KISM Smart Media Journal
Vol. 4, No. 2 pp. 62~67 ISSN:2287-1322

리눅스 SSD caching mechanism의 성능 비교 및 분석

An performance analysis on SSD caching mechanism in Linux

허상복*, 박진희*, 조희승**
(Sang-Bok Heo, Jinhee Park, Heeseung Jo)

요 약

Hard disk drive(HDD)는 보조 저장장치로서 대부분의 컴퓨터 시스템에서 사용되고 있으나 기계적 특성으로 인하여 성능의 향상 측면에서는 한계점에 도달해 있는 상황이다. 반면, 플래시메모리 기반인 solid state drive(SSD)는 고성능 저소음이라는 장점이 있지만, HDD에 비해 높은 가격으로 인해 아직 HDD를 완전 대체하기에는 부담이 크다. 따라서 SSD를 대용량인 HDD의 cache로 사용하는 SSD caching mechanism이 최근 주목을 받고 있다. SSD caching mechanism은 대표적으로 bcache, dm-cache, Flashcache, EnhanceIO 등이 있으며, 각 mechanism들은 구현상의 장단점과 차별성이 존재한다. 본 논문에서는 각 SSD caching mechanism들의 특징을 파악하고, 벤치마크를 통하여 성능을 비교 분석하였다. 이러한 연구를 기반으로 향후 더 성능이 좋은 SSD caching mechanism을 개발하는데 기여할 수 있을 것으로 예상된다.

■ 중심어 : SSD; HDD; cache;

Abstract

During several decades, hard disk drive(HDD) has been used in most computer systems as secondary storage and, however, the performance enhancement of HDD is limited by its mechanical properties. On the other hand, although the flash memory based solid state drive (SSD) has more advantages over HDD such as high performance and low noise, SSD is still too expensive for common usage and expected to take several years to replace HDD completely. Therefore, SSD caching mechanism using the SSD as a cache of high capacity HDD has been highlighted lately. The representatives of SSD caching mechanisms are typically bcache, dm-cache, Flashcache, and EnhanceIO. Each of them has its own internal mechanism and implementation, and this makes them to show their own pros. and cons. In this paper, we analyze the characteristics of each SSD caching mechanisms and compare the performance of them under various workloads. We expect that our contribution will be useful to enhance the performance of SSD caching mechanisms.

■ keywords : SSD; HDD; cache;

Ⅰ. 서 론

마그네틱테이프 저장장치 이후로 HDD는 컴퓨터 시스템에서 보조 저장장치로서 널리 사용되어 왔다. 과거 수년 전 부터 HDD보다 빠른 플래시메모리 기반의 SSD [1]가 나왔으며, SSD는 기계적인 특성으로 인한 성능저하의 문제점을 가지고 있던 HDD의 한계를 극복함으로

* 비회원, 전북대학교 전자정보공학부
** 비회원, 전북대학교 IT정보공학과
본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발 사업의 일환으로 하였음. (No. B0101-15-0644).

접수일자 : 2015년 05월 20일
수정일자 : 2015년 06월 26일
게재확정일 : 2015년 06월 29일
교신저자 : 조희승 e-mail : heeseung@jbnu.ac.kr

써 여러 컴퓨팅 분야에서 HDD를 대체할 차세대 저장매체로써 주목 받았다. SSD는 HDD에 비해 소음이 없으며, 속도가 빠르고, 전력소모/발열이 적지만, 대용량을 쓰기에는 아직 가격의 부담이 크다. 따라서 최근 이러한 부담을 줄이고, HDD와 SSD의 장점들을 이용 할 수 있는 SSD caching mechanism이 각광을 받고 있다. 이것은 대용량인 HDD와 고성능인 SSD를 hybrid로 사용하는 방법이다.

SSD caching mechanism은 대표적으로 bcache[2], dm-cache[3], Flashcache[4], EnhanceIO[5] 등이 있다. 이러한 mechanism들은 dm-cache(3.9)와 bcache(3.10)가 리눅스 커널의 mainline에 포함되면서 주목을 받고 있으며, 여러 회사에서도 SSD caching mechanism에 관심을 보이고 있다. 대표적인 예로, Facebook에서 Flashcache라는 SSD caching mechanism을 자체적으로 개발하여 사용 하고 있다. 이러한 여러 SSD caching mechanism들은 구현상의 장단점과 차별성이 존재한다. 각각의 SSD caching mechanism은 서로 다른 방식으로 구현되었기 때문에, 워크로드와 액세스방법에 따라서 많은 시간의 차이가 나타날 수 있다. 이에 우리는 본 논문에서 각 SSD caching mechanism들이 어떤 워크로드와 액세스방법에 따라 성능의 차이를 보이는지에 대한 분석 결과를 보여준다. 기존 연구에서 각각에 대한 분석 결과를 보여주는 연구결과가 있으나[6][7], 우리는 다양한 mechanism을 비교하여 실험하고자 하였으며, 접근 패턴에 따라 SSD caching mechanism들의 성능 차이를 확인할 수 있다.

Ⅱ. 본 론

1. SSD caching mechanism

SSD caching mechanism은 SSD와 HDD를 하나로 묶어 상위의 계층에 투명하게 하나의 block device로 보여 줄 수 있다. 일반적으로 내부적으로는 모든 read/write operation이 SSD영역에 수행되고 추후 이를 HDD와 동기화 시키는 write-back 방식과 SSD영역에 먼저 수행된 후 추후에 HDD와 동기화 시키는 write-through 방식 두 가지 모드를 기본적으로 제공한다. 또 다른 방식인 write-around은 write-through와 유사한 기술이지만, write I/O는 캐시를 우회하며, 저장장치에 직접 기록하는 방식이다. SSD를 cache로 사용하기 위하여 processor내부의 cache가 동작하는 것과 비슷하게 SSD와 HDD의 영역을 mapping시킨다.

일반적으로 대용량인 HDD를 저장장치로 사용하고, 고성능인 SSD를 캐시장치로 사용한다. 이렇게 생성된 디바이스는 데이터를 읽고 쓰일 때, 저장장치뿐만 아니라, 캐시장치에도 데이터가 읽고 쓰이게 된다. 데이터를 read/write 할 때, 그 데이터가 캐시장치에 존재 할 경우 cache hit가 발생한다. cache hit가 발생하지 않는 경우에는 기존과 동일하게 HDD의 성능과 동일시되며, cach e hit가 발생하는 경우에는 캐시장치로 쓰이는 SSD에 준하는 성능을 보이게 된다.

가. bcache

bcache는 리눅스 블록 계층에서 동작하는 캐시이며, Kent Overstreet에 의해 개발되었다. SSD와 같은 고속 저장장치는 하나 이상의 HDD를 위한 캐시로써 사용되며, 이들은 효과적인 hybrid 볼륨을 생성한다. 기본적으로 순차 I/O를 캐시 하지 않지만, 명령어를 통해 캐시가 가능하고, 데스크톱, 서버, 고성능 저장장치에 적합하다. bcache의 cache mode는 write-back과 write-through뿐만 아니라, write-around를 지원하며, 별도의 formatting 툴을 이용해서 포맷을 한 후에 사용 가능하다. 이러한 작업은 저장장치와 캐시장치 간의 데이터 불일치를 방지하기 위해서지만, 이로 인해, bcache는 볼륨을 생성할 때 저장장치/캐시장치에 포맷을 해야 하므로, 각 장치의 기존의 데이터는 사라진다는 단점이 있다. bcache에서 단일 캐시 장치는 백업장치의 개수만큼 캐시가 가능하다. 또한, 파일시스템에 얽매이지 않으며, write-back mode의 데이터 손실을 방지하기 위한 mechanism이 타 mechanism에 비하여 우수하다고 알려져 있다.

나. dm-cache

dm-cache는 device mapper[8]를 기반으로 구현된 SSD caching mechanism이다. dm-cache에서 지원되는 cache mode는 write-back과 write-through 그리고 pass-through가 있다. pass-through 모드는 캐시 내용이 저장장치와 일관된 것으로 알려지지 않았을 때 유용하는 것으로, 모든 read는 저장장치로부터 제공되고, 모든 write는 저장장치에게 전달된다. dm-cache는 처리해야 할 데이터 양 또는 데이터의 값에 기초하여 블록의 크기와 캐시 용량 조정이 가능하며, 특정 응용프로그램이 순차적인 데이터를 저장할 필요가 없는 경우 bypass 할 수

있다. dm-cache는 다른 mechanism과는 달리 metadata를 저장하기 위해 별도의 공간을 할당하여야 하는 특징이 있다.

다. Flashcache

Flashcache는 2010년 4월에 Facebook에 의해 개발되었으며, 오픈소스로 공개되었다. Flashcache는 write-back, write-through, write-around로 3가지의 cache mode를 제공하며, device mapper의 상단에 내장되어 있고, 이를 사용하여 kernel module을 생성한다. 캐싱이 되어 있을 경우에 read/write hit가 되면 HDD를 접근하지 않기 때문에 빠른 속도를 발휘 할 수 있다. 그러나 Flashcache는 하나 이상의 디스크를 캐싱 할 수 없다는 점과 Flashcache 로드 시에 LVM[9] 사용이 불가하다는 단점이 있다.

라. EnhanceIO

EnhanceIO는 Flashcache에서 재설계된 것으로 dirty data의 양을 정의할 수 있으며, Flashcache 와는 달리 device mapper를 사용하지 않는다. 기존의 HDD에 캐시 장치로 SSD를 사용할 수 있도록 구성되어, EnhanceIO는 사용 중인 device에 캐시 생성 및 삭제가 가능하다는 큰 장점을 가지고 있으며, 주요 기능을 담당하는 모듈과 교체정책을 담당하는 모듈이 분리되어 구성되어 있다. EnhanceIO에서는 순차 I/O bypass 기능이 지원되지 않으며, 지원하는 cache mode는 write-back, write-through, read-only 방식이 있다. read-only 방식은 write의 작업일 경우에는 캐시장치인 SSD를 통하지 않고, 바로 저장장치인 HDD에 쓰이고, read의 작업인 경우에는 SSD에 캐싱되도록 한다.

SSD caching mechanism들의 주요 특징을 표1에 보여준다. 캐싱 정책과 교체 정책은 기본 값을 보여주며, 기본적으로 캐싱정책은 write-back과 write-through, write-around를 제공하지만, dm-cache는 write-around를 제공하지 않고 pass-through 방식을 제공하고, EnhanceIO는 write-around를 제공하지 않고 read-only를 제공한다. 교체 정책은 기본적으로 FIFO, LRU를 지원한다. dm-cache는 multiqueue를 추가로 지원하며, EnhanceIO는 random을 추가로 지원한다.

각 mechanism들은 각 특징이 존재하며, 이러한 특징들은 어떠한 작업을 하냐에 따라 성능이 다르게 나타난다. 우리는 여러 가지 방법으로 테스트를 해보고, 그 결과에 대해 분석해본다.

표 1. 각 SSD caching mechanism의 특징

	bcache	dm-cache	Flashcache	EnhanceIO
리눅스커널 포함 버전	3.10	3.9	not yet	not yet
캐싱 정책	write-through	write-back	write-back	write-through
교체 정책	FIFO	multiqueue	FIFO	LRU
순차 I/O bypass	O	O	X(기능 제공)	X(기능 제거)
장점	여러 개의 HDD 사용 가능	플러그인 캐시 정책	PID 제어	추가적인 장치/포맷 불필요
단점	장치의 포맷	Metadata를 위한 장치 필요	LVM 사용 불가	

표 2. 테스트 환경

HW/SW	Specification
CPU	Inter core i5-2500 3.30GHZ
MEM	4GB
Kernel	Linux 3.16.0 (Ubunutu 14.04)
SSD	840 EVO 120GB
HDD	WDC WD10EZEX-00RKKKA0 1TB

2. 실험

우리가 테스트한 환경은 표2와 같으며, 앞서 설명한

SSD caching mechanism들을 이용하여 벤치마크 테스트를 진행하였다. sysbench 테스트를 위하여 10GB의 SSD와 100GB의 HDD로 hybrid 볼륨을 생성하고, 8GB 워크로드를 sysbench의 옵션을 이용하여 순차적인 write/read와 임의의 write/read 테스트를 진행했다.

그림1은 각각의 mechanism을 sysbench를 통해서 캐싱 되기 전의 순차적인 write/read와 임의의 write/read의 처리량을 그래프(cold 상태)로 표시 한 것이다. 순차적인 write일 경우 bcache가 가장 좋은 성능을 보이고 다른 부분에서도 대체적으로 높은 성능을 보지만, 임의의 read일 경우에는EnhanceIO가 가장 좋은 성능을 보인다.

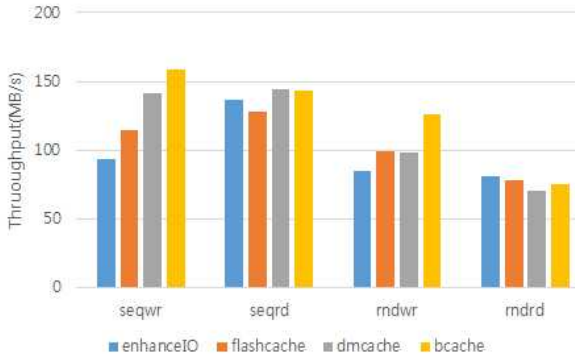


그림 1. 액세스방법에 따른 각 mechanism 처리량(sysbench)

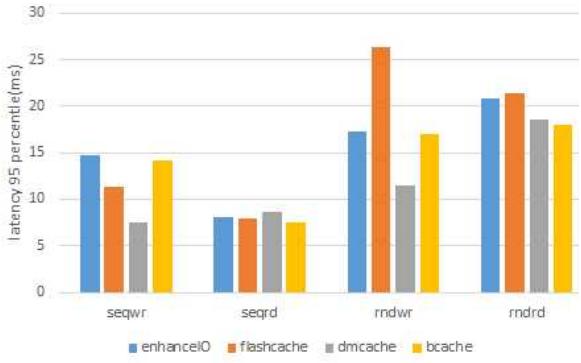


그림 2. 액세스방법에 따른 각 mechanism 지연시간

그림 3. Workload 실행 횟수에 따른 mechanism 처리량

그림2에서는 SSD caching mechanism들의 캐싱의 효과를 보기 위해서, 워크로드를 4번 연속으로 진행하였다. 또한 메모리에서의 캐싱을 피하기 위해, 매 실험에서 페이지 캐시를 비우면서 진행하였다. EnhanceIO, Flashcache의 경우는 횟수를 진행함에 따라 SSD에서 cache hit가 발생하고 처리량이 높아짐을 보여준다. dm-cache(d_seqrd_1, d_rndrd_1)과 bcache(b_seqrd_1, b_rndrd_1)에서는 캐싱의 효과를 충분히 얻지 못함을 알 수 있다. 이는 sysbench의 block 크기를 1M로 테스트를 진행하였는데, 이를 순차 I/O로 인식하여 SSD에 캐싱 시키지 않기 때문이다. 따라서 block 크기를 4K로 설정하여 강제로 캐싱 되도록 한 후 테스트를 다시 진행하였다. 다시 진행한 테스트는 캐싱 효과를 나타내고, dm-cache(d_seqrd_2, d_rndrd_2), bcache(b_seqrd_2, b_rndrd_2)는 이전 테스트보다 우수한 성능을 나타내며, EnhanceIO, Flashcache보다 성능이 약간 더 높음을 연시간이 높음을 보인다.

Latency는 bcache가 순차/임의의 write에서 다른 mechanism에 비해 우수한 것을 볼 수 있다. 또한 EnhanceIO는 임의의 write에서 dmcache에 비해 2.31 지

확인 할 수 있다.

그림 4는 fio 벤치마크를 수행한 결과를 보여주고 있다. fio의 설정은 block size-1M, workload 크기-4G이며, 액세스방법은 순차적인 read/write, 임의의 read/write를 각각 테스트 하였다. 여러 번의 테스트 결과 값의 평균치를 구해서 그래프로 나타냈다. fio 테스트에서는 EnhanceIO와 Flashcache는 캐싱의 효과로 인해 다른 mechanism에 비해 높은 성능을 보인다. 액세스 방법이 read일 경우 그 격차가 크게 나타나고 있는데, 순차적인 read일 경우 성능이 가장 좋지 못한 dm-cache에 비해

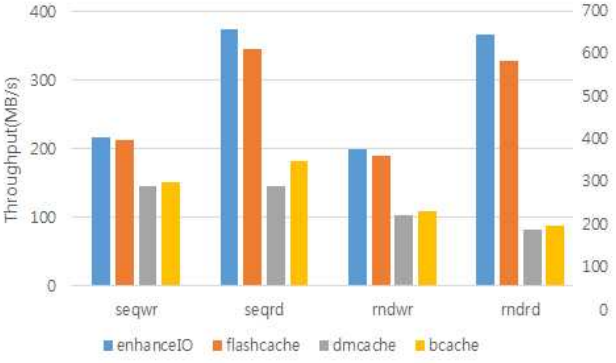


그림 4. 액세스 방법에 따른 각 mechanism 처리량(fio)

EnhanceIO는 2.55배 높은 성능을 보이고, 임의의 read일 경우 dm-cache에 비해 EnhanceIO는 4.49배 높은 성능을 보인다. dm-cache와 bcache가 성능이 낮은 이유는 EnhanceIO와 Flashcache처럼 캐싱의 효과를 보지 못하기 때문이다. 그림 2의 sysbench 벤치마크처럼 fio 벤치마크에서도 두 개의 mechanism은 캐싱이 원활하게 되지 못하고 있다. 그림 2에서는 강제로 캐싱이 되도록 한 후 테스트를 진행 하였지만, fio 벤치마크에서는 강제로 캐싱할 방법을 찾지 못하였다. 유사한 방법으로 시도를 해보았으나, sysbench 벤치마크와는 다르게 강제 캐싱을 하지 못하였다. 이러한 부분은 추후 분석을 통해 알아볼 예정이다.

그림 5에서는 리눅스에서 제공하는 dd 명령어를 이용하여 각 mechanism의 처리량을 확인해 본다. block size를 1M로 테스트를 하였고, 1024의 count 옵션을 주었다. 본 테스트에서도 EnhanceIO와 Flashcache는 높은 성능을 보이고 있으나, dmcache와 bcache에서는 상대적으로 낮은 성능을 보이고 있다. 지금까지 테스트의 내용처럼 비슷한 결과를 보이고 있다. EnhanceIO는 dmcache보다 1.9배 높은 성능을 보인다.

지금까지의 벤치마크의 결과를 보면, EnhanceIO와 Flashcache는 캐싱의 효과로 인해서 SSD에 준하는 성능을 보이는 반면, dmcache와 bcache는 캐싱의 효과를 보지 못해서 대체적으로 낮은 성능을 보인다. 하지만 캐싱이 되었을 경우에는 EnhanceIO와 Flashcache보다 좋은 성능을 보이기도 한다. 벤치마크 테스트할 때의 문제처럼, dm-cache와 bcache는 캐싱이 된 상태에서는 높은 성능을 보이나, 캐싱이 되지 않는 경우가 종종 존재한다. mechanism을 사용할 때 이러한 특징을 고려할

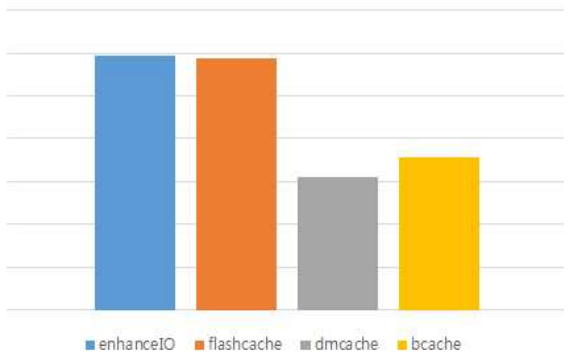


그림 5. DD 명령을 이용한 각 mechanism 처리량

필요가 있다.

Ⅲ. 결 론

본 논문에서는 각각의 SSD caching mechanism들을 sysbench와 fio 벤치마크를 이용하여 액세스방법에 따라 테스트를 수행하였다. 각각의 mechanism들의 특징에 따라 성능이 다른 것을 확인할 수 있었고, 이러한 분석을 통해, 우리는 향후 연구에서 각 mechanism들이 성능 차이를 나타내는 구현상의 문제를 분석하고, 더욱 성능이 좋은 SSD cache mechanism을 개발하고자 한다.

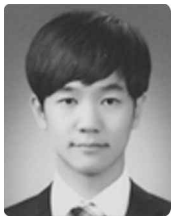
Acknowledgment

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발 사업의 일환으로 하였음. [B0101-15-0644, 매니코어 기반 초고성능 스케일러블 OS 기초연구]

REFERENCES

[1] http://en.wikipedia.org/wiki/Solid-state_drive
[2] Linux, Kernel Documentation, https://www.kernel.org/doc/Documentation/bcche.txt
[3] Linux Kernel Documentation, https://www.kern-el.org/doc/Documentation/devicemapper/cache.txt
[4] Flashcache, http://en.wikipedia.org/wiki/Flashcache
[5] EnhanceIO, https://wiki.archlinux.org/index.php/E-nhanceIO
[6] 송현섭, Dm-cache를 이용한 SSD 캐시의 실험적 평가, 한국정보과학회, 1284-1286, 2013
[7] Christopher, H. et al., The Effect of Flashcache and Bcache on I/O Performance, International Conference on Computing in High Energy and Nuclear Physics, 2013.
[8] Device mapper, http://en.wikipedia.org/wiki/Devic_e_mapper
[9] LVM, http://en.wikipedia.org/wiki/Logical_Volum_e_Manager_(Linux)

저 자 소 개



허상복

2014년 전북대학교 IT정보공학과 학사 졸업.
2014년~현재 전북대학교 전자정보공학부 석사과정

〈주관심분야 : 운영체제, IoT, cache〉

박진희

2015년 전북대학교 IT정보공학과 학사 졸업.
2015년~현재 전북대학교 전자정보공학부 석사과정

〈주관심분야 : 운영체제, gpu〉

조희승

2000년 서강대학교 컴퓨터공학과 학사
2010년 한국과학기술원 전산학과 박사
현재 전북대학교 IT정보공학과 재직
〈주관심분야 : 운영체제, 가상화 시스템, 클라우드, 빅데이터, IoT〉