페이지 단위 매핑 기반 대용량 NAND플래시를 위한 주소변환기법

(An Address Translation Technique for Large NAND Flash Memory using Page Level Mapping)

서 현 민 † 권 오 훈 ††
(Hyunmin Seo) (Ohhoon Kwon)
박 준 석 † 고 건 †††
(Junseok Park) (Kern Koh)

요 약 SSD는 NAND 플래시 메모리 기반의 저장장치로 속도가 빠르고, 전력 소모량이 작으며, 충격과 진동에 강하다는 좋은 특성 때문에 PC뿐 아니라 스토리지 서버 등에서도 사용되는 경우가 늘고 있다. NAND 플래시 메모리는 덮어쓰기가 불가능하다는 제약이 있으므로 SSD에서는 일반적으로 FTL이라고 불리는 소프트웨어 계층을 사용한다. 다양한 형태의 FTL 중 페이지 단위 변환에 기반한 FTL은 유연성이 높고 효율적인 쓰레기 수집 작업이 가능하다는 점에서 가장 성능이 좋다고 알려져 있다. 한편 이 방법은 64GB MLC SSD의 경우 64MB 크기의 변환 테이블이 메모리에 올라와 있을 것을 요구하므로 현실적인 사용이 제한되어 있다. 본 논문에서는 효율적인 캐시 구조를 통해 SSD에서도 순수한 페이지 단위 변환을 사용하는 방법을 제안한다. 제안된 방법에서는 매핑 테이블 메타 데이터를 사용해 완전 연관 캐시를 구성하고 캐시크기에 무관하게 O(1)시간에 주소를 변환한다. 다양한 환경에서 수집한 트레이스를 이용한 시뮬레이션 결과 32KB의 캐시 공간의 경우 80% 이상, 512KB의 경우 90% 이상의 적중률을 보였다. 이 경우 메모리 사용량은 64MB의 1.9%에 불과하며 캐시 미스로 인한 오버헤드는 실행시간 기준으로 2% 미만으로 측정되었다.

키워드 : FTL, SSD, 낸드 플래시, 주소변환

Abstract SSD is a storage medium based on NAND Flash memory. Because of its short latency, low power consumption, and resistance to shock, it's not only used in PC but also in server computers. Most SSDs use FTL to overcome the erase-before-overwrite characteristic of NAND flash. There are several types of FTL, but page mapped FTL shows better performance than others. But its usefulness is limited because of its large memory footprint for the mapping table. For example, 64MB memory space is required only for the mapping table for a 64GB MLC SSD. In this paper, we propose a novel caching scheme for the mapping table. By using the mapping-table-meta-data we construct a fully associative cache, and translate the address within O(1) time. The simulation results show more than 80 hit ratio with 32KB cache and 90% with 512KB cache. The overall memory footprint was only 1.9% of 64MB. The time overhead of cache miss was measured lower than 2% for most workload.

Key words : FTL, SSD, NAND Flash, Address Translation

1. 서 론

NAND 플래시 메모리는 HDD와 달리 기계적 장치를 사용하지 않으므로 접근 속도가 빠르고 전력 소모가 작으며 충격과 진동에 강하다. NAND 플래시의 집적도는 매년 크게 증가하고 있으며 MP3, 핸드폰 등 많은 휴대용 기기에서 NAND 플래시를 데이터 저장 매체로 사용하고 있다. 최근에는 NAND 플래시를 병렬적으로 연결하여 읽기, 쓰기 성능을 크게 향상시킨 SSD(Solid State Disk)가 활발히 개발되고 있다.

NAND 플래시를 저장장치로 사용할 때에는 FTL (Flash Translation Layer)의 사용이 일반적인데 FTL이 전체 저장 장치의 성능을 결정할 정도로 중요하게 인식되고 있다. 페이지 단위 변환 방식에 기반한 FTL은 쓰기 요청이 임의의 물리 페이지에 수행될 수 있으므로 유연성이 크고, 자주 업데이트되는 페이지를 같은 블록에 모아 쓰레기 수집 작업(Garbage Collection)의 효율성을 높일 수 있다는 점에서 성능이 뛰어나다. 반면 64GB MLC SSD의 경우 64MB 크기의 변환 테이블이

· 본 연구는 2009년 정부(교육과학기술부)의 재원으로 한국학술진흥재단의 지원을 받아 수행되었으며(KRF-2009-0076335), 또한 서울대학교 컴퓨터연구소로부터 연구장비와 공간을 지원받았음
· 이 논문은 2009 한국컴퓨터종합학술대회에서 '페이지 단위 변환 방식에 기반한 대용량 NAND 플래시에서의 FTL'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학부
yzzmin@oslab.snu.ac.kr
redo@oslab.snu.ac.kr
†† 비 회 원 : 서울대학교 컴퓨터공학부
ohkwon@oslab.snu.ac.kr
††† 정 회 원 : 서울대학교 컴퓨터공학부 교수
kernkoh@oslab.snu.ac.kr
논문접수 : 2009년 8월 13일
심사완료 : 2010년 1월 7일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제3호(2010.3)

메모리에 올라와 있어야 하므로 현실적으로 사용이 어려운 한계가 있다.

본 논문에서는 변환 테이블 일부만 캐싱하는 방식을 사용해 대용량 NAND 플래시에서도 페이지 단위의 변환 방식을 사용할 수 있는 방법을 제시한다. 제안된 기법은 주소 변환이 캐시 크기에 무관하게 O(1) 시간 내에 이루어지며 알고리즘과 자료구조가 단순해 내장형 기기에서도 큰 오버헤드 없이 구현이 가능하고 메모리 사용량이 적다는 장점이 있다. 실험결과 32KB의 캐시만으로 대부분의 워크로드에서 80% 이상, 512KB의 경우 90% 이상의 적중률을 보였다. 캐시로 인한 오버헤드는 모든 변환테이블이 메모리에 올라와 있는 이상적인 경우와 비교했을 때 추가적으로 발생하는 읽기, 쓰기 연산으로 인한 실행 시간의 증가로 측정하였다. 512KB 캐시를 기준으로 오버헤드는 2% 미만인 것으로 측정되었다.

본 논문은 아래와 같이 구성되어 있다. 2장에서는 FTL과 관련연구에 대해 소개한다. 3장에서는 본 논문에서 제안하는 주소변환 기법에 대해 알아본다. 4장에서는 다양한 환경에서 수집한 트레이스를 통해 제안한 캐시 관리 방법의 성능을 캐시 교환 정책과 함께 분석하고 5장에서 결론을 맺는다.

2. 배경지식

2.1 FTL

본 논문에서는 4KB 크기의 페이지에 블록당 128개의 페이지를 가지는 MLC NAND 플래시를 대상으로 삼는다. Read, Write, Erase latency는 각각 50μs, 900μs, 3500μs 이다[1].

NAND 플래시 메모리에서의 읽기, 쓰기는 페이지(Page) 단위로 이루어지고 한번 쓰기 작업이 이루어진 페이지에 새 데이터를 덮어 쓰려면 그 페이지가 포함되어 있는 블록(Block) 전체에 대해 지우기(Erase) 작업이 수행되어야 한다. 이러한 제약을 호스트 시스템으로부터 숨기고 NAND 플래시를 기존의 일반적인 블록 인터페이스 장치로 추상화시키기 위해 FTL이라고 불리는 변환계층을 사용하는 것이 일반적이다. FTL은 호스트 시스템에서 인식하는 논리적 주소와 실제 NAND 플래시에서의 물리적 주소 사이의 변환정보를 관리하고 쓰레기 수집을 수행하게 된다.

주소 변환정보는 블록 단위로 관리되는 방식과 페이지 단위로 관리되는 방식이 있다. 페이지 단위 방식은 유연성이 크고 효율적인 쓰레기 수집 작업이 가능해 성능이 좋은 반면 변환 테이블의 크기가 커지게 된다. 블록 수준의 변환 방식은 변환 테이블의 크기는 작지만 데이터를 저장하는데 있어서 위치적 제약이 있으며 이로 인해 쓰기 작업과 쓰레기 수집 작업의 효율성이 낮아진다.

64GB의 SSD의 경우, 4Byte 주소를 사용하게 되면 블록 단위 변환 방식의 경우 512KB, 페이지 단위 변환 방식의 경우 64MB의 변환 테이블이 필요하게 된다.

2.2 관련연구

따라서 변환 테이블의 크기는 줄이면서 성능은 높이기 위해 두 방식을 결합한 하이브리드 형태의 변환 방식이 제안되어 왔다[2-6]. 하이브리드 방식은 전체 블록을 데이터 블록과 로그 블록으로 구분하여 데이트 블록에는 블록 변환 방식을, 로그 블록에는 페이지 변환 방식을 사용하는 방법이다. 기존 연구에서는 대략 전체 크기의 3%[6] 정도를 로그 블록으로 할당하였다. 로그 블록이 가득 차게 되면 머지 연산이 수행되는데 그 형태에 따라 스위치 머지, 파셜 머지, 풀 머지로 구분되며 풀 머지 연산의 오버헤드가 가장 크다.

BAST(Block Associative Sector Translation)[3] 방식은 임의의 데이터 블록에 전용의 로그 블록을 할당하는 것으로 소량 임의 쓰기가 많을 경우 로그 블록이 충분히 채워지지 않은 상태에서 머지 연산이 일어날 수 있고 로그 블록 thrashing[4] 문제가 나타난다. FAST[4] (Fully Associative Sector Translation) 방식은 하나의 로그 블록을 전체 데이터 블록이 공유하도록 하여 로그 블록 사용률을 높인 반면 소량 랜덤 쓰기가 많을 경우 로그 블록 하나를 머지하기 위해 여러 번의 풀 머지 연산이 필요하게 되는 단점이 있다. 수퍼블록 FTL [5]의 경우 주소 변환 시 여러 곳의 OOB 영역에 읽기, 쓰기 연산이 일어날 수 있고 워크로드의 특성에 따라 미리 수퍼블록의 크기가 튜닝되어야 한다는 문제점이 있다. LAST (Locality-Aware Sector Translation)[6] 방식은 해당 논문에서 지적된 바와 같이 작은 크기의 순차 특성을 가진 워크로드는 정확히 구분해 내지 못하는 단점이 있다.

순수한 페이지 단위 변환 방식을 사용하게 되면 위 방법들과는 달리 전체 NAND 블록을 하나의 로그로 사용하게 되므로 로그블록의 크기에 제약을 받지 않고 사용률을 크게 높일 수 있다. 또한 페이지가 임의의 위치에 저장될 수 있으므로 풀 머지 연산이 체인 형태로 일어나는 것을 방지할 수 있다. 뿐만 아니라 Hot 데이터와 Cold 데이터의 구분을 통해 쓰레기 수집의 오버헤드를 크게 줄일 수 있다[7].

3. 제안하는 주소변환기법

본 논문에서는 참조 패턴의 지역성을 활용하여 전체 변환 테이블 중 자주 참조 되는 부분만 캐시에 올려놓아 메모리 사용량은 줄이되 페이지 수준 변환 방식의 장점은 그대로 갖는 주소변환 기법을 제안한다.

3.1 캐시 구성 및 매핑 테이블 메타 데이터

제안하는 방법에서는 캐시에 들어오고 나가는 데이터

의 단위를 섹션이라고 부르기로 하고 하나의 섹션에는 블록 하나에 해당하는 변환 정보가 들어가도록 한다. 이 경우 섹션의 크기는 512B, 1MB의 캐시 공간에는 총 2048개의 블록에 대한 변환 정보가 캐시될 수 있다.

임의의 페이지에 대한 읽기 또는 쓰기 요청이 들어올 경우 그 페이지에 대한 변환 정보가 캐시되어 있는지 여부를 알아야 한다. 논문에서는 O(1)시간 내에 변환 정보를 얻어내고 캐시 관리에 대한 오버헤드를 줄이기 위해 추가적인 자료구조, 즉 매핑 테이블 메타 데이터를 활용한다.

매핑 테이블 메타 데이터는 일련의 배열으로 전체 블록 수만큼 엔트리가 존재하고 각각의 엔트리는 48비트로 구성된다. 첫 번째 비트는 그 블록에 대한 변환 정보가 캐시되어 있는지 여부, 두 번째 비트는 캐시된 변환 테이블이 수정되었는지 여부를 나타낸다. 내용이 수정된 변환 테이블은 캐시에서 쫓겨날 때 다시 플래시 메모리에 써져야 한다. 다음 14비트는 매핑 정보가 캐시되어 있을 경우 메모리의 어느 위치에 있는지 알려주는 것으로 전체 캐시 공간을 섹션의 배열로 보고 배열의 인덱스를 나타내게 된다. 마지막 32비트는 그 블록에 대한 매핑 정보가 저장되어 있는 NAND 페이지의 주소를 나타낸다. 64GB SSD의 경우 매핑 테이블 메타 데이터의 크기는 768KB이다. 그림 1은 매핑 테이블 메타 데이터에서 하나의 엔트리에 대한 구조를 나타낸다.

그림 2는 제안된 방법에서의 주소 변환 구조를 보여주는데 임의의 페이지에 대한 요청이 들어올 경우 LBN

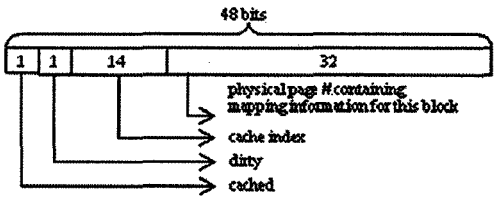


그림 1 매핑 테이블 메타 데이터 엔트리 구조

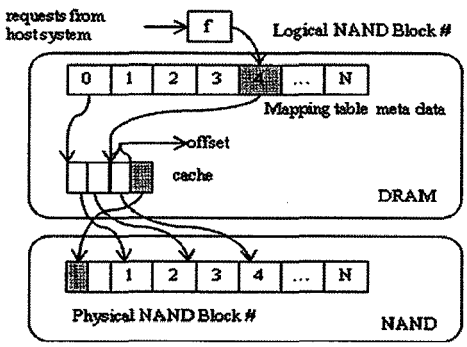


그림 2 매핑 테이블 메타 데이터와 캐시를 활용한 주소 변환

과 매핑 테이블 메타 데이터를 참조해 그 페이지의 변환 정보가 캐시되어 있는지 여부를 살펴본다. 그림 2는 두 개의 블록, LBN 0과 4에 대한 변환 테이블이 캐시되어 있음을 보여주고 DRAM 상에서 캐시된 위치에 대한 포인터를 보여준다. 그 주소로부터 블록 내의 페이지 오프셋 값을 이용하면 요청된 페이지의 변환 정보에 바로 접근이 가능하다.

3.2 캐시 교체 알고리즘

요청된 페이지의 변환 정보가 캐시에 없을 경우(Cache Miss) 하나의 섹션을 캐시로부터 쫓아내고 요청된 페이지가 속한 블록의 변환 정보를 가져와야 한다. 이때 어느 섹션을 쫓아내는 것이 가장 효율적인가와 관련하여 많은 알고리즘이 제시되어 있다. 본 논문에서는 OPT, LRU, LRFU, LRU-K, LIRS, CFLRU, LRUWSR 알고리즘을 비교해 본다. LRFU[8]는 튜닝 상수 λ의 값에 따라 LRU와 LFU 사이의 중간적 성질을 가지는 것으로, 가중치 C(x) 가 모든 블록에 대해 정의되어 있고 매번 데이터가 참조될 때마다 아래 식에 따라 C(x)가 수정된다. 실험에서는 0.1의 λ값을 사용하였다.

$$C(x)=\begin{cases}1+2^{-\lambda}C(x) & x\text{가 시간 } t\text{에 참조됨}\\ 2^{-\lambda}C(x) & \text{그 외}\end{cases}$$

LRU-K[9]는 가장 최근으로부터 K번째 이전의 참조 시간을 기준으로 LRU 알고리즘을 적용하는 것으로 실험에서는 LRU-2를 사용한다. LIRS(Low Inter-reference Recency Set)는 각 블록의 최근 참조 시간과 그 직전 참조시간 사이의 거리를 측정하여 각각을 LIR과 HIR로 구분한 뒤 서로 다른 교체 정책을 적용한다. CFLRU와 LRUWSR은 NAND 플래시의 비대칭적인 읽기, 쓰기 시간을 고려한 교체정책으로 쓰기 횟수를 줄이고 읽기 횟수는 늘림으로 전체적인 성능을 향상시키려는 정책이다.

3.3 기타

한번 쓰기가 일어난 페이지에 대해 그 내용이 업데이트 되면 기존의 페이지는 invalid 페이지가 된다. 이러한 페이지들을 그냥 놔둘 경우 NAND 플래시에서 가용 공간이 줄어들게 되므로 유효한 페이지들을 한 곳에 모으고 쓸모 없게 된 블록을 지우는 쓰레기 수집 작업이 필요하다. 이와 관련하여 Chiang 등은 Hot Data와 Cold Data를 구분하여 NAND 플래시 상에서 서로 다른 지역에 저장되도록 하는 DAC(Dynamic dAta Clustering)[7] 방법을 제안하였고 상당히 우수한 쓰레기 수집 작업 효율성을 보였다. 본 논문에서는 쓰레기 수집 작업은 별도로 고려하지 않도록 한다.

캐시에 올라와 있는 섹션은 캐시에서 쫓겨나게 될 경우 그 내용에 수정이 있었다면 다시 빈 페이지에 기록이 되어야 한다. 이 정보는 다시 매핑 테이블 메타 데이터의

마지막 32 비트에 저장되어야 하고 매핑 테이블 메타 데이터 역시 다시 NAND 플래시에 저장되어야 한다. 본 논문에서는 정기적으로 매핑 테이블 메타 데이터가 백업된다고 가정하고 캐시의 효율성 부분에 초점을 맞춘다.

4. 실험결과

4.1 워크로드

표 1은 실험을 위해 64GB의 하드디스크가 설치된 시스템으로부터 수집한 트레이스에 대한 정보이다. (a)와 (b)는 일반적인 컴퓨터 사용 환경의 워크로드로 (b)의 경우 더 많은 프로그램이 동시에 수행되고 작업의 전환이 자주 일어나는 워크로드이다. (c)에서는 윈도우 운영체제 서비스팩 설치와 보안 업데이트 설치, 기타 다양한 프로그램의 설치와 업데이트를 수행하였다. (d)는 대용량 순차 읽기와 쓰기에 대한 트레이스로 HDD에 존재하는 1.4GB의 파일을 읽어서 다른 폴더에 복사하는 작업의 트레이스이다.

표 1 트레이스별 요청의 총수와 크기(MB)

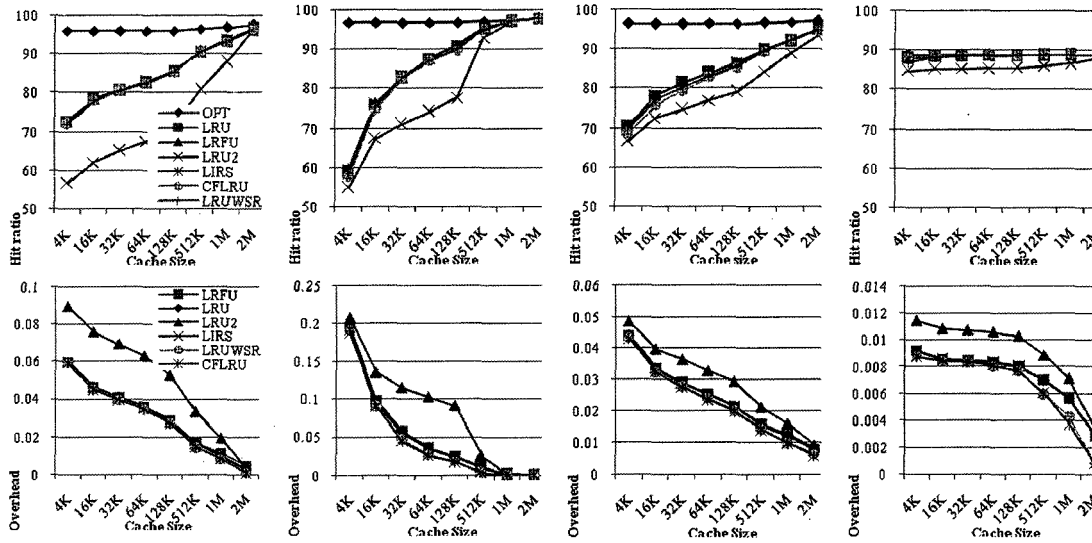
trace	requests	data size [MB]
(a) daily usage	545031	10270.78
(b) multi program	309262	3070.669
(c) install update	1022856	14072.22
(d) large file	45593	2810.333

4.2 캐시 적중률 및 캐시 미스 오버헤드

그림 3은 캐시 크기와 캐시 교체 알고리즘을 변경해 가며 위 네 가지 트레이스에 대해 실험한 결과를 보여준다. 각 트레이스 별로 첫 번째 그림은 캐시 적중률, 두 번째 그림은 캐시 미스로 인한 시간 오버헤드를 나타낸다. 전반적으로 LRU-2의 성능이 가장 좋지 않고 OPT를 제외한 나머지 알고리즘의 성능은 비슷함을 볼 수 있다.

캐시 적중률의 경우 모든 워크로드에서 캐시 사이즈가 32KB만 되어도 LRU 알고리즘을 이용하면 80%이상, 1MB일 경우 *large_file* 트레이스를 제외하면 90% 이상의 적중률을 얻을 수 있었다.

그림 3(d)에서 large file 트레이스는 대용량 순차 읽기와 쓰기를 나타내는데 다른 트레이스와는 달리 캐시 교체 정책 간에 성능 차이가 거의 없으며 캐시 크기가 증가해도 적중률이 거의 올라가지 않음을 볼 수 있다. 대용량 순차 작업의 경우 한번 참조가 이루어진 페이지에 대해서 대부분의 경우 재참조가 일어나지 않으므로 캐시 교체 정책을 통해 이득을 볼 수가 없다. 캐시 크기가 작을 때에도 대략 85%의 적중률을 보이는 이유는 공격적인 선반입(Prefetching) 때문이라고 해석할 수 있다. 대량 순차 읽기, 쓰기의 경우 대부분의 요청이 128 섹터 크기로 들어오게 되는데 이는 64KB에 해당한다. 한편 캐시 미스가 나서 변환 정보를 새로 캐시로 읽어올 때는 하나의 블록 전체에 대한 변환 정보를 가져오게 되는데 실험에서 사용한 MLC 플래시 메모리의 경우 블록 사이즈는 512KB이다. 이는 64KB의 8배에 해당하므로 한번 캐시 미스가 난 이후 7번의 요청은 캐시에서 적중하게 된다. 이것이 캐시 크기에 관계없이 대략



(a) daily usage (b) multi program (c) install update (d) large file

그림 3 캐시 사이즈와 트레이스 종류에 따른 Hit Ratio 및 캐시 미스로 인한 시간 오버헤드

85% 정도의 적중률을 보이는 이유이다. 이 경우 캐시 크기를 2MB로 늘려도 다른 트레이스와 달리 90% 이하의 적중률을 보인다.

4.3 캐시 오버헤드

캐시에서 미스가 나게 되면 해당 매핑 정보를 읽어와야 하고 쫓겨난 섹션이 더티한 섹션일 경우 이를 NAND에 써야한다. 이는 호스트 시스템에서 요청하지 않은 추가적인 읽기, 쓰기 요청으로 캐시로 인한 오버헤드가 된다. 매핑 테이블이 모두 메모리에 올라와 있어 캐시를 사용하지 않는 경우에는 이러한 요청은 발생하지 않는다. NAND는 비대칭적인 읽기, 쓰기 비용을 가지고 있으므로 각 요청을 처리하는데 걸리는 시간을 고려하여 다음과 같이 오버헤드를 측정하였다. Ttrace, Tcache_miss는 해당 트레이스에 있는 요청을 모두 처리하는데 걸린 시간, 캐시 미스로 인해 추가적으로 발생한 읽기, 쓰기 요청을 처리하는데 걸린 시간이다.

$$Overhead = \frac{T_{cache\ miss}}{T_{trace}}$$

한편 캐시 미스로 인해 쓰기가 발생할 경우에는 쓰레기 수집 작업의 비용도 올라가게 된다. 이는 쓰레기 수집 작업에 사용되는 알고리즘과 데이터 재배치 사용 여부에 따라 크게 달라진다. 본 논문에서는 쓰레기 수집 작업이 일어나기 이전의 주소변환만을 고려하고 있으므로 이를 모두 측정할 수는 없었다. 따라서 캐시 미스로 인해 발생하는 추가적 읽기, 쓰기 요청의 비용만을 제안한 기법의 오버헤드로 파악하였다.

그림 3에서 각 트레이스의 두 번째 그림은 위 식을 이용해 오버헤드를 계산한 것으로 대부분의 워크로드에서 512KB의 캐시만 있으면 캐시 미스로 인한 시간 오버헤드는 2%미만인 것으로 측정되었다. NAND 플래시의 경우 쓰기 속도가 읽기 속도보다 10배 이상 느리므로 쓰기 횟수를 줄이는 LRUWSR, CFLRU 등의 알고리즘을 사용하면 시간 오버헤드가 더 줄어드는 것을 볼 수 있다.

4.4 캐시의 메모리 사용량

512KB의 메모리가 캐시로 사용될 경우 매핑 테이블 메타 데이터 768KB를 포함하면 총 1280KB의 메모리가 필요하며 이는 64MB의 페이지 변환 테이블 전체가 메모리에 올라와 있는 경우의 1.9%에 불과하다.

5. 결 론

본 논문에서는 자주 참조되는 변환 테이블만을 메모리에 캐시하는 방법을 통해 대용량 SSD 등에서 사용될 수 있는 페이지 수준 변환 방식을 위한 주소변환기법을 제안했다. 제안된 방법을 사용하면 64MB의 페이지 변환 테이블 전부가 메모리에 올라와 있는 경우에 비해 1.9%의 메모리 사용만으로 90% 이상의 캐시 적중률을 보였고 성능 저하는 시간 기준으로 2% 미만으로 측정되었다.

본 연구는 FTL 여러 성능 지표 중 캐시의 적중률과 캐시로 인한 메모리, 시간 오버헤드만 측정하였는데 추후 연구에서는 타 FTL과 비교한 읽기, 쓰기 성능 및 쓰레기 수집 성능 등 전체적인 FTL의 성능을 비교해 볼 필요가 있다.

참 고 문 헌

[1] Micron MLC NAND (MT29F64G08UAAC4) datasheet http://www.micron.com.

[2] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems," *Consumer Electronics, IEEE Transactions on*, vol.48, no.2, pp.366-375, May 2002.

[3] T. Chung, D. Park, S. Park, D. Lee, S. Lee, and H. Song, "System Software for Flash Memory: A Survey," In *Proceedings of the International Conference on Embedded and Ubiquitous Computing*, pp.394-404, August 2006.

[4] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *Trans. on Embedded Computing Sys.*, vol.6, no.3, pp.18, 2007.

[5] J. Kang, H. Jo, J. Kim, and J. Lee, "A Superblock-based Flash Translation Layer for NAND Flash Memory," In *Proceedings of the International Conference on Embedded Software (EMSOFT)*, pp.161-170, October 2006. ISBN 1-59593-542-8.

[6] S. Lee, D. Shin, Y. Kim, and J. Kim, "LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems," In *Proceedings of the International Workshop on Storage and I/O Virtualization, Performance, Energy*, Evaluation and Dependability (SPEED2008), Feburary 2008.

[7] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang, "Using data clustering to improve cleaning performance for flash memory," *Software: Practice and Experience*, vol.29, no.3, pp.267-290, 1999.

[8] D. Lee, J. Choi, J.-H. Kim, S.H. Noh, S.L. Min, Y. Cho, and C.S. Kim, "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies," *IEEE Trans. Computers*, vol.50, no.12, pp.1352-1360, Dec. 2001.

[9] E.J. O'Neil, P.E. O'Neil, and G. Weikum, "The LRU-K Page Replacement Algorithm for Database Disk Buffering," *Proc. ACM Int'l Conf. Management of Data (Sigmod '93)*, May 1993.