Journal of The Korea Society of Computer and Information
Vol. 20 No. 11, pp. 31-38, November 2015

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.11.031

An Efficient Cache Management Scheme of Flash Translation Layer for Large Size Flash Memory Drives

Hwan-Pil Choi*, Yong-Seok Kim**

Abstract

Nowadays, large size flash memory drives with more than a couple of hundreds of gigabytes are common. This paper presents an efficient cache management scheme of flash translation layer, called TPC-FTL, for large size flash memory drives. Since flash drives of large size usually contain large size RAM, we can enhance the performance of page mapping cache by using more RAM for the cache. But if the size exceeds a threshold, the existing schemes are impractical for real devices, because the time for cache manipulation becomes too long. TPC-FTL manages the cache in translation page unit, not in logical page number unit used in existing schemes. Since a translation page covers a large number of logical page numbers (for example, 512 for 2KB size page), the number of cache elements can be reduced up to a practical level. A performance evaluation shows that average response time, an important performance measure, is better than existing schemes via the effect of utilizing spacial locality in addition to temporal locality.

▸ Keyword : SSD, Flash Memory, FTL, Page Mapping, Cache Management

I. Introduction

최근에 플래시 메모리 드라이브 또는 반도체 드라이브 (SSD: Solid State Drive)의 용량이 급격히 증가하면서 SSD 내부의 처리 알고리즘도 이에 대응하여 적절하게 개선할 필요가 커지고 있다. SSD는 하드디스크 드라이브 (HDD: Hard Disk Drive)에 비해서 동작속도가 빠르면서도 작고, 가볍고, 전력소모량이 적으며, 충격에 강한점 등의 여러 가지 장점이 있어서 그 사용이 빠르게 확산되고 있다. 단점으로는 가격이 상대적으로 비싸다는 점, 제자리에 재기록을 할 수 없다는 점, 지우고 재기록 가능 회수가 HDD에 비해서 적다는 점 등의 제한이 있기는 하지만, 가격 문제와 지우기 회수 문제는 반도체 제조공정의 혁신으로 빠르게 해결되어 가고 있고 제자리 쓰기 제한은 여러 가지 알고리즘으로 해결하고 있다[1-3].

SSD용 플래시 메모리는 대부분 가격대 용량비가 우수한 NAND 플래시를 사용하는데, 읽기/쓰기의 최소 단위는 페이지이며 지우기는 일정한 개수의 페이지들의 집합인 블록이 최소 단위이다. 읽기는 페이지 단위로 이루어진다. 이전에 데이터가 기록된 페이지는 지우기를 한 후에 쓰기가 가능하며, 지우기는 한 블록 단위로만 가능하다는 제약이 있다. 이것은 NAND 플래시 메모리의 특성으로서, 쓰기 과정에서 특정 비트를 1에서 0으로의 변경은 가능하지만 0에서 1로의 변경은 불가능하다는 제약점에서 출발한다. 0에서 1로 바꾸기 위해서는 특정 블록에 포함된 모든 페이지들을 한꺼번에 지우는 작업을 통해서만 가능하다.

• First Author: Hwan-Pil Choi, Corresponding Author: Yong-Seok Kim
*Hwan-Pil Choi (trdzmoon@gmail.com), Department of Computer and Communications Engineering, Kangwon National University
**Yong-Seok Kim (yskim@kangwon.ac.kr), Department of Computer and Communications Engineering, Kangwon National University
• Received: 2015. 09. 07, Revised: 2015. 09. 29, Accepted: 2015. 10. 01.
• This study is supported by 2014 Research Grant from Kangwon National University

제자리에 재기록을 할 수 없다는 문제는 파일시스템 차원에서 처리하는 방법을 사용하기도 하고, SSD 내부에서 적절하게 처리하기도 한다. SSD의 특성을 반영하여 파일 시스템 차원에서 해결하는 것으로는 YAFFS, JFFS, F2FS 등의 파일시스템들이 있다[4-6]. SSD 내부에서 처리하는 방법으로는 일반적으로 플래시 변환 계층 (FTL: Flash Translation Layer)을 두어 해결하는데, SSD 인터페이스를 HDD 인터페이스 표준과 동일하게 제공하여 운영체제 차원에서는 아무런 변경이 없이 바로 SDD로 대체하여 사용할 수 있는 장점이 있다[1-3].

FTL을 적용한 SSD의 내부 구성은 그림 1과 같다. FTL은 SSD의 쓰기 작업에 대하여 제자리 재기록 문제를 해결하는 것이 주목적이다. 추가적인 기능으로는 전체 블록들 간에 지우기 회수를 비슷하게 유지하기 위한 기능(wear leveling)과 블록 내에서 유효한 페이지들만 다른 블록으로 수집하고 이전 블록들을 지워서 재사용하도록 하는 기능(garbage collection) 등도 포함된다.

특정 페이지에 데이터를 덮어 쓰기 위해서는 먼저 해당 페이지를 지운 다음에 쓰기를 해야 하고, 지우기는 페이지 단위가 아니라 페이지들의 집합인 블록 단위로 처리해야 한다. 이러한 제약 조건 때문에 FTL에서 쓰기 작업은 지정된 페이지에 직접 기록하는 것이 아니라 이미 지워놓은 페이지 중에 하나를 골라서 기록하고, 읽기 요청에 대해서는 이들 간의 변환 표를 활용하여 실제 기록된 페이지를 결정하여 읽기 작업을 한다. 즉, 읽기나 쓰기 요청이 온 페이지 번호를 논리적 페이지 번호 (LPN: Logical Page Number)로 사용하고 플래시에 실제로 기록된 물리적 페이지 번호 (PPN: Physical Page Number)로 적절히 매핑하는 작업을 FTL에서 처리한다.

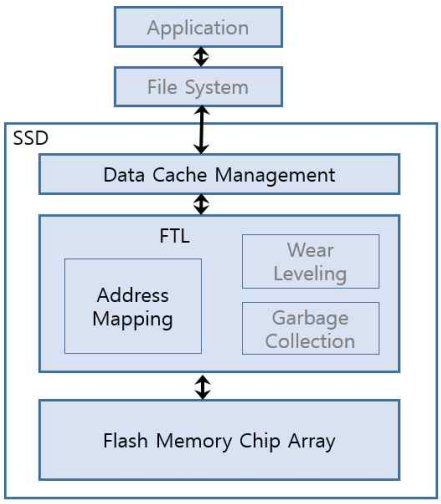


Fig. 1. Organization of SSD Using FTL

FTL을 구현하기 위한 방법으로는 크게 나누어서 블록 매핑, 페이지 매핑, 및 이들을 혼합한 하이브리드 매핑이 있다[1,2]. 블록 매핑은 매핑 정보를 블록 단위로 관리하며 논리적 블록 번호를 매핑 정보에 따라서 물리적 블록번호로 변환하고 블록내의 특정 페이지 위치인 오프셋은 그대로 사용한다. 특정 페이지에 대해 쓰기 요청이 오면, 매핑 정보로부터 이 페이지가 포함된 물리적 블록 번호로 변환하고 해당 페이지가 지워진 상태이면 그대로 쓰기를 하지만 이미 다른 데이터가 기록되어 있으면 완전히 지워진 새로운 블록을 할당하고 이전 블록의 페이지들과 지금 쓰기 요청이 온 페이지 데이터들을 기록한다. 이전 블록은 지워서 다른 쓰기 요청에 사용한다. 이러한 작업은 한 페이지를 재기록하기 위해서 전체 블록을 복사하는 부담이 커지므로 실제로는 그대로 사용하지 않고 하이브리드 매핑 방식을 사용한다.

이에 비해서 페이지 매핑은 매핑 정보를 페이지 단위로 관리하며 LPN을 매핑 정보에 따라서 PPN으로 변환한다. 특정 페이지의 쓰기 요청이 오면 지워진 임의의 페이지를 할당하고 여기에 쓰기를 한 다음 페이지별 매핑 정보에 등록한다. 페이지 매핑은 블록 매핑의 페이지 복사 부담이 없는 것이 장점이지만 매핑 테이블의 크기가 너무 커진다는 문제가 있다. 예를 들어서 100GB 용량의 SDD에 페이지 크기가 2KB라면 총 50M 개의 항목이 필요하며, 한 항목당 물리적 페이지 번호를 32비트 정수로 표현한다면 200MB (50M x 4B)라는 아주 큰 용량의 RAM이 소요된다. 페이지 매핑의 장점을 취하면서도 RAM 소요량을 줄이는 방법으로서 DFTL[2]과 CFTL[7]은 매핑 정보는 모두 플래시 메모리에 기록하되 자주 사용되는 페이지들만 매핑 정보를 RAM에 캐시 형태로 보관하는 방법을 사용한다.

하이브리드 매핑은 RAM 소요량이 작은 블록 매핑을 기본으로 하면서 페이지 단위의 쓰기작업을 로그 블록 등을 활용함으로써 오버헤드를 어느 정도 줄이는 방법을 적용한다[1]. 일정 용량의 SSD 영역을 로그 영역으로 정의하고 쓰기 작업은 이 로그 영역에 페이지 단위로 기록한다. 로그 영역의 매핑 정보는 페이지 매핑을 적용한다. 따라서 매핑 정보를 위한 RAM 용량은 블록 매핑을 위한 크기에 로그 영역의 페이지 매핑을 위한 크기만큼이 추가된다. 요청된 LPN에 대하여 먼저 로그 영역의 페이지 매핑을 검사하고 여기에 없을 경우에만 블록 매핑 정보를 활용하여 PPN을 결정한다. 로그 영역이 가득차면 일부 블록들을 선택하여 여기에 기록된 페이지들을 블록 매핑 영역으로 이동하고 이 블록들을 지워서 재사용한다. 블록 매핑 영역으로의 이동은 지워진 블록을 하나 할당하고 여기에 기존의 페이지들과 로그 영역에 갱신된 페이지들을 합병하여 기록한다. 이러한 합병 과정이 하이브리드 매핑에서는 페이지 복사를 위한 많은 부담을 유발하므로 이를 줄이기 위한 여러 가지 방안들이 제안되었다[8,9].

본 논문은 FTL을 구현하는 데 있어서 SSD 내부의 한정된 RAM 용량을 활용하여 주소 매핑을 효율적으로 처리하는 방안에 대하여 설명한다. 기본적으로 DFTL과 같은 방식을 적용하지만 캐핑 캐시 관리 방법을 달리한다. 이를 통해서 DFTL이 캐시 관리에 있어서 항목들을 검사하는 과정에 비현실적으로 많은 시간을 소모하는 문제를 해결하였으며 응답시간도 성능평가를 통하여 기존의 DFTL보다 개선됨을 확인하였다.

II. Related Works

페이지 매핑 방식을 적용한 대표적인 방식이 DFTL인데 전체적인 매핑 관리 구조는 그림 2와 같다. 기본적으로 모든 페이지 매핑 정보를 플래시의 변환 페이지(TP: Translation Page)들에 기록해 놓고 사용하며, TP는 PPN들의 배열로 구성된다. TP들이 기록된 플래시 페이지 번호(TPN)들을 상위 레벨에서 관리하는 것이 GTD (Global Translation Directory) 이다. PPN을 얻기 위해서는 TP를 읽어 와야 하는데 이를 위해 소모되는 시간을 줄이기 위해서 CMT (Cached Mapping Table)을 사용하는데 이것은 페이지들에 대한 매핑 정보를 RAM 상에 관리하는 캐시이다.

LPN으로부터 PPN으로 변환이 필요할 때마다 TP를 읽어오는 것은 상당한 오버헤드가 되므로 자주 요청되는 LPN들에 대해서는 CMT에 LPN 별로 대응 PPN 정보를 관리한다. 특정 LPN에 대한 요청이 오면, 이 LPN이 CMT에 존재하는지를 먼저 검사하고 성공하면 대응되는 PPN을 바로 사용한다. 실패하면 GTD를 통하여 관련 TP을 읽어 와서 사용하고 그 매핑 정보는 CMT에 등록하여둔다. DFTL의 성능은 CMT의 성공률에 의해 좌우되므로 이를 개선하기 위한 방안들이 필요하다. 기본적으로 CMT의 크기를 키울수록 성공률은 높아지지만 제한된 크기의 RAM 용량을 사용하면서도 성공률을 높이는 것이 관건이다.

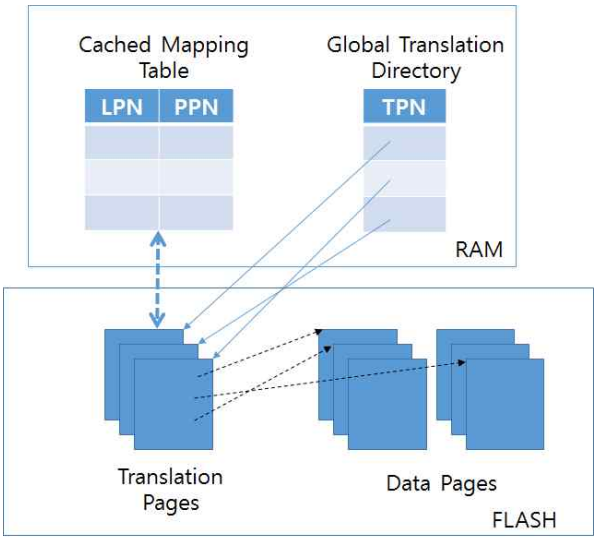


Fig. 2. Mapping Information of DFTL

CFTL은 연속된 페이지들을 하나의 CMT 항목에 표시할 수 있도록 CMT의 각 항목을 <LPN, PPN, k> 형식으로 표현하는데, LPN 부터 연속된 k 개의 페이지들이 플래시의 PPN부터 연속된 k개의 페이지들에 기록되어 있음을 표시하도록 하였다. 더 나아가서 하나의 논리적 블록의 페이지들이 동일한 물리적 블록에 연속적으로 기록되어 있으면 별도의 블록 매핑 캐시에 <LBN, PBN> 형식으로 기록한다. 자료구조가 복잡하긴 하지만 이렇게 함으로써 DFTL에 비해서 캐시에 더 많은 정보를 기록할 수 있도록 하였고 따라서 캐시의 성공률도 일정부분 향상시키는 효과가 있다.

DFTL은 FTL을 구현하는데 있어서 RAM 소요량을 제한하면서도 비교적 우수한 응답시간을 얻을 수 있지만, 실용적인 CMT를 구현하데 있어서 실행 속도에 심각한 문제가 존재한다. CFTL도 캐시에 기록하는 정보의 양을 DFTL보다 늘리기는 했지만 캐시를 관리하는 기본 과정에 근본적인 차이는 없다. DFTL의 CMT는 기본적으로 논리적인 페이지 번호 LPN과 이에 대응되는 물리적 페이지 번호 PPN의 쌍으로 이루어진 항목들의 집합이다. CMT의 검사는 다음과 같이 세 가지의 경우에 실시된다. 원하는 LPN 항목이 캐시에 있는지를 검사할 때, 캐시에서 제거할 희생 항목을 선택할 때, 그리고 TP를 플래시에 저장할 때이다.

먼저, 캐시에 원하는 항목이 있는지를 검사하는데 있어서 별도의 하드웨어 회로 지원이 없다면 단순히 순차적으로 모든 항목들을 검사하거나 해시테이블 등의 좀 더 복잡한 자료구조를 도입해야 한다. 다음으로는 캐시에서 희생 대상 항목을 선택하는데 있어서 널리 사용되는 LRU (Least Recently Used) 정책을 사용한다면 전체 항목들 중에서 사용 시간이 가장 오래된 것을 선택하기 위해 검사하는 과정이 필요하다. 이것을 위해서는 단순히 전체 항목들을 차례대로 모두 검사하거나, 시간을 단축하기 위해서 계층적 방법 등을 강구해야 한다. 세 번째로는, 선택된 희생 항목이 쓰기가 이루어진 페이지라면 변경된 PPN을 플래시의 TP에 기록해야 하는데, 이때 이 항목과 동일한 TP에 기록되어야 하는 모든 항목들을 CMT 전체에서 모두 찾아서 함께 기록해야 한다. 이 과정에서 복잡한 자료구조를 도입하지 않는 한 어쩔 수 없이 CMT의 모든 항목들을 검사해야 한다. SSD를 실물로 제작하는 데에는 가능하면 단순한 자료구조와 알고리즘을 사용하는 것이 필요할 것이다.

예를 들어서 매핑 테이블 캐시로 256KB 크기의 RAM을 사용한다면, 캐시의 항목 수는 3만 개가 넘으며 (LPN 및 PPN을 위해 각 4바이트를 사용한다면 256K/8 = 32K) 이들을 모두 검사하는 데에는 상당히 많은 시간을 소모할 수 밖에 없다. 단순 계산으로 하나 검사하는데 100ns라면 3200us라는 많은 시간을 소모하게 된다. 일반적으로 SLC 구조의 NAND 플래시를 적용한 소 용량 SSD는 한 페이지 읽기에 25us, 한 페이지 쓰기에 200us, 한 블록 지우기에 1500us 이며, TLC 구조의 대용량 SSD는 한 페이지 읽기에 100us, 한 페이지 쓰기에 1500us, 한 블록 지우기에 5000us 정도가 소요된다[3]. 따라서 DFTL에서 캐시를 검사하는데 소모하는 시간은 비현실적으로 긴 것이다. SSD의 용량이 커지면서 내부의 RAM 크기도 증가하고 있으며, 256KB 이상을 캐시로 사용하기에 충분하다. 실제로 삼성의 SSD 850PRO 시리즈 128GB 모델은 RAM이 256MB 이며, 1TB 모델의 경우에는 1GB의 RAM이 장착되어 있다[10].

실제 SSD 제품에는 여전히 하이브리드 매핑 방식을 주로 사용하고 있는 것으로 알려져 있는데 여기에는 이러한 DFTL의 한계점도 작용한 것으로 보인다. DFTL에서는 캐시의 크기가 그리 크지 않은 것을 전제로 기술하고 있으며 캐시 검사 방법에 대해서는 구체적인 언급이 없다. 본 논문에서 제시하는 TPC-FTL (Translation Page Cache FTL)은 대용량 SSD에서 캐시의 크기를 충분히 크게 확보할 수 있을 때 적용하기 위한 것이다. 매핑 정보를 저장하는 캐시를 LPN 별로 관리하지 않고 TP 단위로 관리하는 것이 핵심인데, 동일한 RAM 용량의 캐시라면 DFTL에 비해서 항목수가 대폭 줄어들고 따라서 캐시에서 항목들을 검사할 때 시간을 대폭 단축할 수가 있게 된다. 그러면서도 캐시의 성공률을 DFTL 수준 이상으로 유지할 수 있는지는 성능평가 결과에서 설명한다.

III. The Proposed Scheme

1. Structure of TPC-FTL

TPC-FTL의 전체적인 매핑정보 관리는 그림 3과 같다. 변환 페이지(TP)들은 플래시에 저장되어 있으며 변환 디렉터리(TD: Translation Directory)는 이들이 저장된 플래시 페이지 주소(TPN)들을 기록하고 있다. 즉, LPN으로부터 PPN으로 변환하는 과정은 TD 및 TP를 통하여 2단계로 이루어진다. 주소 변환 과정에서 TP를 읽는 작업은 많은 시간을 소요하므로 자주 사용되는 TP들을 변환 페이지 캐시 (TPC: Translation Page Cache)에 저장하여 사용한다. TPC에 저장된 TP들에 대해서는 TD에 TPC의 인덱스 (TCI)로 기록함으로써 TD로부터 바로 TPC의 해당 항목을 찾을 수 있도록 한다. TPC에는 저장된 TP별로 대응되는 TD의 인덱스(TDI)를 기록한다. TPC의 TPN 필드에 대해서는 추가 성능개선 방안에서 설명한다.

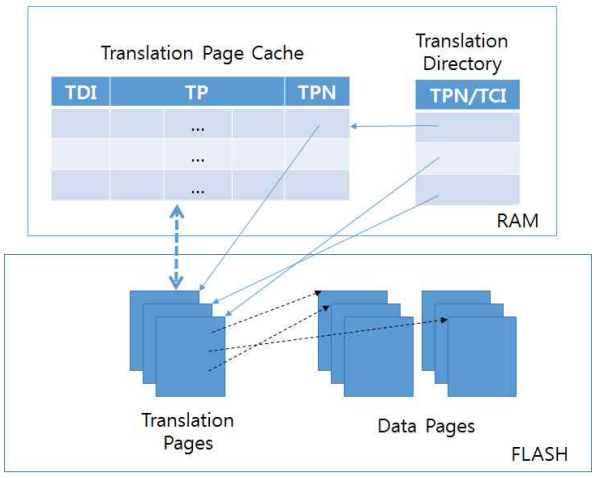


Fig. 3. Mapping Information of TPC-FTL

그림 4는 LPN으로부터 PPN으로 변환하는 과정을 세부적으로 보여준다. LPN은 TD 인덱스 부분과 TP 오프셋 부분으로 나눈다. 예를 들어서 LPN과 PPN이 32비트 (4바이트) 정수이고 페이지 당 크기가 2KB라고 가정하자. 한 페이지 크기의 TP에는 PPN 512개 (2KB/4B = 512)를 저장할 수 있으므로 TD 항목 하나당 512개의 LPN이 대응된다. 따라서 LPN의 하위 9비트 (512개 중 하나를 구별할 수 있는)가 TP 내부의 오프셋이 부분이 되고, 상위 23비트가 TD의 인덱스 부분이 된다. 페이지 번호 매핑 과정은 LPN의 TD 인덱스 부분을 이용하여 TD로부터 TP가 저장된 플래시 페이지 번호 (TPN)를 확인하고, 이 TP를 읽어서 TP 오프셋 위치로부터 PPN을 얻게 된다.

TPC의 각 항목을 위한 메모리 크기는 TP를 위한 한 페이지 크기의 메모리, TDI 및 TPN을 위한 약간의 메모리, 여기에 최근에 사용한 시간과 수정 여부를 표시하는 약간의 메모리만 추가하면 된다. 한 페이지의 크기를 2KB로 가정하고, TDI 3바이트, TPN 4바이트, 최근 사용 시간을 위해 4바이트, 수정 여부를 위해 1비트를 가정하면, TPC 한 항목당 대략 2K+12 바이트의 메모리가 필요하다. 따라서 256KB의 캐시 메모리에는 대략 128개의 캐시 항목들이 기록되며 캐시 검사는 이들 128개만 검사하면 된다.

페이지 매핑을 처리하는 전체적인 과정은 알고리즘 1과 같다. 특정 페이지의 읽기 요청에는 이 LPN에 대응되는 TP가 캐시에 있는지를 검사하고 실패했을 경우에는 해당 TP를 플래시 메모리에서 읽어 캐시에 저장한다. 캐시에 해당 TP가 확보되면 LPN의 TP 내 오프셋 위치로부터 해당 PPN을 결정한다. 특정 페이지의 쓰기 요청에 대해서도 읽기 요청과 마찬가지로 캐시 관리를 하고, 실제로 데이터를 기록하기 위해 할당한 플래시의 PPN을 TPC의 해당 항목에 기록한다. 읽기나 쓰기 요청을 처리하는 시점에 현재의 시간을 해당 캐시 항목에 기록하고, 쓰기일 경우에는 수정여부도 수정(Dirty)으로 표시한다.

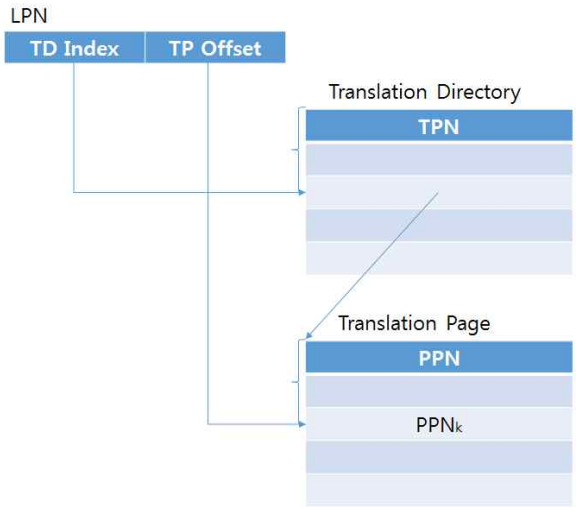


Fig. 4. Page Mapping of TPC-FTL

Algorithm 1. Page Mapping Algorithm of TPC-FTL

```
on Read Request of LPNk:
  if the requested LPNk is found on TPC
      select the TPC slot
  else
      load the TP covering LPNk to a victim slot
                of TPC
  determine PPNk corresponding to the LPNk
  update access time of the slot
  return PPNk

on Write Request of LPNk:
  if the requested LPNk is found on TPC
      select the TPC slot
  else
      load the TP covering LPNk to a victim slot
                of TPC
  allocate a free flash page PPNk
  update the corresponding LPNk entry
                of the selected slot as PPNk
  set the slot as dirty
  update access time of the slot
  return PPNk
```

TP의 캐시 항목 관리는 기본적으로 LRU 정책에 기초해서 이루어진다. 캐시에 새로운 TP를 적재하는 과정에서 제거대상 희생 항목을 선택하고 새로운 항목을 추가하는 과정은 알고리즘 2와 같다. 제거 대상을 고를 때 수정되지 않은(Clean) 항목을 우선적으로 선택하는데 이들은 TP를 플래시에 저장할 필요가 없고 그냥 제거하는 것으로 충분하기 때문이다. 제거 대상으로 수정된(Dirty) 항목이 선택되었다면 수정된 TP 내용을 새로운 플래시 페이지(TPN)를 할당하여 기록하고 이 TPN을 TD의 해당 항목에 갱신하여 기록한다.

Algorithm 2. TP Loading Algorithm Based on LRU

```
load the TP covering LPNk to a victim slot of TPC:
  if (there are empty slots)
      select any empty slot as victim
  else
      if (there are clean slots)
          select the oldest clean slot as victim
      else
          select the oldest dirty slot as victim
          save the TP of the victim
                        to a free flash page P
          update the corresponding TD entry as P
  load the TP covering LPNk to the victim slot
  set the slot as clean
  return the victim slot index
```

2. Additional Performance Enhancements

특정 LPN에 대응되는 TPC 항목을 검사하는 시간을 단축하는 방안으로서 TD에는 TPN과 TCI를 선택적으로 기록할 수 있도록 한다. TPC에 저장되지 않은 TP들에 대해서는 정상적으로 그 플래시 페이지 주소인 TPN을 기록하지만, TPC에 저장되어 있는 TP들에 대해서는 TPC의 해당 인덱스(TCI)를 기록한다. TD에 기록된 것이 TPN인지 TCI인지를 구별하는 방법으로는 최상위 1비트를 이용하거나 별도의 플래그 비트를 사용할 수 있을 것이다. 페이지 매핑 과정에서 먼저 그림 4와 같이 LPN으로부터 TD 인덱스를 계산하고 TD의 해당 항목이 TCI이면 바로 TPC에서 해당 항목을 결정할 수 있으므로 TPC 전체를 검사하는 시간을 생략할 수 있다. 이와 관련하여 TPC에서 특정 항목을 제거하는 과정에서는 TD에 다시 원래의 TPN을 기록해야 하므로 TPC 각 항목에는 원래의 TPN을 기록하여 보관하고 있어야 한다. 이것이 그림3에서 TPC에 TPN 필드가 존재하는 이유다. 쓰기 작업을 처리하는 과정에서 TP가 변경되어 새로운 TPN에 기록되었다면 변경된 TPN을 이 필드에 관리한다.

쓰기 요청에 대한 성능 개선 방안으로서 TPC에 TP를 적재하는 작업을 지연(DTR: Delayed TP Read) 시키도록 한다. 특정 TP에 해당되는 페이지들에 모두 쓰기가 이루어진다면 TP의 PPN 값들은 모두 수정되므로 이전의 TP 내용을 읽어오는 수고는 생략할 수 있는 것이다. DTR을 적용하기 위해서는 알고리즘 2를 약간 수정해야 한다. 이 알고리즘이 호출될 때 읽기 처리를 위한 것인지 쓰기 처리를 위한 것인지를 구별할 수 있도록 하고, 쓰기 처리를 위한 요청에 대해서만 다음과 같이 변경한다. TP를 적재하는 작업에서 플래시의 TP를 읽어오는 과정은 생략하고 대신에 PPN들을 모두 무효(예를 들어서 -1)로 표시해 둔다. 이와 관련하여 TPC에서 제거대상으로 선택된 항목이 수정된 것이라면, 그 TP의 모든 PPN들을 검사하여 하나라도 무효인 것이 남아있는 경우에 한해서 이 시점에 플래시에 저장되어 있던 TP를 읽어 와서 무효인 것들을 보충한 다음에 완성된 TP를 새로운 플래시 페이지에 기록하도록 한다. 실제로 널리 사용되는 트레이스들을 적용해 보면 DTR을 적용함으로써 플래시의 TP 읽기 횟수를 줄일 수 있음을 확인할 수 있는데, 그 효과에 관해서는 성능평가에 설명한다.

IV. Performance Evaluation

성능평가를 위해서 실제 시스템에서 수집한 트레이스 정보를 활용하였다. 대표적으로 많이 사용되는 것으로서 UMass 트레이스 [11] 파일의 Financial 1과 2, 및 WebSearch 1을 사용하였다. WebSearch 2와 3은 특성이 1과 거의 유사하므로 생략하였다. 표 1은 이들의 특성을 보여준다. Financial 1과 2는 OLTP 처리를 위한 응용에서 수집한 것으로서 읽기와 쓰기가 적절히 섞여있으며, 그 중에서도 1이 쓰기 요청 비율이 훨씬 높다. WebSearch 1은 웹의 검색위주로 이루어지는 응용에서 수집한 것으로서 읽기가 압도적으로 많이 이루어지며, 요청간 평균 간격 (average inter-arrival time)은 Financial 1과 2보다 훨씬 짧다.

Table 1. Characteristics of Traces

Trace	Write Request Ratio	Average Inter-arrival Time
Financial 1	76.8%	8.2ms
Financial 2	17.7%	11.1ms
WebSearch 1	0.02%	3.0ms

TPC-FTL의 목적은 DFTL과 같이 페이지 매핑을 기본으로 하되 매핑 정보를 위한 캐시의 검사 시간을 현실적으로 적용이 가능한 수준으로 줄이면서도 응답시간을 적절하게 유지하는 것이다. 그림 5는 트레이스 별로 평균 응답시간을 보여준다. 제안한 TPC-FTL의 응답시간을 비교하기 위해서 PM과 기존의 DFTL을 비교대상으로 하였다. PM(Page Mapping)은 최적의 비교대상으로 삼기위한 것으로서 페이지 매핑 정보를 전부 RAM에 저장한 것을 가정하였으며, 따라서 매핑 정보를 위한 플래시 페이지 읽기나 쓰기 오버헤드가 없다. 응답시간 평가에는 오직 플래시의 읽기와 쓰기를 위한 시간만 반영하였으며 캐시를 검사하는 시간 등의 SSD 제어기가 알고리즘을 수행하면서 소모하는 시간은 반영하지 않았다. 따라서 DFTL은 캐시 검사에 많은 시간을 소모하지만 본 응답시간 평가에는 포함시키지 않았다. 페이지의 크기는 2KB, 한 블록 당 페이지 수는 64, 캐시를 위한 RAM은 128KB를 적용하였다. NAND 플래시의 속도는 한 페이지 읽기에 25us, 한 페이지 쓰기에 200us, 그리고 한 블록 지우기에는 1500us를 적용하였다.

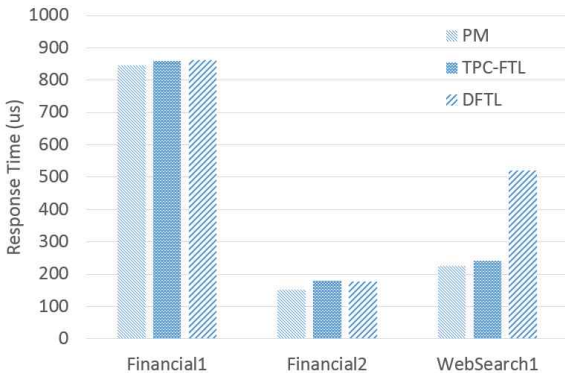
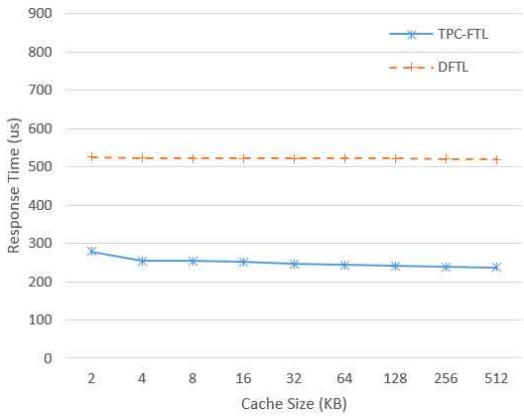


Fig. 5. Comparison of Average Response Time

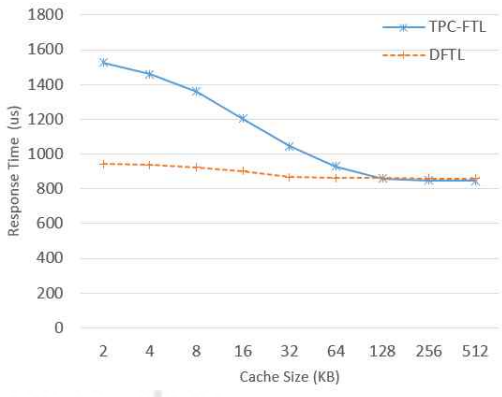
전체적으로 TPC-FTL은 최적에 해당하는 PM에 비해서 응답시간에 약간(15~27us)의 지연이 있지만 PM이 페이지 매핑 테이블 전체를 RAM에 관리하는 데 비해서 TPC-FTL은 TD와 128KB의 캐시를 위한 RAM만 사용한다. TD는 한 항목 당 LPN 512개에 대응되므로 PM의 페이지 매핑 테이블에 비해서 1/512 만큼의 RAM만 필요로 한다.

TPC-FTL과 DFTL의 응답시간을 비교하면 Financial 1과 2에서는 거의 비슷한 결과를 보여주었으며, WebSearch에서는 응답시간이 절반 정도로 대폭 개선되었다. 이러한 성과는 공간적 지역성 특성이 반영된 결과이다. 응용 프로세스들이 SSD의 페이지들을 접근하는데 있어서 일반적으로 시간 지역성과 공간 지역성이 모두 나타난다. 시간 지역성은 최근에 사용된 페이지는 조만간 다시 사용될 가능성이 높다는 것인데, 이 특성을 캐시에 반영한 것이 LRU 정책이다. 공간 지역성은 최근 사용된 페이지의 인근에 해당하는 페이지들도 다시 사용될 가능성이 높다는 것인데, DFTL에서는 LPN 단위로 캐시에 등록하기 때문에 공간 지역성을 활용하지 못하고 있다. TPC-FTL은 TP 단위로 캐시에 등록하므로 특정 LPN이 사용되었다면 동일한 TP내의 인접한 LPN들이 한꺼번에 캐시에 등록되어서 공간 지역성이 잘 반영되고 있다. 특히 WebSearch와 같이 파일들을 순차적으로 읽기를 하는 응용들에서는 공간 지역성이 그 효과를 잘 발휘한다.

그림 6은 주소변환 캐시의 크기에 따라서 TPC-FTL과 DFTL의 응답시간의 변화를 비교한 것이다. 여기에도 앞에서와 동일한 파라미터들을 적용하였다. WebSearch1의 경우에는 전체적으로 캐시의 크기에 별로 영향을 받지 않는다. 응용 프로세스에서 검사대상 파일들을 순차적으로 읽기를 요청하는 특성 때문에 최근에 요청된 페이지가 다시 요청될 확률이 낮아서 시간적 지역성이 낮다. 따라서 캐시용량이 늘어나더라도 별 효과가 없을 것이다. 대신에 인접한 페이지들을 연속해서 읽기 요청할 가능성이 높으므로 공간 지역성이 높고 따라서 TPC-FTL이 전체적으로 DFTL에 비해서 우수한 응답시간을 보여준다.



(a) WebSearch1



(b) Financial1

Fig. 6. Response Time Related to Cache Size

그림 6 (b)의 Finalcial1은 주소변환 캐시의 용량이 늘어남에 따라서 응답시간이 점진적으로 단축되는 것을 보여준다. 그러나 DFTL의 경우에는 캐시 용량이 32KB 이상이 되면 더 이상 응답시간이 개선되지 않는데, 이것은 시간적 지역성이 한정되어 있기 때문에 일정한 용량 이상의 캐시는 의미가 없어지기 때문이다. TPC-FTL의 응답시간은 캐시 용량이 작을 때에는 DFTL에 비해서 응답시간이 현저히 길지만 캐시의 용량이 128KB 정도 이상이면 DFTL과 유사한 성능을 보여준다. 여기서 다시 한 번 강조할 점은, 2장에서 분석한 바와 같이 DFTL은 캐시 항목들을 검사하는 데에 비현실적으로 많은 시간을 소모하므로 실제 시스템에 그대로 사용하는 데에는 무리가 있다는 점이다. 이에 비해서 TPC-FTL은 캐시의 항목수가 많지 않으므로 이러한 문제가 자연스럽게 해결된다. 최근에 SSD 관련 기술이 급격히 발전하면서 100GB 이상의 대용량 모델이 보편화되고 이들에는 100MB 이상의 RAM이 장착되어 있으므로 128KB 이상의 RAM을 주소변환 캐시에 사용하는 것은 아주 자연스러운 선택일 것이다.

플래시 메모리에 저장된 TP를 읽는 작업을 지연시키는 DTR의 효과는 의미는 있으되 그리 크지는 않다. UMass 트레이스 들을 활용하여 평가해보면 Financial 1은 TP 읽기 회수를 0.85% 정도 감소시키는 정도로 효과가 크지 않고, 이것이 Financial 2의 경우에는 효과가 더욱 미미하며, 읽기 위주의 WebSearch 1의 경우에는 효과가 거의 나타나지 않는다. DTR은 기본적으로 쓰기 요청이 많이 발생하는 용용에 있어서 동일한 TP의 PPN들이 모두 새로운 페이지들로 변경되는 경우에만 그 효과가 크게 나타날 수 있을 것이다.

V. Conclusions

본 논문에서는 대용량 SSD에서 주소변환 캐시의 관리방법을 개선하여 캐시 검사에 소요되는 시간을 현실적으로 사용이 가능한 수준으로 단축하면서도 응답속도 면에서는 기존의 방식에 비해서 개선된 결과를 보여주었다. SSD에 사용되는 플래시 메모리에서는 제자리에 새로운 데이터를 기록하는 것이 허용되지 않으므로 SSD 내에 FTL 계층을 두어서 여기서 읽기나 쓰기 요청이 온 LPN을 PPN으로 변환하여 사용한다. 이러한 주소 변환을 위해서 페이지 변환 테이블을 사용한다.

DFTL은 주소변환을 페이지 단위로 처리하는 페이지 매핑 방식을 사용하는데, RAM 요구량을 줄이기 위해서 변환 테이블을 플래시의 TP들에 기록하고 자주 사용되는 LPN들은 RAM 상의 주소 변환 캐시에 저장해두고 사용한다. 그런데 캐시의 크기가 일정수준 이상으로 커지면 DFTL은 캐시 항목들을 검사하는 데에 비현실적으로 많은 시간을 소모하므로 실제 시스템에 그대로 사용하는 데에는 무리가 있다. 최근에 SSD 관련 기술이 급격히 발전하면서 100GB 이상의 대용량 모델이 보편화되고 이들에는 100MB 이상의 RAM이 장착되어 있으므로 128KB 이상의 RAM을 주소변환 캐시에 사용하는 것은 아주 자연스러운 선택이 되었으므로, 본 논문에서는 이 정도 이상의 RAM을 캐시로 활용하여 성능을 개선할 수 있음을 보여주었다.

본 논문에서 제안한 TPC-FTL은 주소 변환 캐시를 LPN 단위가 아니라 TP 단위로 관리함으로써 검사해야할 캐시의 항목수를 대폭 줄여서 캐시 검사에 소요되는 시간을 현실적으로 적용 가능한 수준으로 줄였다. 그러면서도 응답 속도는 기존의 DFTL에 비해서 나은 결과를 얻을 수 있었다. TP 단위로 캐시에 관리하는 것의 추가적인 장점으로는 응용들에서 플래시 페이지를 요청할 때 발생하는 공간 지역성을 적극 활용하는 효과가 있어서 읽기 위주의 응용에서는 DFTL보다 훨씬 단축된 응답시간을 보여주었다. 성능 평가 결과 쓰기가 많은 트레이스들에서는 캐시의 용량이 128KB 이상이면 DFTL과 동일한 수준의 응답시간을 보여주었고 읽기 위주의 트레이스에서는 응답시간이 DFTL의 50% 정도로 대폭 단축되었다.

본 논문에서는 TPC-FTL의 주소 변환 관점에 집중하여 기술하였는데, 실제 SSD에 적용하기 위해서 필요한 추가적인 기능들은 기존의 DFTL 및 관련 논문에서 제시하는 방법들을 그대로 사용해도 무방하다. 여기에는 무효화된 페이지들을 수집하여 재사용하는 기능과 전체 블록들 간에 지우기 회수를 평준화하는 기능 등이 포함된다.

REFERENCE

[1] S. Lee, D. Park, T. Chung, D. Lee, S. Park, H. Song, “A log buffer based flash translation layer using fully associative sector translation,” ACM Trans. Embedded Computing Sys. Vol. 6, No. 3, pp.1-27, 2007.

[2] A. Gupta, Y. Kim, and B. Urgaonkar, “DFTL: a Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings,” Proc. 14th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’09), New York, 2009.

[3] E. Goossaert, “Coding for SSDs – Part 2: Architecture of an SSD and Benchmarking,” http://codecapsule.com/2014/02/12/coding-for-ssds-part-2-architecture-of-an-ssd-and-benchmarking/

[4] “Yet another flash file system,” http://www.yaffs.net

[5] A. B. Bityutskiy, “JFFS3 design issues.” http://www.linux-mtd.infradead.org

[6] C. Lee, D. Sim, J. Hwang, and S. Cho, “F2FS: A New File System for Flash Storage”, Proc. 13th USENIX Conference on File and Storage Technologies (FAST ’15), pp.273-286, Feb. 2015.

[7] D. Park, B. Debnath, and D. Du, “CFTL: An

Adaptive Hybrid Flash Translation Layer with Efficient Caching Strategies," IEEE Trans. on Computers, pp. 1-15, Sep. 2011.

[8] C. Wang, W. Wong, "ADAPT: efficient workload-sensitive flash management based on adaptation, prediction and aggregation," Proc. IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), April 2012.

[9] J. Boukhobza, et al., "MaCACH: An Adaptive Cache-Aware Hybrid FTL Mapping Scheme Using Feedback Control for Effieient Page-Mapped Space Management," Journal of Systems Architecture, Elsevier, pp. 157-171, 2015.

[10] Samsung Electronics, "Samsung SSD 850 PRO Data Sheet, Rev.2.0," Jan. 2015. http://www.samsung.com/global/business/semiconductor/minisite/SSD/kr/html/ssd850pro/specifications.html

[11] "Storage Traces of UMass Trace Repository," http://traces.cs.umass.edu/index.php/Storage/Storage

Authors

Yong-Seok Kim received B.S. degree in Oceanography from Seoul National University, Korea, in 1984, and M.S. and Ph.D. degrees in Electric and Electronics Engineering from KAIST (Korea Advanced Institute of Science and Technology), Korea, in 1986 and 1989, respectively. Dr. Kim is a professor in Department of Computer and Communications Engineering at Kangwon National University, Kangwon-do, Korea, from 1995. He was a research staff of KETI (Korea Electronics Technology Institute) in 1994, and KITECH (Korea Institute of Industrial Technology) from 1990 to 1993. He is interested in system software for real-time and embedded systems, and internet of things.

Hwan-Pil Choi received the B.S. and M.S. degrees in Dept. of Computer and Communications Engineering from Kangwon National University, Korea, in 2009 and 2011, respectively. He is currently a PhD Student in the Department of Computer and Communications Engineering, Kangwon National University. He is interested in real-time systems.