Journal of The Korea Society of Computer and Information
Vol. 22 No. 12, pp. 9-15, December 2017
www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.12.009

An Efficient Flash Translation Layer Considering Temporal and Spacial Localities for NAND Flash Memory Storage Systems

Yong-Seok Kim*

Abstract

This paper presents an efficient FTL for NAND flash based SSDs. Address translation information of page mapping based FTLs is stored on flash memory pages and address translation cache keeps frequently accessed entries. The proposed FTL of this paper reduces response time by considering both of temporal and spacial localities of page access patterns in translation cache management. The localities of several well-known traces are evaluated and determine the structure of the cache for high hit ratio. A simulation with several well-known traces shows that the presented FTL reduces response time in comparison to previous FTLs and can be used with relatively small size of caches.

▸Keyword: Temporal Locality, Spacial Locality, FTL, Page mapping, Flash memory

I. Introduction

최근에 NAND 플래시 메모리를 기반으로 하는 SSD (Solid State Drive)가 기존의 HDD(Hard Disk Drive)를 급격히 대체해 나가고 있다. 특히 스마트폰이 폭발적으로 보급되는 과정에서 휴대형 저장장치로서의 플래시 메모리 시장의 규모는 급격하게 커지고 있으며, 이제는 서버 컴퓨터의 대규모 저장장치 영역에서도 플래시 메모리의 비중이 날로 증가하고 있다. SSD는 HDD에 비해서 가볍고 처리속도가 빠르며 전력소모도 적고 충격에도 강한 장점이 있다. 핵심적인 단점으로 꼽히는 가격 문제도 이제는 반도체 제조기술의 발달로 점점 극복되어가고 있는 것이다.

SSD는 HDD에 비해서 쓰기 과정이 확연히 다르다. HDD는 임의의 위치에 임의의 데이터를 바로 기록할 수 있지만 SSD는 기존의 데이터를 지운 후에 새로운 데이터를 기록할 수 있다는 제한이 있다. NAND 플래시 메모리는 데이터를 기록하거나 읽기 위한 최소 단위가 페이지이고 지우기는 일정한 개수의 페이지들의 집합인 블록 단위로만 가능하다. 따라서 한 페이지를 새로운 데이터로 갱신하기 위해서는 해당 블록의 모든 페이지들을 읽어내서 갱신할 페이지만 수정하고, 해당 블록을 지운 후에 페이지들을 전부 다시 기록해야 하는 것이다. 이러한 문제를 해결하는 한 방법으로서 SSD 내부에서 실제로 기록되는 위치를 결정하고 그 위치 정보를 내부적으로 적절히 관리하는 것이다. SSD에서 이러한 역할을 하는 부분을 FTL(Flash Translation layer)이라고 한다.

FTL에서는 파일시스템으로부터 요청된 논리적 페이지 번호 (LPN: Logical Page Number)를 실제로 기록된 플래시 메모리 페이지 번호인 물리적 페이지 번호 (PPN: Physical Page Number)로 적절히 변환하여 처리한다. 주소 변환을 위한 정보는 FTL에서 관리해야 하는데 블록단위로 관리하는 방법인 블록 매핑 방법과 페이지 단위로 관리하는 방법인 페이지 매핑 방법이 있다. 페이지 매핑은 LPN마다 실제로 기록된 PPN을 주소변환 정보로 관리한다. 파일 시스템으로부터 페이지 읽기 요청이 있으면 그 LPN에 대응되는 PPN을 찾고 그 페이지를 읽어낸다. 페이지 쓰기 요청에 대해서는 새로운 물리적 페이지에 기록하고 그 주소인 PPN을 주소변환 정보에 반영한다. 이 방식은 단순하기는 하지만 매핑 정보를 관리하기 위한 RAM의 용량이 매우 커야 한다는 근본적인 단점이 있다. 이러한 단점을 해결하는 방법으로서 DFTL에서는 주소변환 정보를 모두 플래시 메모리 페이지에 기록하고 일부만 캐시 형태로 RAM에 관리하는 방법을 사용하였다[1].

블록 매핑은 주소변환 정보를 블록 단위로 관리하는 것으로서 논리적 블록 번호(LBN: Logical Block Number)를 물리적 블록

• First Author: Yong-Seok Kim, Corresponding Author: Yong-Seok Kim
*Yong-Seok Kim (yskim@kangwon.ac.kr), Department of Computer and Communications Engineering, Kangwon National University
• Received: 2017. 11. 06, Revised: 2017. 11. 27, Accepted: 2017. 12. 14.

번호(PBN: Physical Block Number)로 변환한다. 하나의 논리적 블록에 포함되는 페이지들은 하나의 물리적 블록에 순서대로 배치되는 것을 전제로 한다. LPN을 블록 번호 부분인 LBN과 블록 내에서 해당 페이지의 위치인 페이지 오프셋(PO: Page Offset) 부분으로 구분하고 LBN 부분만 PBN으로 변환하고 PO는 그대로 사용한다. 이 방법은 매핑 정보를 블록 단위로만 관리하므로 필요한 RAM의 용량이 페이지 매핑에 비해서 작지만, 한 페이지에 새로운 데이터를 기록하기 위해서는 블록 전체를 새로운 블록에 기록하고 이전 블록은 지우기를 하는 오버헤드가 매우 크다. 이러한 단점을 해결하기 위해서 페이지들을 임시로 저장하는 로그 블록을 활용하고 로그 블록이 다 차면 로그 블록의 최신 페이지들과 기존 블록의 남은 유효 페이지들을 병합하여 새로운 블록에 기록하도록 한다[2-4].

블록 매핑은 페이지 매핑에 비해서 유효 페이지 복사를 위한 오버헤드가 크고 블록의 지우기가 빈번하게 발생한다는 단점이 있다. 본 논문은 DFTL과 마찬가지로 페이지 매핑을 기반으로 하되 RAM에 관리하는 주소변환 캐시의 관리 방법을 개선하여 SSD의 성능을 높이는 것을 목표로 한다. RAM의 용량이 충분하다면 페이지 매핑 정보 전부를 RAM에 관리하는 것이 최적이겠지만, 한정된 용량의 RAM만 사용할 수 있다면 주소변환 캐시의 성공률을 최대한 높이는 것이 매우 중요하다.

II. Related Works

응용 프로세스들이 파일에 대하여 페이지들을 요청할 때 대체로 시간적 지역성과 공간적 지역성이 동시에 있으므로 이러한 특성을 잘 활용해야 주소변환 캐시의 성공률을 높일 수 있다. 그림 1은 실제 시스템에서 이루어지는 페이지 읽기/쓰기 요청에 대한 트레이스를 분석한 한 예로서 요청된 논리적 페이지들의 분포를 보여준다[5]. 가로로 점들이 인접해서 이어진 것은 이전에 요청된 페이지가 이후에도 계속 요청되는 시간적 지역성을 의미한다. 뚜렷하지는 않지만 곳곳에 세로로 점들이 이어진 부분들은 비슷한 시점에 인접한 페이지들이 요청되는 공간적 지역성을 의미한다. 우상향 사선들은 인접한 페이지를 순차적으로 접근을 하는 부분들로서 공간적 지역성을 의미한다.

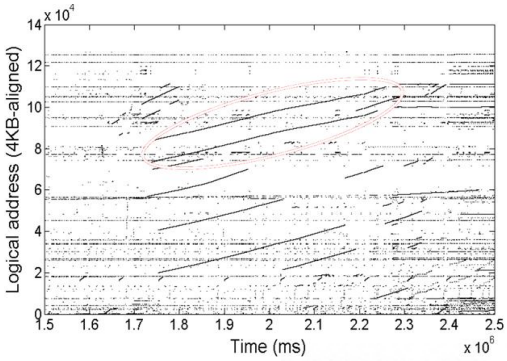

Fig. 1. Access Distribution of UMass Financial1 [5]

DFTL은 시간적 지역성을 활용하는 대표적인 예로서 페이지 매핑을 기반으로 하고 있다[1]. 그림 2와 같이 모든 주소변환 정보를 플래시 메모리 페이지에 TP(Translation Page)로 저장하며 하나의 TP에는 PPN들이 배열 형태로 기록된다. RAM에는 TP들의 주소 목록을 GTD(Global Translation Directory)에 기록한다. 주소 변환이 필요할 때마다 TP에서 변환 정보를 읽어오면 시간이 많이 소모되므로 자주 사용되는 논리적 페이지들에 대한 변환 정보를 <LPN, PPN> 쌍으로 캐시(CMT: Cached Mapping Table)에 관리하여 시간을 단축한다. 만약 주소변환 캐시의 성공률이 1에 가깝다면 주소변환을 위해서 TP를 읽고 쓰는 오버헤드는 매우 작아서 무시할 수 있는 수준이 될 것이다.

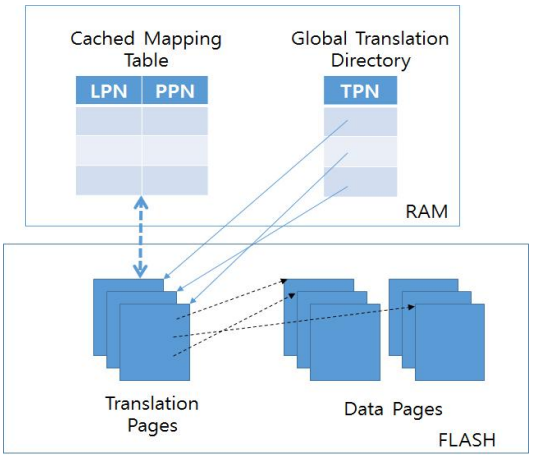

Fig. 2. Address Translation of DFTL

DFTL은 주소 변환 캐시를 LPN 단위로 관리하므로 시간적 지역성에 대한 특성은 잘 활용하지만 공간적 지역성은 활용하지 못하고 있다. 특정 논리적 페이지가 사용되었다면 공간적 지역성에 의해서 인접한 논리적 페이지들도 사용될 가능성이 높지만 그러한 특성을 활용하지 못하고 있는 것이다. 공간적 지역성을 활용하기 위해서 주소변환 캐시의 항목을 TP 단위로 관리하는 방법을 적용한 것으로는 CDFTL, S-FTL, TPFTL, TPC-FTL 그리고 STP-FTL 등이 있다[5-9]. 하나의 TP에 포함된 인접한 LPN들은 한꺼번에 캐시에 등록되므로 공간적 지역성을 활용하게 되는 것이다.

CDFTL에서는 캐시를 CMT와 CTP(Cached Translation Page)의 두 단계로 구분하고 CTP는 TP 단위로 캐시를 관리한다[6]. CMT 관리는 기본적으로 DFTL과 동일하지만 캐시에서 제거할 항목은 우선 CTP로 이동해서 보관하고, CTP에서 제거해야 하는 시점에 TP에 기록한다. 필요한 LPN이 CMT와 CTP 어디에도 없으면 TP를 읽어서 CMT와 CTP에 모두 등록한다. CDFTL의 문제점으로는 캐시 관리가 이원화 되어 복잡하고 CTP를 보조수단으로만 활용한다는 점이다. 또한 CMT에는 쓰기가 이루어진 후 오랫동안 사용되지 않는 페이지의 항목이 계속 남아있을 수 있다는 것도 지적된 바 있다[5]. S-FTL은 주소변환 캐시를 TP 단위로 관리한다. 인접한 LPN들이 인접한 PPN들에 순차적으로 기록되는

경우에 대해서는 TP를 캐시에 표현하기 위한 RAM을 절약할 수 있는 방법으로서 비트맵을 선택적으로 활용한다. 캐시에서 제거되는 TP는 플래시 페이지에 바로 기록하는 것이 아니라 수정된 항목이 5% 이내이면 일단 별도의 LPN 별 캐시에 임시로 보관하는데 이 부분은 DFTL의 CMT와 마찬가지의 역할을 한다[7]. TPFTL은 TP를 그대로 캐시에 기록하면 RAM 공간 낭비라는 관점에서 DFTL 방식에 캐시의 표현을 TP별로 모아서 압축하여 표현함으로써 캐시를 효율적으로 활용하도록 하였다[5]. 그러나 캐시를 압축하여 표현하는 것일 뿐이므로 DFTL과 마찬가지로 공간적 지역성을 활용하지는 못하고 있다. 다만 캐시에 여유가 생기면 인접한 LPN들도 함께 캐시에 등록하도록 하고 있다. TPC-FTL은 주소변환 캐시를 단순히 TP 단위로만 관리한다. 따라서 캐시 관리가 매우 단순하다. 적절한 캐시의 용량을 분석하기 위해서 대표적인 트레이스들을 적용한 결과 128K 바이트 이상의 RAM을 캐시로 사용하면 우수한 성능을 얻을 수 있음을 보여주었다[8]. STP-FTL은 주소변환 캐시를 두 단계로 나누어서 TP 단위로 관리하는 부분과 TP를 일정한 크기로 나눈 단위들을 관리하는 부분으로 구성한다[9].

본 논문에서는 한정된 크기의 캐시를 사용하되 시간적 지역성과 공간적 지역성을 적절히 조화롭게 활용함으로써 캐시의 성공률을 높이도록 한다. 기존의 논문들은 시간적 지역성 관점에서 LPN별로 캐시에 관리하거나 공간적 지역성 관점에서 TP 단위로 캐시에 관리하는 방법을 적용하였지만, 두 가지를 조화시키는 점이 소홀했다. 캐시를 단순히 TP 단위로 관리하는 것은 나름의 장점이 있다. 특정 TP의 일부 항목만 수정해야 한다면 먼저 기존의 TP 내용을 읽어 낸 후에 필요한 부분만 수정하여 기록해야 한다. 반면에 TP 단위로 캐시를 관리한다면 TP의 내용을 그대로 새로운 페이지 기록하면 되므로 기존의 내용을 읽어내는 작업을 생략할 수 있어서 효율적일 것이다. 본 논문에서는 캐시를 무조건 TP 단위로만 관리할 것이 아니라 어느 정도의 크기로 분할하여 관리하는 것이 효율적인지를 검토하였다. 시간적 지역성을 충분히 활용하려면 캐시에 항목수가 많을수록 유리할 것이고 공간적 지역성을 충분히 활용하려면 일정한 개수의 인접한 LPN들의 묶음을 단위로 관리하는 것이 유리할 것이다. 시간적 지역성을 충분히 활용하려면 캐시에 몇 개 정도의 항목들이 필요한지와 공간적 지역성을 충분히 활용하려면 캐시의 관리 단위의 크기를 어느 정도로 해야 하는 지를 대표적인 트레이스들을 활용하여 분석하고, 그 결과를 새로운 FTL에 반영하였다.

III. Cache Management and Locality Evaluation

1. Management of Address Translation Cache

본 논문의 TSL(Temporal and Spacial Locality) FTL에서는 주소변환 캐시를 그림 3과 같이 단순하게 TP를 일정한 크기로 나눈 부분 TPS (TP Segment) 들로 관리한다. LPN을 TPS의 크기로 나눈 몫이 TPSN(TPS Number)이고 그 나머지가 TPSO (TPS Offset)이다. TPS 하나에 포함되는 페이지 번호들의 개수를 K라고 하면, TPSN = LPN / K (나눈 몫)이고, TPSO = LPN % K (나눈 나머지)이다. 특정 LPN에 대한 접근 요청이 오면 TSL FTL에서는 그림 4와 같이 그 TPSN이 캐시에 있는지를 검사하고, 있다면 해당 TPS에서 TPSO의 항목을 선택하면 된다. 캐시에 없으면 GTD로부터 얻은 TP의 주소를 활용하여 플래시 메모리에서 TP를 읽은 다음 해당 TPS 부분만 선택하여 캐시에 등록한 후에 사용한다.

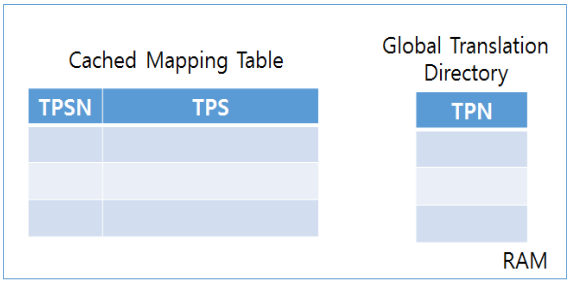

Fig. 3. Address Translation Cache of TSL FTL

```
TPSN = LPN / K; TPSO = LPN % K;
if (TPSN is not found on CMT)
  get TPN covering LPN from GTD;
  read TPS related to LPN;
  add <TPSN, TPS> on CMT;
get the TPS of TPSN;
if (the request is read)
  PPN = the TPSO entry of TPS;
  read the flash page of PPN;
else
  write the requested page on a new flash
    page PPN;
  update the TPSO entry of TPS with PPN;
```

Fig. 4. Action on a Request of LPN

캐시에 새로운 항목을 추가하기 위해서는 CMT에 빈 공간이 있어야 한다. 만약 빈 공간이 없으면 LRU (Least Recently Used) 정책에 따라 선택한 TPS 항목을 제거하는데 만약 이 TPS가 수정된 것이면 플래시 메모리의 TP에 기록하여야 한다. 이를 위해서는 그림 5와 같이 먼저 플래시 메모리에서 대응 TP를 읽어 내어서 해당 TPS 부분만 수정한 후에 새로운 페이지에 기록하고 그 주소를 GTD에 갱신한다. 만약 TPS가 하나의 TP와 일치한다면 플래시 메모리에서 읽어내는 작업은 생략할 수 있을 것이다. 기존의 FTL들에서 TP를 더 세분화하지 않고 TP 단위로 캐시를 관리한 이유는 이렇게 TP에 기록할 때 기존의 내용을 읽어낼 필요가 없기 때문이다.

```
if (CMT has no empty slot)
  select a victim TPS2 based on LRU;
  get TPN covering the TPS2 from GTD;
  read the TP of TPN and
    replace the victim TPS portion;
  set the slot as empty;
  write the TP on a new flash page TPN2
    and update the TPN2 on GTD;
add the requested <TPSN, TPS> on
  an empty slot;
```

Fig. 5. Addition of <TPSN,TPS> Entry on CMT

2. Evaluation of Temporal Locality and Spacial Locality

적절한 TPS의 크기를 결정하기 위해서는 TPS의 크기를 변화시켜 가면서 시간적 지역성과 공간적 지역성의 특성을 분석해야 하는데 UMass 트레이스를 활용하였다[10]. 이것은 대표적인 몇 가지 응용들을 실제 시스템에서 실행하면서 수집한 페이지 요청 패턴에 대한 정보이다. 그 중에서 Financial1과 Financial2, 및 WebSearch1을 사용하였다. Financial1은 OLTP 처리를 하는 응용에서 수집한 것으로서 페이지 쓰기 요청 비율이 77%로 매우 높다. Financial2는 페이지 쓰기 요청 비율이 18%로 비교적 적은 편이다. WebSearch1은 웹의 검색 위주로 이루어지는 응용에서 수집한 것으로서 페이지 읽기 요청이 99.8%로 압도적으로 많이 이루어진다. WebSearch2와 WebSearch3은 특성이 WebSearch1과 거의 유사하므로 생략하였다.

FTL의 성능 평가를 위하여 적절한 플래시 메모리 파라미터를 적용하여야 한다. 플래시 메모리의 칩 구성은 다양하지만 여기서 적용한 것으로는 전형적인 1G 바이트 NAND 플래시 메모리 칩의 동작 특성 파라미터를 적용하였다. 한 블록은 64개의 페이지로 구성되고, 한 페이지의 크기는 2K 바이트이다. 칩의 동작 속도는 한 페이지 읽기에 25us, 한 페이지 쓰기에 200us, 그리고 한 블록 지우기에 1500us이다[11]. 페이지 번호는 LPN과 PPN 모두 32비트 (4바이트)로 표현된다고 가정하였다. 그러면 하나의 TP에는 512개의 PPN들이 기록된다.

동일한 크기의 캐시에 대하여 TPS의 크기를 줄이면 캐시 항목 개수는 늘어나게 된다. 예를 들어서 캐시의 크기가 16K 바이트일 경우에 2K 바이트 크기의 TP 단위로 관리하면 (하나의 TP는 하나의 TPS이며 하나의 TPS에는 512개의 인접한 LPN들을 위한 항목들이 포함됨) 캐시 항목이 8개이고, TP의 절반 크기 단위로 관리하면 (하나의 TP는 2개의 TPS로 분리되며 하나의 TPS에는 256개의 인접한 LPN들을 위한 항목들이 포함됨) 캐시 항목이 16개가 된다. 역으로 말해서 캐시 항목수가 8이면 하나의 TPS에는 512개의 인접한 LPN들을 포함하며, 16이면 256개, 32면 128개의 인접한 LPN들이 포함된다. 항목수가 4096이 되면 TPS 당 1개의 LPN만 포함된다. 동일한 크기의 캐시 메모리에 대하여 항목수가 늘어날수록 하나의 TPS에 포함되는 인접한 LPN들의 개수가 줄어드는 것이다.

그림 6은 Financial1에 대하여 캐시의 항목 개수에 따른 성공률을 보여준다. 캐시의 크기가 비교적 작은 4K 바이트일 때에는 항목 수가 64개일 때 성공률이 가장 높게 나오고, 캐시의 크기가 16K 바이트이면 항목수가 128개일 때 성공률이 가장 높다. 이 정점을 기준으로 캐시의 항목수가 줄어들면 성공률이 점점 감소하는데, 이것은 시간적 지역성을 충분히 활용하지 못하기 때문이다. 반대로 캐시의 항목수가 더 늘어나도 성공률이 낮아지는데 이것은 캐시 항목인 하나의 TPS에 포함되는 LPN 개수가 줄어들어서 공간적 지역성을 활용하지 못하게 되기 때문이다. 즉 시간적 지역성에 대한 긍정적인 효과에 비해서 공간적 지역성에 대한 부정적 효과가 더 크기 때문이다.

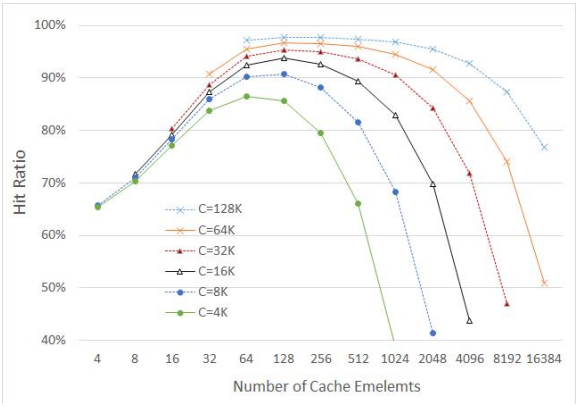

Fig. 6. Cache Hit Ratio of Financial1

이러한 특성은 그림 7과 같이 WebSearch1에서도 비슷하게 나타난다. 여기서 주목할 점은 WebSearch1은 페이지 읽기 요청의 비율이 99.8%로서 읽기 위주의 응용이고 Financial1은 그 비율이 23%로서 쓰기 위주의 응용이며, 응용 분야가 하나는 웹 검색이고 다른 하나는 OLTP로서 서로 전혀 다르지만 이들의 지역성에 대한 특성은 유사하게 나타나고 있다는 점이다.

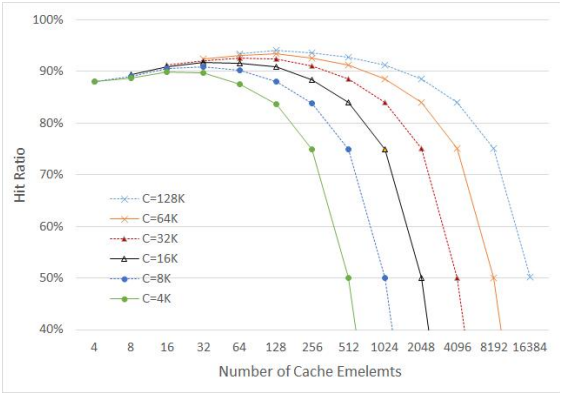

Fig. 7. Cache Hit Ratio of WebSearch1

이러한 지역성에 대한 특성을 바탕으로 캐시에 관리해야할 항목인 TPS의 적절한 개수와 크기를 결정해야 한다. 이러한 파라미터는 캐시를 위한 RAM의 용량에 따라서 달라진다. 따라서 캐시의 크기에 따라서 적절한 파라미터를 조절할 필요가 있는

것이다. 캐시에서 항목 검사에 실패할 경우에는 TP를 읽거나 기록하는 오버헤드가 발생한다. 따라서 캐시의 성공률이 높다고 무조건 성능이 높아지는 것이 아니라 이러한 오버헤드를 포함하여 적절한 파라미터를 선택해야 한다.

IV. Performance Evaluation

1. TPS Size of TSL FTL

캐시의 성공률은 SSD의 응답 성능에 직접적으로 영향을 미친다. 캐시 검사에서 실패하면 플래시 메모리로부터 TP를 읽는 오버헤드가 추가되고, 캐시에서 일부 항목을 제거할 때에는 수정된 항목이라면 플래시 메모리에 TP를 기록하는 오버헤드가 추가된다. TP를 수정할 때에 만약 TPS가 TP의 크기와 일치하지 않으면 TP를 기록하기 전에 이전의 내용을 읽는 오버헤드도 추가된다. 이전의 TP를 읽어 낸 다음에 TPS의 수정 내용을 반영한 후에 다시 기록해야 하는 것이다. 이렇게 플래시 메모리의 TP를 관리하기 위한 오버헤드는 트레이스별로 차이가 있을 것이며 캐시의 항목 수에 따라서도 성공률의 차이로 인해서 달라질 것이다.

그림 8은 캐시의 크기가 16K 바이트일 때 TP 관리를 위한 오버헤드를 트레이스 별로 비교하여 보여준다. Financial1은 항목 수가 32 이하로 줄어들면 오버헤드가 급격히 늘어나고, 반면에 WebSearch1은 항목수가 512 이상으로 늘어나면 (TPS의 크기가 줄어들면) 오버헤드가 급격히 늘어난다. 그러나 트레이스에 상관없이 대체로 중간에 최적의 지점이 있다. 따라서 트레이스에 상관없이 적절한 TPS의 크기를 결정하기 위해서는 이들의 평균이 의미가 있을 것이다. 평균 오버헤드는 캐시 항목수가 128개일 때 가장 작게 나온다. 16K 바이트 크기의 캐시를 128개의 TPS로 나누면 TPS 하나당 128 바이트가 되고, 이것은 2K 바이트 크기의 TP를 32개로 분할한 것을 의미한다. 128 바이트 크기의 TPS에는 32개의 인접한 LPN 항목들을 포함한다.

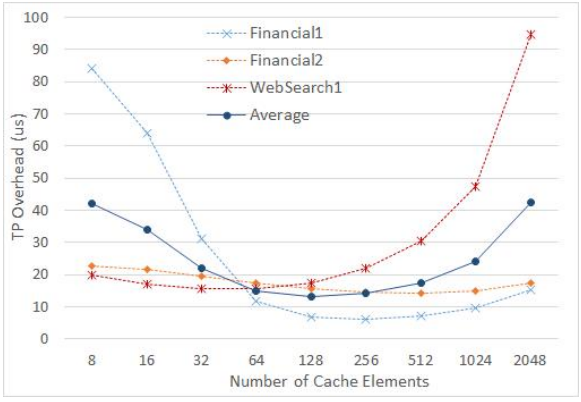

Fig. 8. TP Management Overhead (Cache Size: 16K Bytes)

그림 9는 트레이스들의 평균 오버헤드에 대하여 캐시의 크기별로 캐시의 항목 수에 따른 변화를 보여준다. 최적의 캐시 항목 수는 캐시 크기가 4K 바이트일 때에는 64개, 16K 바이트일 때에는 128개, 64K 바이트일 때에는 256개 정도로 나타난다. 따라서 캐시 크기에 따른 적절한 캐시 항목 수는 4K 바이트일 때에 64개를 기준으로 하여 캐시 크기가 4배로 증가할 때마다 캐시 항목수를 2배로 늘이는 것이 적절함을 암시하고 있다. 따라서 TSL FTL에서는 이 기준으로 TPS의 크기를 결정한다. 캐시의 크기가 4K 바이트일 때에는 TPS 크기는 16개의 인접한 LPN 항목들을 포함하도록 하여 64개의 TPS들을 캐시에 관리한다. 캐시의 크기가 4배로 증가할 때마다 TPS의 크기를 2배로 키우고 캐시의 항목수도 2배로 증가하도록 한다.

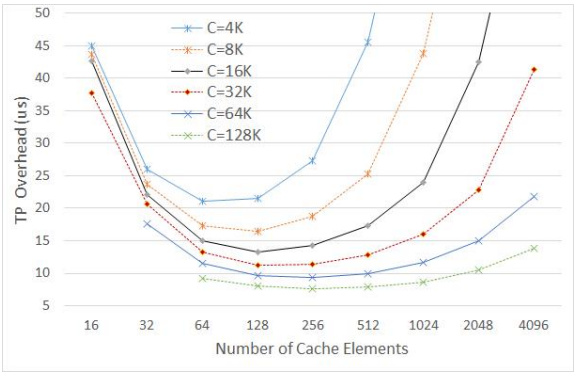

Fig. 9. Average TP Management Overhead

한 가지 더 중요하게 검토해야 할 사항은 캐시에 항목수가 많을수록 캐시를 검사하는 데에 많은 시간이 소모된다는 점이다. 경험적인 관점에서 보면 1000개 이상의 캐시 항목들을 검사하는 시간은 SSD의 성능에 영향을 미칠 수 있을 정도가 될 것이다. 그림 9에서 캐시의 크기가 128K 바이트 이상으로 크면 캐시의 항목 개수가 256에서 더 늘어나더라도 오버헤드를 감소시키는 효과가 미미함을 알 수 있다. 따라서 실용적인 관점에서 최대 256개 이내로 제한하는 것이 합리적일 것이다.

2. Performance Evaluation of TSL FTL

TSL FTL의 성능을 비교하기 위하여 트레이스 별로 평균 응답시간을 시뮬레이션을 통하여 평가하였다. 평균 응답시간에는 페이지 읽기와 페이지 쓰기가 모두 포함되며 FTL에서 발생하는 큐잉 지연 시간까지 포함된 것이다. 비교대상으로서 PPM(Pure Page Mapping)은 페이지 매핑을 그대로 적용한 것으로서 모든 주소변환 정보가 RAM에 저장되어 있고 따라서 주소변환을 위한 오버헤드가 없는 것을 의미한다. PPM은 필요한 RAM의 용량이 너무 커서 현실적으로 적용할 수는 없지만 FTL들의 성능평가에 있어서 최적의 비교 값으로 사용하였다. TPC는 캐시를 단순히 TP 단위로 관리하는 것이며 CDFTL과 S-FTL도 약간의 차이는 있겠지만 대체로 여기에 해당한다. DFTL은 캐시에 <LPN, PPN> 쌍으로 주소변환 정보를 관리하므로 하나의 캐시 항목에는 8바이트가 사용된다. 따라서 캐시의 크기가 4K 바이트일 때에는 DFTL의

캐시 항목수는 512개이고 캐시의 크기가 증가함에 따라 항목수도 비례하여 증가한다.

그림 10은 캐시의 크기가 4K 바이트로 작은 경우에 대한 응답 시간을 보여준다. TSL FTL은 세 가지 트레이스들 모두에서 DFTL과 TPC에 비해서 우수한 성능을 보여주며, 최적의 비교대상인 PPM에 가장 근접한 것이다. 이렇게 캐시의 크기가 작으면 TPC는 WebSearch1에서는 DFTL보다 우수하지만 Financial1에서는 성능이 좋지 못하다.

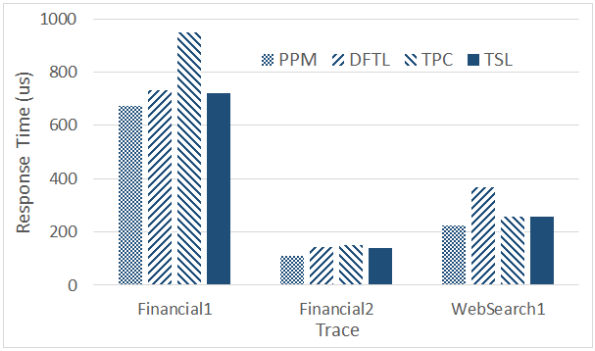

Fig. 10. Comparison of Response Times (Cache Size: 4K Bytes)

그림 11은 캐시의 크기가 64K 바이트로 늘어났을 때의 응답시간을 보여준다. 여전히 TSL FTL이 가장 우수한 성능을 보여주며 최적의 비교대상인 PPM에 비해서도 평균 7.8%의 성능 저하만 있을 뿐이다. 캐시가 4K 바이트일 때와는 응답시간이 평균 6.8% 정도만 차이가 있으므로 캐시의 크기를 4K 바이트 정도로 작게 잡아도 실용적으로는 별 문제가 없을 것이다.

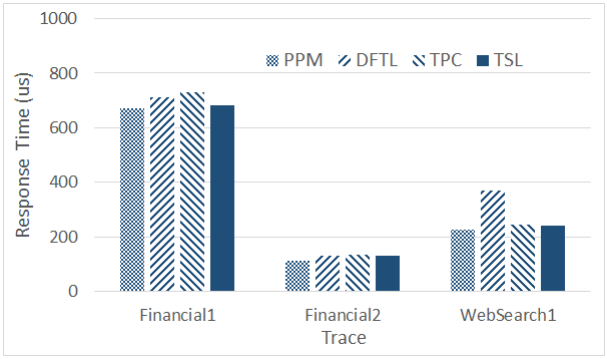

Fig. 11. Comparison of Response Times (Cache Size: 64K Bytes)

참고로 DFTL은 캐시의 크기를 64K 바이트로 늘리면 Financial1의 응답시간이 대폭 개선되지만 이때의 캐시 항목수는 8192개로서 캐시를 검사하는 데에 많은 시간을 소모하므로 실제 적용하기에는 적절하지 않을 것이다. TPC는 캐시의 크기가 64K 바이트로 늘어나면서 응답시간의 개선효과가 비교적 크게 나타났다.

IV. Conclusions

페이지 매핑 기반의 FTL에서는 주소변환 정보가 너무 커서 전부 RAM에 관리할 수가 없고 플래시 메모리에 기록할 수밖에 없다. 대신에 자주 사용하는 논리적 페이지들에 대해서는 RAM에 캐시 형태로 관리함으로써 주소변환 정보를 관리하는 오버헤드를 줄일 수 있다. 주소변환 캐시의 성공률은 SSD의 응답성능에 핵심적인 요소이므로 이를 높이는 것이 매우 중요하다.

일반적으로 프로세스들이 페이지들을 요청할 때에 시간적 지역성과 공간적 지역성의 특성을 보여주므로 캐시를 관리할 때 이 특성들을 충분히 활용해야 한다. 그러나 기존의 페이지 매핑 기반의 FTL들은 단순히 시간적 지역성만을 활용하거나, 공간적 지역성을 활용한다고 해도 단순히 한 페이지 크기인 TP 단위로 고정된 크기를 적용하였다. 본 논문의 TSL FTL에서는 TP을 일정한 개수로 나눈 TPS 단위로 캐시를 관리하며, TPS의 크기는 여러 가지 트레이스들에서 나타나는 시간적 지역성과 공간적 지역성에 대한 특성을 동시에 반영하여 결정하였다.

트레이스별로 응답시간을 평가한 결과 모든 트레이스에서 TSL FTL이 가장 우수한 성능을 보여 주었다. 특히 캐시의 크기가 4K 바이트 정도로 매우 작아도 가장 우수한 성능을 보여주었으며 최적의 비교대상인 순수 페이지 매핑에 가장 근접한 성능을 보여주었다. 실제 SSD에 적용할 때에는 캐시의 항목 수가 많으면 캐시를 검사하는 데에 소요되는 시간도 중요하다. TSL FTL은 캐시의 항목수가 256개 이내로도 충분하므로 캐시 검사를 위한 오버헤드도 문제가 되지 않을 정도로 작다. 앞으로 가비지 컬렉션과 웨어 레벨링 기능을 위한 연구를 보완하면 TSL FTL이 실제 SSD에 적용될 수 있을 것이다.

REFERENCES

[1] A. Gupta, Y. Kim, and B. Urgaonkar, “DFTL: A Flash Translation Layer Employing Demand-Based Selective Caching of Page-Level Address Mappings,” in Proc. 14th Int. Conf. Archit. Support Program. Languages Operating Syst., pp. 229-240, 2009.

[2] J. Kim, J. M. Kim, S. H. Hoh, S. L. Min, and Y. Cho, “A Space Efficient Flash Translation Layer for CompactFlash System,” IEEE Trans. Consum. Electron., vol. 48, no. 2, pp. 366-375, May 2002.

[3] S.-W. Lee, D.-J. Park, et al., “A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation,” ACM Trans. Emb. Comput. Syst., vol. 6, no. 3, pp. 1-27, Jul. 2007.

[4] Y. Guan, et. al., “A Block-Level Log-Block Management Scheme for MLC NAND Flash Memory Storage Systems,” IEEE Trans. on Computers, vol. 66, no. 9, Sep. 2017.

[5] Y. Zhou et al., "An Efficient Page-level FTL to Optimize Address Translation in Flash Memory," Proc. 10th European Conference on Computer Systems, Article bo. 12, Bordeaux France, April, 2015
[6] Z. W. Qin, Y. Wang, D. Liu, and Z. Shao. "A Two-Level Caching Mechanism for Demand-Based Page-Level Address Mapping in NAND Flash Memory Storage Systems," In Proc. of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 157-166, 2011.
[7] S. Jiang, L. Zhang, X. Yuan, H. Hu, and Y. Chen. "S-FTL: An Efficient Address Translation for Flash Memory by Exploiting Spatial Locality," In Proc. of Mass Storage Systems and Technologies (MSST), 2011.
[8] Hwan-Pil Choi, Yong-Seok Kim, "An Efficient Cache Management Scheme of Flash Translation Layer for Large Size Flash Memory Drives," Journal of The Korea Society of Computer and Information, vol. 20, no. 11, pp. 31-38, November 2015
[9] Hwan-Pil Choi, Yong-Seok Kim, "An Efficient Cache Structure for Demand-Based Flash Translation Layer," Journal of The Korea Society of Computer and Information, vol. 22, no. 7, pp. 1-7, July 2017
[10] "Storage Traces of UMass Trace Repository," http://traces.cs.umass.edu/ index.php/Storage/Storage
[11] "Samsung 1G x 8 Bit - 2G x 8 Bit- 4G x 8 Bit NAND Flash Memory Datasheet (K9XXG08UXA)," https://www.scribd.com/document/7010323/Samsung-1G-x-8-Bit-2G-x-8-Bit-4G-x-8-Bit-NAND-Flash-Memory-Datasheet

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.