QCOW2 가상 디스크에서 증폭된 sync 연산의 지연 감소 기법

김명선 ○ 원유집
한양대학교 컴퓨터·소프트웨어 학과
audtjs9139@gmail.com, yjwon@hanyang.ac.kr

Delay Reduction Technique for Amplified Sync Operation of QCOW2 Virtual Disk

Myeongseon Kim○ Youjip Won
Department of Computer and Software, Hanyang University

요 약

QCOW2는 QEMU 가상머신에서 사용되는 가상 디스크 이미지로 Copy-on-Write, 스냅샷과 같은 유용한 기능을 제공하는 장점이 있다. 하지만 QCOW2에는 Guest OS에서 *fdatasync()*를 호출했을 때 QEMU에서 최소 3배 이상의 *fdatasync()*가 호출되는 sync 증폭으로 인해 성능이 저하되는 문제를 가지고 있다. 본 논문에서는 sync 증폭으로 인한 성능 저하 문제를 해결하고자 QEMU에 BFS에서 제공하는 *fdatabarrier()*를 적용하였다. 기존 QEMU 대비, latency는 최대 3.6% 감소하였고 throughput은 최대 81.1%까지 증가하여 성능 향상을 보였다.

1. 서 론

QCOW2(QEMU Copy-on-Write)[3]는 QEMU 가상머신에서 사용되는 가상 디스크 이미지이다. QCOW2는 Copy-on-Write를 통해 원본 이미지의 내용에 영향을 미치지 않고 다른 이미지에서 변경한 내용을 저장할 수 있고, 스냅샷을 통해 원본 이미지에서 변경된 내용들을 메타데이터에 기록하여 특정 시각에 저장된 내용을 복원할 수 있는 장점들을 가진다.

그러나 QCOW2에는 치명적인 단점[1]이 존재한다. QCOW2를 사용하는 Guest OS에서 *fdatasync()*를 호출하면 QEMU에서 최소 3배 이상의 *fdatasync()*가 호출되는 sync 증폭이 발생하여 성능이 저하된다. 이는 가상 디스크를 관리하기 위한 추가적인 메타데이터들을 실제 디스크에 쓸 때 순서를 보장해야 하기 때문이다. QCOW2는 클러스터의 주소를 가지는 L2 테이블과 L2 테이블의 주소를 가지는 L1 테이블, 그리고 각 클러스터들의 스냅샷을 관리하기 위한 refcount 테이블을 메타데이터로 가진다. QEMU는 이러한 추가적인 메타데이터들의 쓰기 순서를 보장하기 위해 메타데이터를 쓸 때마다 *fdatasync()*를 호출하고 마지막으로 불필요한 *fdatasync()*까지 호출한다. 만약 저널링으로 인해 저널 메타데이터의 쓰기 요청까지 발생하면 sync 증폭은 두 배 이상 증가한다.

본 논문에서는 QCOW2에서 증폭된 sync 연산들의 지연시간을 감소시켜 sync 증폭으로 인한 성능 저하를 해결하고자 QEMU에 *fdatabarrier()*를 적용하였다.

2. 배 경

QCOW2의 sync 증폭 문제를 해결하기 위해 Qingshu Chen은 per virtual disk internal journaling[1] 기법을 제안하였다. 이 기법은 가상 디스크 내부에 저널 영역을 만든다. 그리고 Guest OS에서 *fdatasync()*를 호출할 때 발생하는 추가적인 메타데이터들(L2 테이블, refcount 테이블)을 하나의 트랜잭션으로 취급하여 QCOW2의 저널 영역에 쓰고 트랜잭션의 checksum이 포함된 저널 커밋 블록을 끝에 추가한 뒤, 체크포인트한다. 이 기법은 sync 연산을 트랜잭션당 하나로 감소시켰다. 하지만 per virtual disk internal journaling의 단점은 가상 디스크에 항상 저널링을 위한 공간이 할당되어 있어야 하고, 저널 영역과 원 위치에 각각 한 번씩 총 두 번의 쓰기가 발생한다는 점이다.

본 논문에서는 가상 디스크에 추가적인 공간과 추가적인 쓰기없이 증폭된 sync 연산들의 지연시간을 감소시켜 sync 증폭으로 인한 성능 저하를 해결하기 위해 QEMU에 *fdatabarrier()*를 적용하였다.

3. *fdatabarrier()*를 적용한 QEMU

QEMU는 Guest OS에서 *fdatasync()*가 호출되어 갱신된 메타데이터들을 순서대로 실제 디스크에 쓸 때마다 *fdatasync()*를 호출하고 마지막에 불필요한 *fdatasync()*를 한 번 호출한다. 본 논문에서는 QEMU에서 메타데이터들을 순서대로 실제 디스크에 쓰기위한 *fdatasync()*를 *fdatabarrier()*로 수정하고, 마지막에 호출되는 불필요한 *fdatasync()*는 데이터와 메타데이터들이 실제 디스크에 완전히 쓰여지는 것을

보장하기 위해 그대로 사용하였다. Host OS는 QEMU에서 호출되는 *fdatabarrier()*를 사용할 수 있도록 BFS를 적용하였다.

*fdatabarrier()*는 BFS(BarrierFS)[2]에서 제공하는 시스템 콜이다. *fdatabarrier()*를 사용하는 순서 보장 메커니즘은 *fdatasync()*를 사용하는 것 보다 더 효율적이다. BFS에서는 쓰기 요청이 발생한 데이터가 durable하게 될 때까지 기다릴 필요 없이 순서를 보장할 수 있기 때문이다.

4. 성능 평가

본 논문에서는 *fdatabarrier()*를 적용한 QEMU의 성능을 평가하였다. 성능을 평가하기 위한 실험에는 YCSB(0.12.0)[4], Filebench(1.5 alpha 3), Mobibench(1.0.8)[5], GNU coreutils의 dd 명령어를 사용하였다. QEMU는 2.1.2 버전을 사용하였고 실험 PC의 스펙은 표 1, 실험 구성은 표 2와 같다.

표 1. PC 스펙

CPU	Intel® Core™ i5-2500 CPU @ 3.30GHz
RAM	4G
Storage	256GB (Samsung 850 PRO)
OS	Host: Linux 3.10.61 (Ext4, BFS) Guest: Linux 3.10.61 (Ext4)

표 2. 실험 구성

구성	Host	Guest	Sync operation
QEMU	Ext4	Ext4	*fdatasync()*
QEMU(Barrier)	BFS	Ext4	*fdatabarrier()*

4.1 YCSB

YCSB에서는 Insert 연산의 비율을 1로 수정한 Insert 워크로드 와 Update 연산의 비율을 1로 수정한 Update 워크로드를 사용하였다. 두 워크로드는 64KB의 레코드를 5GB만큼 Insert하거나 Update하도록 설정하였다. YCSB에 사용된 DB는 Cassandra (3.11.1)이다.

그림 1과 같이 *fdatabarrier()*를 적용한 QEMU의 throughput이 기존의 QEMU에 비해 Insert에서 3.3% 증가하였고 Update에서 0.6% 증가하였다. 그림 2에서는 latency가 Insert에서 3.6% 감소하였고 Update에서 0.5% 감소한 결과를 볼 수 있었다.

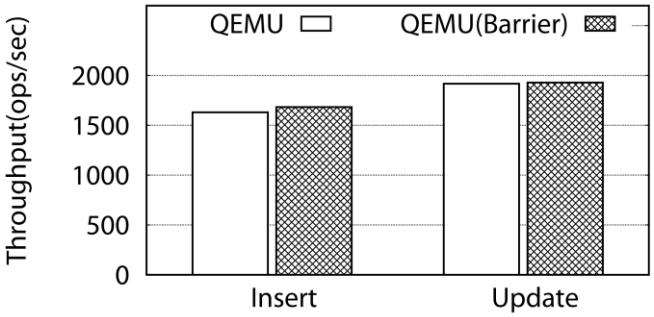

그림 1. YCSB 결과(Throughput)

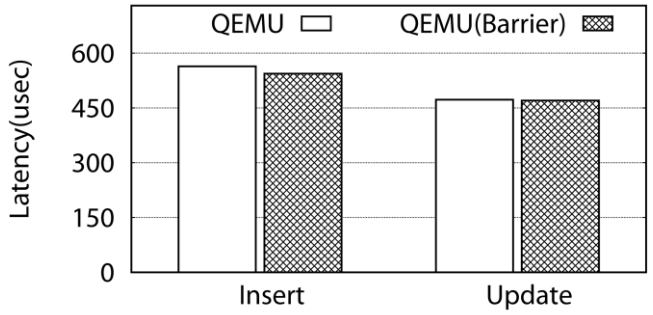

그림 2. YCSB 결과(Latency)

4.2 Filebench

Filebench에서는 varmail 워크로드에서 iosize를 64KB, 실행 시간을 180초로 수정하여 실험을 진행하였다.

*fdatabarrier()*를 적용한 QEMU의 throughput이 기존의 QEMU에 비해 9.2% 증가한 결과를 볼 수 있었다.

4.3 Mobibench

Mobibench에서는 append 실험과 overwrite 실험을 나눠 진행하였다. append는 파일을 새로 생성하여 sequential write로 진행하였고, overwrite는 미리 만들어진 파일에 random write하여 진행하였다. 두 실험은 64KB의 데이터 쓰기를 요청하고 *fdatasync()*를 호출하는 과정을 총 5GB를 채울 때까지 반복하였다.

그림 3과 같이 *fdatabarrier()*를 적용한 QEMU의 throughput이 기존 QEMU와 비교하여 append에서 4.7% 증가하였고, overwrite에서 2.7% 감소하였다.

4.4 GNU dd

GNU dd 명령어에서는 파일을 새로 생성하는 append 실험과 이미 존재하는 파일에 덮어쓰는 overwrite 실험으로 나눠 진행하였다. 두 실험은 64KB의 데이터 쓰기를 요청하고 sync 연산을 수행하는 과정을 총 5GB를 채울 때까지 반복하였다. overwrite에서는 'conv=notrunc' 옵션을 추가하여 파일을 새로 쓰지않고 overwrite 할 수 있도록 하였다.

그림 3과 같이 *fdatabarrier()*를 적용한 QEMU의

throughput이 기존 QEMU와 비교하여 append에서 81.1% 증가하였고, overwrite에서 4.5% 증가하였다.

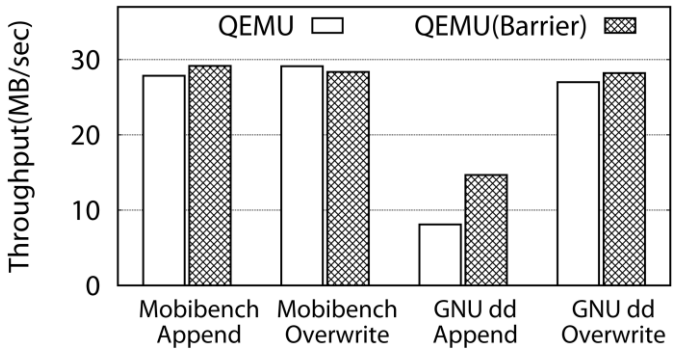

그림 3. Mobibench, GNU dd 결과

5. 결론 및 향후 연구

본 논문에서는 QCOW2의 sync 증폭으로 인한 성능 저하를 해결하기 위해 QEMU에 *fdatabarrier()*를 적용하였다. *fdatabarrier()*를 적용한 QEMU의 성능을 측정한 결과에서 최대 81.1%에서 최소 0.6%까지의 성능 향상을 확인할 수 있었지만 Mobibench의 overwrite에서는 오히려 성능이 2.7% 감소한 것을 확인하였다.

앞으로 Mobibench의 overwrite에서 성능이 감소한 원인을 명확히 분석할 것이다. 그리고, 현재는 Host OS에만 적용한 BFS를 Guest OS에도 적용하여 Guest OS에서 *fdatabarrier()*를 사용할 수 있도록 연구할 것이다. 이를 위해서는 Guest OS에서 *fdatabarrier()*를 호출 했을 때 QEMU에서 *fdatabarrier()*를 인지하고 처리할 수 있도록 수정이 필요하다.

참 고 문 헌

[1] Qingshu Chen et al., “Mitigating Sync Amplification for Copy-on-write Virtual Disk”, in Proc. of FAST, 2016.

[2] Youjip Won et al., “Barrier-Enabled IO Stack for Flash Storage”, in Proc. of FAST, 2018.

[3] Mark McLoughlin, The QCOW2 Image Format, https://people.gnome.org/~markmc/qcow-image-format.html, 2008

[4] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with ycsb”, in Proc. of the 1st ACM SoCC, 2010, pp. 143-154

[5] Sooman Jeong, Kisung Lee, Jungwoo Hwang, Seongjin Lee, Youjip Won, “AndroStep: Android Storage Performance Analysis Tool”, in Proc. of the 1st EWME, 2013