파일 시스템 별 Direct I/O 동작 방식 분석 및 성능 평가

박동일 ○ 원유집

한양대학교 컴퓨터·소프트웨어학과

ldoitlpg@hanyang.ac.kr, yjwon@hanyang.ac.kr

Analysis of Direct I/O Implementation and Performance

Dongil Park○ Youjip Won

Department of Computer and Software, Hanyang University

요 약

파일 시스템은 다양한 종류의 workload에 대해 높은 성능을 보장해야 한다. 일반적으로 사용되는 buffered I/O 방식은 page cache를 활용하여 데이터가 저장장치에 저장되는 주기를 조절 함으로서 성능 향상의 이점을 얻고 있다. 하지만 buffered I/O와는 달리 유저 공간의 데이터를 page cache를 사용하지 않고 저장 장치에 직접 저장하는 Direct I/O는 DBMS와 가상 머신 에서 주로 사용되고 있으며 성능에 민감한 workload이다. 본 논문에서는 동작 방식이 다른 4가지 파일 시스템의 direct I/O 성능을 측정하고 파일 시스템 별 구현 내용을 확인하였다. 성능 평가는 direct I/O를 사용한 순차 쓰기 방식과 임의 쓰기 방식에 대해 4KB부터 4MB의 레코드 크기를 사용하여 진행하였다. 플래시 메모리의 특성을 활용한 파일 시스템인 F2FS의 경우 direct I/O 쓰기 기능이 구현되어 있지 않았지만 최근 구현된 in-place-update 방식의 direct I/O 성능은 타 파일 시스템 대비 뛰어난 성능을 보여주었다.

1. 서 론

파일 시스템(file system)의 유저 데이터 저장방식은 크게 두 가지로 구분된다. 페이지 캐시(Page cache)를 사용하여 주기적으로 데이터를 반영하는 Buffered I/O와 페이지 캐시를 사용하지 않고 직접 저장 매체에 저장하는 Direct I/O가 있다. Buffered I/O 방식은 기본 입출력 방식이지만 페이지 캐시를 직접 관리하거나 데이터가 임시 저장장소에 저장되는 것을 필요로 하지 않는 DBMS 또는 가상 머신 환경[1]에서는 direct I/O를 사용해야 한다. 또한 buffered I/O 방식은 가용 메모리를 기준으로 데이터를 주기적으로 저장하는 반면 direct I/O 경우 매 I/O 요청 시 마다 데이터를 저장해야 하기 때문에 그 성능이 매우 중요하다. 본 논문에서는 파일 시스템 동작방식에 따른 direct I/O 구현 방식과 그 성능을 확인하였다.

파일 시스템은 동작 방식에 따라 크게 세가지 방식으로 구분된다. 첫 번째 방식은 유저 데이터가 저장되는 위치가 항상 같은 파일 시스템 블록에 저장되는 In-place-update의 파일 시스템이며 대표적으로 EXT4[2] (Fourth extended file system) 파일 시스템이 있다. 두 번째 방식은 유저 데이터가 쓰여지는 시점에 데이터가 복사되어 저장되는 copy-on-write 방식의 파일 시스템이며 대표적으로 BTRFS[3] (B-tree file system)가 있다. 마지막으로 저장 매체를 하나의 큰 Log로 간주하여 유저 데이터를 로그의 마지막에 저장하는 로그 기반의 파일 시스템이 있으며 대표적으로 NILFS2[4] (New Implementation of a Log-structured File System) 와 플래시 메모리에 최적화된 F2FS[5] (Flash-Friendly File System)가 있다.

EXT4 파일 시스템은 Linux에서 가장 대중적으로 사용되는 파일 시스템이며 저널을 사용하여 유저 데이터에 대한 consistency를 보장하고 있다.

2. 실험 설계

각 파일 시스템의 직접 쓰기 성능 측정을 위해 아래와 같은 환경에서 실험을 진행하였다. 3.6Ghz의 Intel i7-3820 Quad core CPU와 삼성 DDR3 SDRAM PC3 12800 4GB 메모리를 4개 사용하여 총 16GB 메모리를 사용하였고 OS는 64bit 버전의 Ubuntu 12.04에서 Kernel 3.13-rc2 버전을 사용하였다. I/O 성능을 측정하기 위한 Storage device는 삼성전자의 840 Pro 128GB SSD를 사용하였으며, Benchmark tool은 MobiBench[5]를 사용하여 레코드 크기(record size)를 4KB 부터 4MB까지 증가시키며 순차 쓰기와 임의 쓰기에 대한 성능을 각 5회 측정하였다.

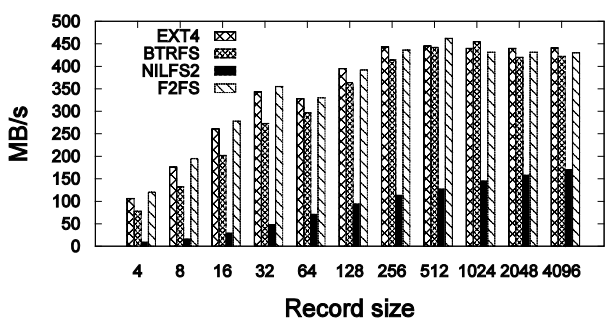

그림 1. 순차 쓰기 성능 비교

3. 실험 결과 및 분석

그림 1은 EXT4, BTRFS, NILFS, F2FS 파일 시스템의 순차 쓰기 성능을 나타낸 그래프이다. 실험 결과 EXT4와 F2FS가 대부분의 레코드 크기에서 높은 성능을 보였으며 레코드 사이즈가 512KB 이하의 경우 F2FS가 가장 높은 성능을 보이며 그 이상의 레코드 크기에서는 EXT4가 F2FS보다 높은 성능을 보여주었다. BTRFS의 경우 레코드 크기가 1024KB일 때 약 455MB/s의 높은 성능을 보였지만 F2FS 대비 평균 87%의 낮은 성능을 보였다. NILFS2의 경우 F2FS 대비 평균 22%의 성능을 확인하였고 실험에 사용된 전체 파일 시스템 중 가장 낮은 성능을 보였다.

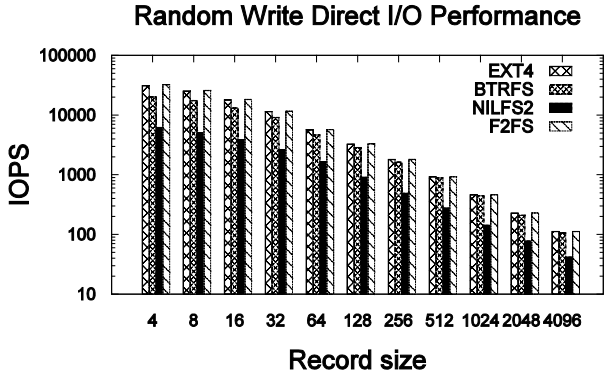

그림 2. 임의 쓰기 성능 비교

그림 2는 각 파일 시스템의 임의 쓰기 성능을 나타낸 그래프이다. 그래프의 y축은 임의 쓰기 결과를 초당 IO의 개수로 표현하였다. 임의 쓰기의 경우에도 F2FS가 4개 파일 시스템 중 가장 높은 성능을 보였으며 순차 쓰기와는 달리 전체 레코드 크기에서 EXT4보다 높은 성능을 보였다. EXT4는 평균적으로 F2FS의 98% 성능을 보였으며, BTRFS는 F2FS 대비 약 82%의 성능을 보였다. 순차 쓰기와 마찬가지로 NILFS2는 F2FS대비 약 27% 성능으로 가장 낮은 성능을 보였다.

4. 파일 시스템 별 구현 방식

일반적으로 direct I/O는 I/O를 수행하려는 목적 파일을 open()할 때 O_DIRECT flag를 사용하여 해당 파일에 대한 I/O에 대해 direct I/O를 수행하도록 요청한다. 이후 read(), write() 시스템 콜에 의해 I/O가 요청되면 해당 파일의 flag를 확인하여 O_DIRECT flag의 유무에 따라 buffered I/O 또는 direct I/O를 수행한다. direct I/O는 파일 시스템의 동작 방식에 따라 구현되어 각 파일 시스템의 address_space_operation 구조체의 direct_IO 함수에 등록된다. 일반적으로 각 파일 시스템의 direct_IO 함수는 VFS (virtual file system)에서 제공하는 direct I/O를 위해 제공하는 라이브러리 함수인 blockdev_direct_IO()를 사용한다. 이 함수는 유저 영역의 페이지를 bio 구조체의 bio vector에 저장하고, 해당 페이지를 위한 파일 시스템의 블록 할당을 요청한다. 각 파일 시스템의 get_block() 함수는 파일 시스템의 동작 방식에 따라 블록 할당을 하여 요청된 유저 페이지를 저장할 파일 시스템의 블록을 할당한다. 파일 시스템의 블록 할당이 완료되면 유저 영역의 페이지를 저장하고 있는 bio 구조체가 submit되어 파일 시스템 상의 블록에 최종적으로 저장된다.

EXT4 파일 시스템은 저널 파일 시스템 이지만 direct I/O를 수행하는 경우에는 제한적으로 저널을 사용한다. direct I/O를 수행하여 파일 크기가 기존 크기보다 커지는 경우에 새로 할당되는 블록에 대해 저널을 사용하며, 만약 크기가 변하지 않는 경우 저널은 사용되지 않는다. 또한 데이터가 갱신되는 경우 in-place-update 파일 시스템의 동작 방식을 사용하여 기존 파일 시스템 블록에 다시 쓰여지게 된다.

BTRFS의 경우 in-place-update 방식의 EXT4와는 다르게 데이터가 갱신 되는 경우 copy-on-write 방식의 동작을 통해 새 파일 시스템 블록을 사용한다. 이는 임의 쓰기 방식임에도 데이터를 순차적으로 쓰기 때문에 성능상 이득을 얻을 수 있다. BTRFS와 마찬가지로 log 기반의 파일 시스템인 NILFS2와 F2FS는 유저 공간의 데이터를 저장할 때 로그의 마지막 위치에 데이터가 저장 되어 항상 새로운 파일 시스템 블록이 사용된다. 하지만 NILFS2의 경우 direct_IO의 쓰기 기능이 구현되어 있지 않기 때문에 direct I/O를 수행하려는 데이터들은 buffered I/O 방식으로 처리된다. 이는 direct I/O를 사용하는 유저의 의도와는 상반되는 결과로서 사용자의 주의를 요한다. F2FS의 경우 NILFS2와 마찬가지로 direct I/O의 쓰기 기능이 구현되어 있지 않았지만, 현재 실험에 사용한 커널 3.13-rc2에 direct I/O기능이 구현되었다. F2FS에 구현된 direct I/O는 타 파일 시스템 대비 뛰어난 성능을 보이는 반면 F2FS의 log 기반의 특성을 버리고 EXT4와 같이 in-place-update 방식의 I/O를 수행하도록 구현되었다.

5. 결론

본 논문에서는 최신 Linux Kernel에서 사용되고 있는 대표적인 파일시스템인 EXT4, BTRFS, NILFS2, F2FS의 direct I/O 성능을 평가하고 그 동작 방식을 확인해 보았다.

플래시 메모리의 고유 특성을 이해하고 이를 활용한 파일 시스템인 F2FS의 경우 direct I/O 쓰기 기능이 구현되어 있지 않았으나, 최근 in-place-update 방식으로 구현된 direct I/O의 경우 실험에 사용된 타 파일 시스템 대비 뛰어난 성능을 보여주었다. Log 기반

파일 시스템의 특성과는 상반되는 동작 방식이지만 순차 쓰기와 임의 쓰기 모두 뛰어난 성능을 보였다. 하지만 in-place-update를 통해 기존 파일 시스템 블록을 재사용 함으로서 log 기반의 파일 시스템이 가지고 있는 큰 장점 중에 하나인 recovery의 기능을 잃게 되었으며 이를 개선하기 위해 log 기반의 direct I/O 방식을 사용하는 것도 성능과 recovery 를 모두 보장 할 수 있는 좋은 방법이 될 수 있을 것으로 기대된다.

Acknowledgment
본 연구는 지식경제부 및 한국산업기술평가관리원의 산업원천 기술개발사업(정보통신)의 일환으로 수행하였음. [No.10041608, 차세대 메모리 기반의 스마트 디바이스용 임베디드 시스템 소프트웨어]

참고 문헌

[1] Dingding Li; Xiaofei Liao; Hai Jin; Bingbing Zhou; Qi Zhang, "A New Disk I/O Model of Virtualized Cloud Environment," Parallel and Distributed Systems, IEEE Transactions on , vol.24, no.6, pp.1129,1138, June 2013
[2] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, “The new ext4 filesystem: current status and future plans,” in Proceedings of the Linux Symposium, vol. 2. Citeseer, 2007, pp. 21–33.
[3] C. Mason, “The Btrfs Filesystem,” Oracle Corporation, Tech. Rep., 2007.
[4] R. Konishi, Y. Amagai, K. Sato, H. Hifumi, S. Kihara, and S. Moriai, “The linux implementation of a log-structured file system,” SIGOPS Oper. Syst. Rev., vol. 40, no. 3, pp. 102–107, Jul. 2006.
[5] J. Kim, “F2FS,” 2012. [Online]. Available: https://www.kernel.org/doc/Documentation/filesystems/f2fs.txt
[6] S. Jeong, K. Lee, J. Hwang, S. Lee, and Y. Won, “Framework for Analyzing Android I/O Stack Behavior: From Generating the Workload to Analyzing the Trace,” Future Internet, vol. 5, pp. 591–610, 2013