순서 증가를 인식하는 블록 단위 매핑 기법을 사용하여 로그 블록 방식의 임의 쓰기 성능 개선

안정철O, 신동군O
성균관대학교O
jeongcheol@skku.edu, donkun@skku.edu

Increasing-order Aware Block-Level Mapping Technique for Random Write Performance Enhancement of Log-Block Scheme

Jeongcheol AhnO, Dongkun ShinO
Sungkyunkwan University Department of Electrical and Computer EngineeringO

요 약

최근 플래시 메모리 분야는 반도체 직접 기술의 향상에 따라 비트 당 비용이 감소하고 있으며 이에 따라 제조 업체는 16TB의 대용량 SSD까지 지원할 예정이다[1]. 그러나 플래시 메모리는 물리적 특성 상 주소 변환을 위한 매핑 테이블 사용이 불가피하고 이러한 용량 증가에 따른 테이블 공간 요구량 문제는 현재 해결해야 할 과제이다. 만약 비교적 조밀한 페이지 단위로 테이블을 구성 한다면 16TB 크기의 SSD에 대해서는 16GB(4K-페이지,4B-매핑값)에 달하는 테이블 공간이 필요하다. 요구 기반 페이지 매핑 기법(DFTL)은 테이블의 일부 영역을 RAM에 동적 적재 하는 방식으로 작은 메모리 공간으로 구현 가능하지만 cache miss에 의한 성능 저하가 단점이다. 다른 해결 방식으로 로그 블록 방식은 블록 단위 매핑 을 사용하면서 hot zone에 대해서는 페이지 단위로 매핑되는 로그 블록을 할당하여 쓰기 성능을 보완한다. 공간 요구량이 매우 작기 때문에 대용량 SSD에 적합 할 수 있으나 임의 쓰기 workload에 대해 로그 블록 확보를 위한 병합 비용 때문에 성능이 제약된다. 임의 쓰기에 대한 병합 비용을 줄일 수 있다면 로그 블록 방식은 대용량 SSD에 적합한 방식이 될 수 있다. 본 논문에서는 로그 블록 방식의 병합 비용을 줄 일 수 있는 순서 증가 인식 블록 매핑 기법을 소개한다. 작은 메모리 공간을 사용하여 순서 증가 임의 쓰기가 발생한 블록을 순차 쓰기가 발생한 블록과 같이 매핑하기 때문에 로그 블록 병합에 유리하다.

1. 서 론

현재 플래시 메모리는 물리적 특성에 따른 저전력, 빠른 접근속도 및 높은 내구성을 바탕으로 컴퓨팅 시스템의 비 휘발성 저장장치로 널리 쓰이고 있다. 최근에는 반도체 집적 기술의 향상과 3차원 수직 낸드 플래시 기술이 개발되어 비트당 비용 또한 감소하고 있는 추세이다. 플래시 메모리 제조 업체는 최대 4TB의 대용량 SSD를 공급 하고 있으며 곧 8TB 에서 16TB 용량의 SSD까지 개발될 전망이다[1].

플래시 변환 계층(Flash Translation Layer, 이하 FTL)은 플래시 메모리의 장치 구조적인 제약에 기인하는 하드 디스크와는 다른 특성을 보완 하기 위해 도입되었다. 플래시 메모리의 "쓰기 전 삭제" 특성은 같은 삭제 범위에 속하는 다른 페이지를 보존 하기 위한 추가 입출력을 요구하므로 쓰기 성능을 과도하게 제약할 할 뿐만 아니라 기존 하드 디스크 기반의 호스트 입출력 인터페이스에 호환되지 않는 문제가 있다. FTL은 호스트 주소와 장치 주소간 매핑 테이블을 유지하며 호스트의 입출력 주소를 장치 내 비어 있는 다른 물리 주소로 변환 하여 이러한 문제들을 해결한다.

최근 FTL은 고 수준의 성능을 호스트에 제공하기 위해서 조밀한 단위의 매핑 방식을 주로 쓴다. 그러나 페이지 (읽기/쓰기 단위)와 같이 비교적 조밀한 단위의 매핑을 구현 하기 위한 공간 비용 문제는 현재 해결해야 할 과제 중 하나 이다. 매핑 테이블의 크기는 그 역할을 고려해 볼 때 스토리지 공간에 비례한다. 예를 들어 페이지와 매핑 값의 크기를 각 각 4KB, 4byte로 가정 하면 테이블이 차지하는 공간은 전체 스토리지 용량의 0.1%에 해당한다. 이것은 4TB 크기의 SSD에 대해서는 4GB의 테이블 공간이 필요하며 8TB와 16TB에 대해서는 각 각 8GB, 16GB의 공간이 필요하다는 것을 의미한다. 고 수준의 성능을 위해 테이블 공간을 SRAM 또는 DRAM 상에 적재 하고자 한다면 지나친 메모리 비용과 전력 소모량이 요구될 것이다.

이러한 문제를 해결하기 위해 요구 기반 FTL (Demand-based FTL, 이하 DFTL)[2]이 적용 될 수 있다. DFTL은 페이지 단위 매핑을 유지하면서 모든 매핑 값을 DRAM에 적재하는 대신 매핑 값이 필요할 때 마다 플래시 메모리로부터 동적으로 적재한다. 따라서 DRAM 요구 량이 작을 수 있지만 적재와 갱신을 위한 플래시 접근 비용은 단점이다.

다른 방식으로 로그 블록 방식(log block scheme)[3][4]이 있다. 로그 블록 방식은 데이터 블록에 대해서는 블록 단위로 매핑 하고 페이지 단위로 매핑 되는 제한된 수의 로그 블록들을 데이터 블록들에 할당 하여 데이터 블록에 대한 페이지 쓰기를 로그 블록에 대신 기록 한다. hot zone에 대해서는 페이지 단위 쓰기가 가능 하므로 블록 단위 매핑에 기인하는 취약한 쓰기 성능을 보완 한다. 또한 DFTL과 달리 테이블 공간 요구량이 매우 작아 전체 테이블을 DRAM 상에 유지 할 수 있다. 예를 들어 블록의 크기를 1MB로 가정하면 (블록 당 256 4KB-페이지) 16TB SSD에 대한 테이블 공간 요구량은 228MB에 (1% 로그 공간) 불과하다. 그러나 임의 쓰기 workload 발생 시 로그 블록 확보를 위한 병합 비용은 큰 단점이다. 만약 임의 쓰기 workload에 효율적으로 대응하여 병합 비용을 줄일 수 있다면 로그 블록 방식은 그러한 매핑 테이블 공간 문제의 대안이 될 수 있다.

본 논문에서는 순서 증가 임의 쓰기가 발생한 블록을 별도의 페이지 테이블 없이 적은 공간의 비트맵으로 매핑 가능하게 하는 순서 증가 인식 블록 매핑 기법을 소개한다. 제안 기법은 로그 블록 방식에 적용 될 경우 병합 비용을 줄여 성능을 향상 시킨다.

2. 본 론

2.1 순서 증가 쓰기 workload

현재 호스트 운영체제에는 임의 입출력을 순차 입출력으로 변환하는 많은 기법이 존재 하기 때문에 비록 애플리케이션이 임의 입출력이 수행하더라도 실제 디바이스에는 주소가 정렬되어 전달 된다. 특히 쓰기는 다음 두 가지 특성 때문에 디바이스에 순서가 증가하는 형태로 전달될 가능성이 높다.

1) 쓰기 입출력은 페이지 캐시에 먼저 캐싱이 된 후 플러시 프로세스에 의해 주소 공간 앞부터 차례로 스토리지에 전달된다

2) 입출력 스케줄러는 입출력들을 주소 값에 따라 정렬한다

2.2 순서 증가 인식 블록 매핑 기법

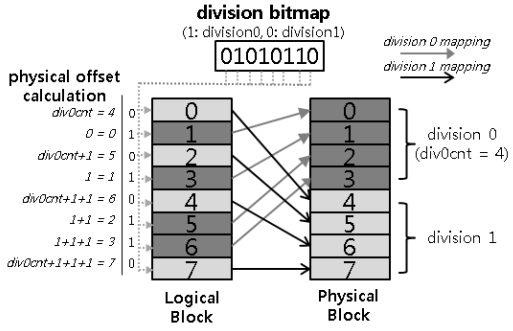

그림 1 순서 증가 인식을 위한 블록 내 증가 오프셋 분할 매핑

그림 1은 순서 증가 인식 블록 매핑 기법을 사용 한 논리 오프셋과 물리 오프셋 간 매핑을 나타낸 것이다. division bitmap은 각 논리 페이지의 division 번호를 나타낸다. 1일 경우 해당 논리 페이지는 division 0에 위치하며 0일 경우 해당 논리 페이지는 division 1에 위치한다. division 내에서의 위치는 비트 카운팅을 통해 알 수 있다. 그림 1에서 논리 페이지 3의 경우 division bitmap에서 비트 오프셋 3 이전의 비트 true의 수가 1이므로 division 0 내에서 오프셋 1에 해당하고 물리 오프셋은 1이 된다. 그림 1에서 논리 페이지 4의 경우 division bitmap에서 비트 오프셋 4 이전의 비트 false의 수가 2이므로 division 1 내에서 오프셋 2에 해당하고 따라서 물리 오프셋은 division 0의 크기를 더한 6이 된다. 그림 2에 이러한 물리 오프셋 계산 과정을 pseudo code로 나타내었다. 만약 쓰기가 순차 쓰기 포함하여 순서 증가 형이 아닐 경우 division bitmap에 의한 분할은 무효화 되며 기존 로그 블록과 같이 페이지 매핑 테이블로부터 물리 오프셋을 얻는다.

1) $_{before/after}N_{true/false}$ ← number of true/false count before/after logical offset
2) test division bitmap at given logical offset
3) If true, Physical offset = $_{before}N_{true}$
4) If false, Physical Offset = $_{before}N_{true}$ + $_{after}N_{true}$ + $_{before}N_{false}$

그림 2 bitmap을 이용한 논리 오프셋에서 물리 오프셋 계산 알고리즘

2.2.1 로그 블록 방식

로그 블록 방식에서 로그 블록 확보를 위한 병합 방식은 두 가지 형태로 나눌 수 있다. 첫 번째 형태는 전체 병합이다. 데이터 블록의 페이지 업데이트가 순차 발생하지 않아 로그 블록 상의 오프셋과 하나라도 일치 하지 않는 경우 전체 페이지들을 정렬하기 위해 새로운 블록에 다시 써야 한다. 다시 쓴 후 매핑 값은 새로운 물리 블록 주소로 갱신 된다. 두 번째는 교체 병합이다. 데이터 블록의 페이지 업데이트가 순차로 발생 하여 로그 블록 상의 오프셋과 모두 일치 하는 경우 남은 페이지들만 데이터 블록에서 로그 블록으로 복사하여 오프셋이 순차 정렬된 블록을 만들 수 있다. 매핑 값은 로그 블록의 물리 주소로 교체 된다. 비교적 다수의 페이지들을 복사하는 전체 병합은 임의 쓰기 대해서는 자주 발생 할 수 밖에 없다.

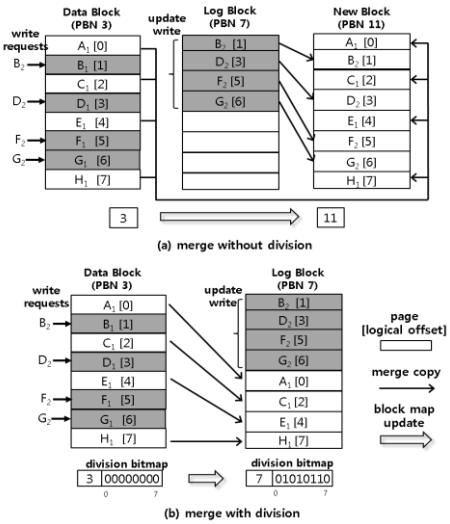

그림 3 제안 기법을 적용하지 않은 경우와 적용한 경우의 블록 병합 과정 비교

그림 3은 순서 증가 임의 쓰기에 대해 제안 기법을 사용한 경우와 사용하지 않은 경우의 블록 병합 과정을 비교한 것이다. 먼저 그림 3-a에서 논리 오프셋 1, 3, 5, 6에 페이지 쓰기가 발생 하였을 때 해당 쓰기는 로그 블록에 차례로 업데이트 된다. 제안 기법이 적용되지 않았기 때문에 논리 오프셋과 물리 오프셋이 일치 하지 않는 페이지에 대해 대응할 수 없다. 따라서 병합 시 새로운 블록을 할당하고 오프셋 재 정렬을 위해 전체 병합을 수행한다. 그림 3-b에서 제안 기법이 사용된 경우 데이터 블록의 남은 페이지들을 오프셋 순서에 따라 로그 블록에 복사하면 증가 오프셋 분할이 유효하여 division bitmap을 통해 매핑이 가능하다. 따라서 오프셋을 재 정렬할 필요 없이 교체 병합이 수행된 다. 전자의 경우 블록의 전체 페이지 수인 8 페이지가 복사 되는 것에 반해 후자의 경우 임의 쓰기가 발생 한 블록임에도 불구하고 4 페이지만이 복사되어 성능이 증가한다.

3. 성능 평가

3.1 평가 환경

평가를 위해 제안 기법을 Jasmine[5] 보드에 구현 하였다. Jasmine 보드는 open SSD 플랫폼으로 NAND 플래시 모듈을 장착하고 FTL 펌웨어를 구동할 수 있는 환경을 제공한다. 실험은 4GB DRAM을 장착한 머신에서 구동되는 리눅스 운영체제 상에서 진행하였으며 사용된 커널 버전은 4.2.30 이다. workload 생성 도구는 fio[6] 이다.

3.3 성능 측정

그림 4는 EXT4 파일 시스템 상에서 256MB 파일에 대한 임의 쓰기 성능을 각 매핑 방식 별로 비교한 것이다.

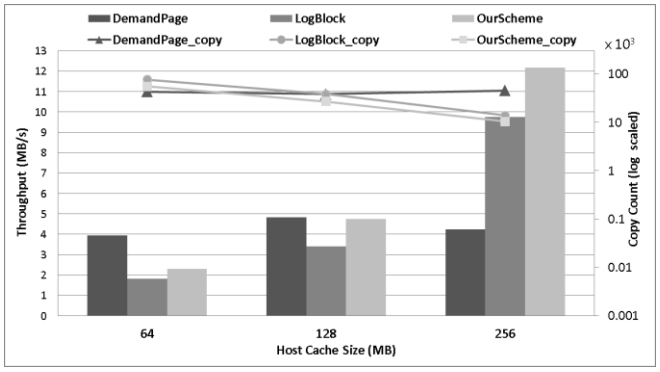

그림 4 각 매핑 방식 별 256MB 파일에 대한 임의 쓰기 성능 비교
(X축: 호스트 버퍼 크기, Y축-좌: throughput, Y축-우: 병합 복사 횟수)

실험에서 호스트 캐시는 실험을 위해 실제 DRAM 보다 더 작은 크기로 제한 하였으며 일반적으로는 제한한 크기 보다 더 큰 공간을 캐싱 할 수 있다. 먼저 제안 방식은 기존 로그 블록 방식에 비해 30% 정도 향상된 성능을 보인다. 이것은 쓰기 패턴 중 순서 증가를 인식한 결과 병합 비용이 향상 되었기 때문이다. 요구 기반 페이지 매핑 (DemandPage)은 호스트 캐시 64MB에서 제안 방식 (OurScheme)에 비해 40% 더 나은 성능을 보인다. 제안 방식의 경우 임의 영역 대비 캐시의 크기가 작기 때문에 비교적 입출력 정렬이 되지 않아 요구 기반에 비해 병합 비용이 크다. 그러나 호스트 캐시의 크기가 128MB 수준 일 때 제안 방식 및 로그 블록(LogBlock)은 병합 비용이 낮아 지는 반면 요구 기반은 같은 수준의 GC 비용을 유지한다. 특히 제안 방식이 적용된 경우 병합 비용이 많이 낮아져 throughput은 요구 기반과 동일한 수준을 유지한다. 호스트 캐시의 크기가 임의 영역의 크기와 같아지는 256MB에서는 거의 모든 쓰기 요청이 순차 정렬되어 요청되므로 병합 비용이 더 작아지며 이에 따라 throughput이 더 증가한다. 반면 요구 기반 페이지 매핑은 지속적으로 dirty page 회수를 위한 GC를 수행해야 하기 때문에 입출력 패턴이 좋아지더라도 성능이 좋아지지 않는다. 실험 동안 요구 기반은 demand loading에 의한 플래시 접근 보다 GC에 의한 블록 병합 비용이 성능 저하에 더 큰 영향을 미치는 것으로 나타났다. 실험 동안의 입출력 단위 크기는 플래시 메모리의 페이지 크기와 동일하다.

항목	*전체 병합*	*교체 병합*	*블록 삭제*
요구 기반	1725	0	1737
로그 블록	995	430	2420
제안 방식	24	1442	1490

표 1 전체 성능 측정 동안의 병합 및 블록 삭제 횟수 비교

표 1은 전체 성능 측정 동안 각 방식 별 병합 횟수를 비교한 것이다. 요구 기반의 경우 교체 병합이 없이 모두 전체 병합이 수행 된다. 제안 방식은 순서 증가를 인식하기 때문에 거의 대부분 교체 병합을 수행 하였다. 따라서 블록 삭제는 로그 블록 대비 38% 더 적은 블록 삭제 횟수를 보이며 요구 기반에 비해서도 14% 더 적다. 세 가지 방식 중 제안 방식과 요구 기반은 동일한 양의 메모리 공간을 사용하였으며 로그 블록 방식은 제안 방식에서 division bitmap 공간을 제외한 만큼의 메모리 공간을 사용하였다. 로그 블록 방식과 제안 방식의 로그 공간 크기는 48MB이다. 본 실험은 파일 시스템 상에서 수행 한 것으로 파일 시스템의 저널과 메타 쓰기는 고려하지 않은 것이다. EXT4는 저널과 메타 영역이 고정되어 있으므로 해당 영역을 고려하여 블록을 구분 한다면 훨씬 더 나은 성능을 기대 할 수 있다.

3.4 메모리 요구량

SSD 용량	*페이지 매핑*	*제안 방식*	*로그 블록*
4GB (실험)	1068K(46K)	46K	12.4K
64GB	64MB	9MB	7MB
16TB	16GB	740MB	228MB

표 2 매핑 방식 별 매핑 테이블 공간 요구량 비교. (페이지 크기 4KB, 블록당 페이지 수 256 기준, 로그 블록 방식과 제안 기법의 페이지 매핑 공간은 전체 스토리지 용량의 1%로 산정)

표 2는 매핑 방식 별 SSD 용량에 따른 매핑 테이블 공간 요구량을 비교한 것이다. 먼저 페이지 매핑과 로그 블록 방식을 비교해 볼 때 64GB 용량의 일반적인 SSD에서는 64MB로 7MB인 로그 블록에 비해 대략 9배 정도의 차이를 보이고 있으나 8TB와 16TB 크기의 대용량 SSD에서는 73배와 72배로 그 차이가 상당히 커진다. 제안 기법의 경우 기존 로그 블록 방식에서 페이지 당 1 비트의 추가 공간을 사용한다. 로그 블록 방식에 대해 1.3배에서 최대 3.2배 정도의 작은 차이를 보이며 페이지 매핑에 비해서는 7배 에서 최대 22배 더 작은 공간 요구한다. 로그 블록 방식 수준의 저렴한 비용으로 페이지 단위 매핑을 수행 할 수 있는 것은 제안 기법의 큰 장점으로 볼 수 있다.

4. 결 론

로그 블록 방식은 적은 양의 매핑 테이블 공간을 사용하면서 부분적인 페이지 매핑을 지원하기 때문에 대용량 SSD의 매핑 테이블 공간 요구량 문제에 대한 대안이 될 수 있다. 그러나 전통적인 블록 매핑 알고리즘에 기인하는 한계가 있기 때문에 임의 쓰기 workload에 대해서 지나친 병합 비용에 의해 쓰기 성능이 저하되는 단점이 있다. 본 논문에서는 순서 증가를 인식하는 블록 단위 매핑을 사용하여 로그 블록 방식의 임의 쓰기 성능을 향상 시켰다. 실험 결과 비교적 넓은 범위의 임의 쓰기에 대해서 30% 가량 성능이 향상 되었으며 블록 삭제 횟수는 38% 줄었다. 또한 요구 기반 페이지 매핑에 비해서도 호스트 캐시의 지원에 따라 더 나은 성능을 보여 준다.

참고 자료

[1] 김혁중, 신동군. "SHRD: SSD의 임의쓰기 성능향상을 위한 로깅 기법", 한국정보과학회

[2] A. Gupta, Y. Kim, and B. Urgaonkar, “DFTL: a Flash Translation Layer Employing Demand-based Selective Caching of Page-Level Address Mappings,” in Proceedings of 2009 ASPLOS

[3] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, “A Space-Efficient Flash Translation Layer for CompactFlash Systems,” IEEE Transactions on Consumer Electronics

[4] S. Lee, D. Park, T. Chung, D. Lee, S. Park, and H. Song. "A Log Buffer based Flash Translation Layer Using Fully Associative Sector Translation," In ACM TECS

[5] http://www.openssd-project.org/wiki/Jasmine_OpenSSD_Platform

[6] http://freecode.com/projects/fio