전 별 테스트 프로그램의 지연 시간 결과를 나타낸다. write() 시 블록 할당이 일어난 경우, 모든 버전에서 파일의 크기에 상관없이 fsync()의 지연시간이 일정한 것을 확인할 수 있다. 이는 블록 할당이 일어나는 경우에는 파일의 개수에 상관없이 fsync()의 인자로 주어진 파일만을 처리하기 때문이다.

3.8.0 커널은 다른 두 개의 버전에 비해 최소 10%에서 최대 30% 정도 빠른 처리 시간을 보여주고 있다. 이는 3.8.0에서 트랜잭션의 아이노드 리스트를 순회하여 더티 페이지를 찾는 작업을 진행하지 않기 때문이다. 4.6.2 커널의 경우 3.8.0 버전보다는 성능이 떨어지지만, 안정성을 보장하기 위해 아이노드를 트랜잭션에 넣었음에도 불필요한 더티 데이터 탐색을 제거하여, 3.4.0 버전 대비 최소 5%에서 최대 10%의 성능 향상을 보여준다.

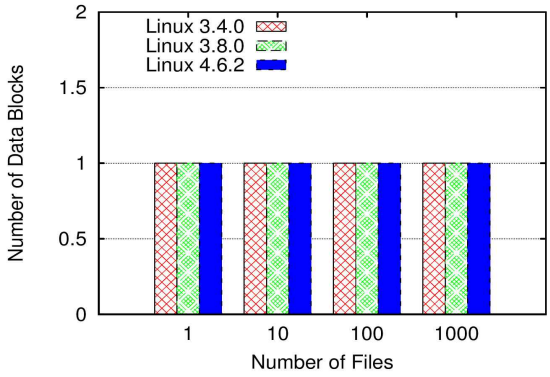
[그림6]은 커널 버전 별 write() 시 블록 할당이 일어나지 않는 경우에 대한 결과를 나타낸다. [그림5]와 달리 3.4.0의 경우 파일의 개수가 증가할수록 fsync()의 작업 시간이 최소 10%에서 최대 40% 가량 증가하는 것을 볼 수 있다. 3.4.0 커널에서는 write() 동작 중 새로운 블록을 할당 받지 않는 경우 아이노드를 트랜잭션에 삽입하여, fsync()에서 JBD2 데몬이 write()된 파일의 데이터를 디스크에 기록할 때까지 대기하기 때문이다.



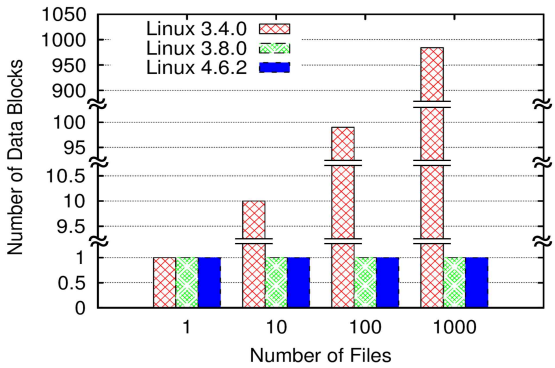
[그림 5] 버전 별 fsync() 지연시간 (블록 할당 O)



[그림 6] 버전 별 fsync() 지연시간 (블록 할당 X)



[그림7] 버전 별 fsync()에 의해 기록된 블록 수(블록 할당 O)



[그림8] 버전 별 fsync()에 의해 내려간 블록 수 (블록 할당 X)

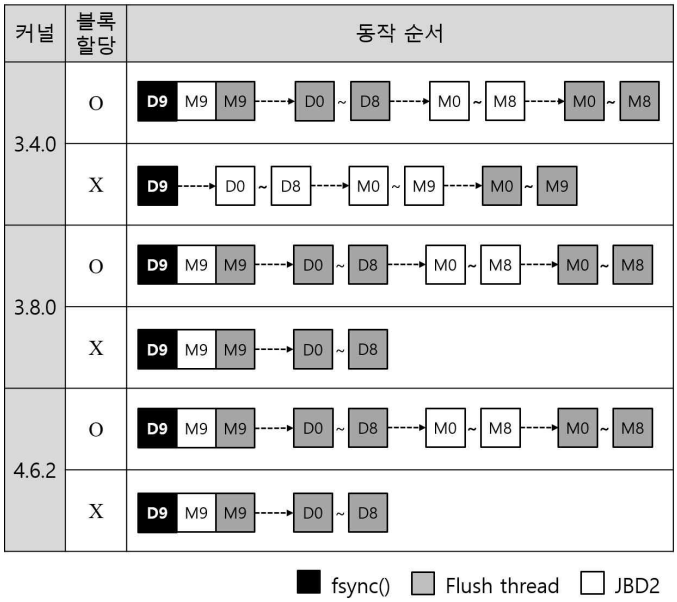
4.3.2. fsync()에 의해 기록된 데이터 블록 수

[그림7]은 Write() 작업 중 블록 할당이 일어나는 경우, fsync() 시스템 콜에서 디스크에 기록하는 데이터 블록의 개수를 나타낸 그래프이다. 모든 버전에선 동일하게 한 블록의 데이터가 fsync()에 의해 디스크에 기록하며, 이는 파일의 개수와 관계없이 항상 fsync()에 인자로 주어진 파일에 대해서만 데이터 블록이 내려가기 때문이다.

[그림8]은 write() 작업 중 블록 할당이 일어나지 않는 경우에 fsync()를 통해 디스크에 기록된 데이터 블록의 개수를 나타낸다. 3.8.0 버전과 4.6.2 버전은 [그림7]과 같은 결과를 나타내며, 이를 통해 마지막 파일의 데이터만 fsync()에 의해 내려가는 것을 알 수 있다. 하지만, 3.4.0 버전의 경우 [그림7]과 달리 파일의 개수가 증가할수록 fsync() 작업에서 디스크에 기록되는 데이터 블록의 개수가 증가하는 것을 확인할 수 있다. 이는 write()된 파일들의 아이노드가 트랜잭션의 아이노드 리스트에 삽입되었기 때문이다. 즉, fsync() 함수에서 트랜잭션을 커밋할 때 트랜잭션에 존재하는 모든 아이노드의 더티 데이터를

디스크에 내리는 작업을 진행하였기 때문이다.

4.3.3. 테스트 프로그램 동작 순서



[그림9] 버전 별 테스트 프로그램 동작 순서

[그림9]는 커널 버전 별 10개 파일에 대한 테스트 프로그램의 실행 결과를 나타낸다. 3.4.0 ~ 3.7.10 버전은 write() 시 새로운 블록을 할당 받는 경우 fsync() 처리 함수에서 fsync()된 파일9의 데이터가 디스크에 기록되고, 파일9의 메타데이터가 JBD2 데몬에 의해 저널에 기록된다. flush 쓰레드가 깨어나면 파일9의 메타데이터와 write()된 파일0 ~ 파일8의 9개 데이터가 디스크에 기록된다. 이후 JBD2 데몬에 의해 파일0 ~ 파일8의 메타데이터가 저널에 기록되고, flush 쓰레드가 깨어나면 파일0 ~ 파일8의 메타데이터가 디스크에 기록된다. write() 동작 중에 새로운 블록을 할당 받지 않는 경우에는 fsync() 처리 함수에서 fsync()된 파일을 디스크에 기록한다. JBD2 데몬이 깨어나면 write()된 파일0 ~ 파일8의 데이터를 디스크에 기록하고, 파일0 ~ 파일9의 메타데이터를 저널에 기록한다. 이후 flush 쓰레드에 의해 파일0 ~ 파일9의 메타데이터가 디스크에 기록된다.

커널 3.8.0 ~ 4.6.1 버전에서 write() 시 새로운 블록을 할당 받는 경우에는 3.4.0 ~ 3.7.10 버전에서 새로운 블록을 할당 받는 경우와 동작이 같다. 하지만 write() 시 새로운 블록을 할당 받지 않는 경우 write()된 파일의 메타데이터가 저널 및 디스크에 기록되지 않는다.

커널 4.6.2 ~ 4.7.4 버전은 3.8.0 ~ 4.6.1 버전에서의 테스트 프로그램 출력 결과와 동일한 것을 확인 할 수 있다.

5. 결론

본 논문에서는 리눅스 커널 3.4.0 ~ 4.7.4 버전의 fsync() 동작을 분석하였다. fsync()의 동작을 확인하는 실험을 통해 write() 시 새로운 블록의 할당 유무에 따라 테스트 프로그램의 실행 결과가 다르게 나타난 것을 확인 할 수 있었다. 커널 3.8.0 이전 버전에서는 write() 동작 중에 새로운 블록을 할당하지 않는 경우 write()된 파일의 데이터가 JBD2 데몬에 의해 디스크에 기록되어 fsync()의 지연시간이 커지는 단점이 있다. 커널 3.8.1 ~ 4.6.1 버전은 write()된 파일의 데이터가 JBD2 데몬에 의해 기록되지 않아 fsync()의 지연시간이 낮아졌다. 하지만, JBD2 데몬이 메타데이터를 저널에 기록하기 전에 flush 쓰레드에 의해 데이터가 기록될 때까지 기다릴 수 없게 되어 ordered가 보장되지 문제가 발생할 수 있다. 이러한 ordered 모드의 결함을 해결하기 위해 커널 4.6.2 버전부터 변경된 파일의 아이노드가 flush 쓰레드에 의해 트랜잭션에 추가되도록 변경되었다.

참고 문헌

[1] Mathur, Avantika, et al. "The new ext4 filesystem: current status and future plans." Proceedings of the Linux symposium. Vol. 2. 2007.

[2]Tweedie, Stephen C. "Journaling the Linux ext2fs filesystem." The Fourth Annual Linux Expo. 1998.

[3] Pai, Vivek S., Peter Druschel, and Willy Zwaenepoel. "IO-Lite: a unified I/O buffering and caching system." ACM Transactions on Computer Systems (TOCS) 18.1 (2000): 37-66.

[4] “ext4: remove calls to ext4_jbd2_file_inode() from delalloc write path”. January 11, 2016, from http://marc.info/?l=git-commits-head

[5] “[1/2]ext4: Fix data exposure after a crash”. November 12, 2012, from https://patchwork.ozlabs.org/patch/565741/