JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 26 No. 2, pp. 11-18, February 2021
https://doi.org/10.9708/jksci.2021.26.02.011

A Multi-Level Flash Translation Layer for Large Capacity Solid State Drives

Yong-Seok Kim*

*Professor, Dept. of Computer Engineering, Kangwon National University, Gangwon-do, Korea

[Abstract]

The flash translation layer(FTL) of SSD maps the logical page number requested from the host to the actual recorded flash memory page number. It is very important to reduce the amount of RAM used to manage the mapping information. In the existing demand-based FTLs, two-level method is applied in which mapping information is also recorded in flash memory pages and only their addresses are managed as a table in RAM. As the capacities of SSDs are growing to tens of terabytes, the amount of RAM for mapping table becomes too large. In this paper, ML-FTL was proposed as a method of managing mapping information in three levels to reduce the amount of RAM required drastically. From an evaluation, the increase in overhead was minimal compared to the conventional two-level method by properly utilizing cache.

▸ **Key words**: FTL, SSD, Flash Memory, Page Mapping, Cache, LRU

[요 약]

SSD의 FTL에서는 호스트로부터 요청된 논리적 페이지 번호를 실제 기록된 플래시 메모리 페이지 번호로 매핑하는 작업을 한다. 매핑 정보를 관리하기 위해서 사용되는 RAM의 용량을 줄이는 것은 매우 중요하다. 기존의 요구기반 FTL에서는 매핑 정보도 플래시 메모리 페이지에 기록하고 그들의 주소만 RAM에 테이블로 관리하는 2단계 방법을 적용하였다. 그러나 SSD의 용량이 수십 테라바이트 수준으로 늘어나고 있으므로 이 방법만으로는 충분하지 않다. 본 논문에서는 소요되는 RAM의 용량을 획기적으로 줄이기 위해서 매핑 정보를 3단계로 관리하는 방법인 ML-FTL을 제안하고 그 성능을 평가하였다. 캐시를 적절히 활용함으로써 기존의 2단계 방법에 비해서 오버헤드가 늘어나는 정도가 미미하다는 것을 확인하였다.

▸ **주제어**: 플래시변환계층, SSD, 플래시 메모리, 페이지 매핑, 캐시, LRU

• First Author: Yong-Seok Kim, Corresponding Author: Yong-Seok Kim
*Yong-Seok Kim (yskim@kangwon.ac.kr), Dept. of Computer Engineering, Kangwon National University
• Received: 2020. 11. 16, Revised: 2021. 01. 22, Accepted: 2021. 01. 23.

Copyright © 2021 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945
www.kci.go.kr

I. Introduction

플래시 메모리 기반의 SSD (Solid State Drive)는 저장 장치의 주류로서 HDD (Hard Disk Drive)를 급속히 대체하고 있다. 그 배경에는 HDD와 같은 기계적 작동 부분이 없고 순수하게 반도체만으로 구성된 특성 때문에, 빠르고, 가볍고, 전력 소모량도 작고, 충격에도 강하다는 특성이 있다. HDD에 비해서 유일한 단점이었던 가격 문제도 이미 극복되어서, 휴대형 기기는 물론이고 개인용 컴퓨터 뿐만 아니라 대규모 서버에도 핵심 저장장치로 자리매김하고 있다.

SSD의 저장 용량도 급속히 증가해서 이제 소형 저장장치도 테라바이트 급의 용량이 보편화 되어 있다. 이러한 발전은 회로 미세화 기술과 트랜지스터 1개당 저장하는 비트 수의 증가에 의한 것이다. 트랜지스터 당 1비트의 정보를 저장하는 기술에서 시작하여 지금은 트랜지스터 당 3비트를 저장하는 TLC(Triple-Level Cell)를 넘어서 4비트를 저장하는 QLC(Quad-Level Cell) 기술까지 상용화되어 있다. 이러한 기술의 발전에 힘입어서 단위 용량당 가격까지도 HDD와 경쟁하는 수준까지 도달한 것이다.

SSD는 여러 개의 플래시 메모리 칩에 제어기를 더해서 구성된다. SSD도 HDD처럼 일정한 크기의 페이지 단위로 읽기와 쓰기를 한다. 플래시 메모리는 블록들의 집합으로 구성되며 하나의 블록은 페이지들의 집합으로 구성된다. 한 페이지의 읽기는 특정 페이지를 지정해서 읽어낼 수 있지만 한 페이지의 쓰기는 과정이 조금 복잡하다. 이미 데이터가 기록된 페이지에 새로운 데이터를 쓰려면 HDD는 바로 덮어서 쓰기가 가능하지만 플래시 메모리는 이것이 허용되지 않는다. 깨끗이 지워진 페이지에만 쓰기가 가능하다. 항상 새로운 페이지에 쓰기를 해야 하고, 따라서 쓰기 요청을 한 논리적 페이지 번호(LPN: Logical Page Number)를 실제로 어디에 기록되어 있는지에 대한 물리적 페이지 번호(PPN: Physical Page Number)로 매핑하기 위한 정보를 SSD 내부에 관리해야 한다. 이러한 작업 부분이 플래시 변환 계층(FTL: Flash Translation Layer)인데, SSD에 내장된 제어기에서 처리한다.

일반적으로 FTL에서는 LPN을 PPN으로 매핑하는 작업뿐만 아니라 가비지 컬렉션(Garbage Collection)과 웨어 레벨링(Wear Leveling)도 처리한다. 이미 데이터가 기록된 페이지에 새로운 데이터를 기록하려면 먼저 지우기 작업을 해야 하는데, 데이터의 지우기는 NAND 플래시 메모리의 특성상 블록 단위로만 허용된다. 일부 페이지들만 유효한 블록들은 묶어서 새로운 블록에 유효한 페이지들만 골라서 옮겨서 기록하고 기존의 블록들을 지우기를 하는데, 이러한 작업이 가비지 컬렉션이다. 각 페이지들은 지우고 다시 기록할 수 있는 횟수에 한계가 있다. 지우고 쓰기를 반복하면 트랜지스터의 물성이 변하기 때문이다. 따라서 일부 페이지들에 집중적으로 지우고 쓰기가 반복되지 않도록 모든 페이지들에 적절히 분산시켜야 하는데 이러한 처리가 웨어 레벨링이다.

본 논문에서는 대용량 SSD를 위한 FTL에 대한 것으로서 LPN을 PPN으로 매핑하는 방법에 대한 문제를 다룬다. 모든 LPN들에 대하여 매핑 정보를 SSD 내부의 RAM에 기록해 둘수 있다면 문제는 간단하지만, 비현실적으로 큰 용량의 RAM을 필요로 한다. 그 해결 방안으로서 요구기반의 FTL에서는 모든 매핑 정보를 플래시 메모리 페이지에 기록해 두고 자주 사용되는 일부분만 캐시에 관리하는 방법을 적용한다[1]. 여기에 다양한 개선 방법들이 제안되었는데, 대부분 캐시 메모리의 크기를 줄이면서도 캐시의 성공률을 높이는 것에 초점을 맞추고 있다. RAM에는 캐시뿐만 아니라 매핑 체계의 최상단 정보도 기록하여야 한다. 본 논문에서는 캐시 용량을 줄이는 것과는 별개로 RAM에 기록할 최상단 매핑 정보를 획기적으로 줄임으로써 RAM의 용량을 줄이는 방법을 다룬다. 특히 수십 테라바이트 이상의 대용량 SSD가 보편화되고 있는 상황에서 RAM의 용량을 현실적으로 적용이 가능한 수준으로 대폭 줄일 수 있게 한다.

II. Related Works

호스트로부터 요청이 온 LPN을 FTL에서 PPN으로 매핑하는 방법으로는 기본적으로 페이지 매핑과 블록 매핑이 있고, 이들을 혼합한 방법이 있다. 페이지 매핑은 SSD 내의 모든 페이지들에 일정한 규칙에 따라서 일련의 페이지 번호인 PPN을 매기고, LPN마다 그에 대응되는 PPN을 매핑 테이블에 관리한다. 이것은 매우 간단하지만 매핑 테이블이 지나치게 커서 SSD 내부의 RAM에 관리할 수 없게 된다[1]. 블록 매핑은 블록 단위로만 매핑 정보를 관리하고 블록 내에서 페이지별 오프셋은 LPN과 PPN이 동일하게 유지한다. 이렇게 하면 매핑 테이블의 크기를 줄일 수 있지만 한 페이지를 새로운 데이터로 기록하려면 제자리에 기록할 수가 없으므로 블록내의 나머지 유효한 페이지들까지도 새로운 블록에 옮겨서 기록해야하는 엄청난 오버헤드가 수반된다. 이러한 문제를 해결하기 위해서 혼합 방법에서는 일부 블록들을 로그 블록으로 설정하고 페이지 쓰기는 이 블록들을 이용한다[2,3]. 일정한 시점에 가

www.kci.go.kr

서 이렇게 기록된 페이지들을 모아서 정상 블록들에 정리하여 기록하는 과정을 거친다.

페이지 매핑을 적용하면서도 RAM 용량을 줄이는 방법으로 제안된 것이 요구기반의 FTL인 DFTL이다[1]. 모든 매핑 정보는 기본적으로 플래시 메모리에 기록한다. 대신에 자주 사용되는 매핑 정보만 RAM에 캐시 형태로 관리한다. LPN을 PPN으로 매핑할 때마다 플래시 메모리에서 매핑 정보를 기록한 변환 페이지(TP: Translation Page)를 읽어 와야 하므로 오버헤드가 매우 크지만, 캐시에 관리되는 정보에서 대부분 처리되므로 오버헤드는 낮아진다. 예를 들어서 20TB (테라 바이트) 용량의 SSD에서 한 페이지당 2KB의 크기라고 하면 매핑 정보가 총 20TB/2KB = 10G 개에 달한다. 한 항목 당 4바이트(32비트 정수)로 표현한다면 총 10Gx4B = 40GB의 용량이 필요하지만 이들을 모두 플래시 메모리에 기록하고 일부만 RAM에 캐시로 관리한다. 캐시의 성공률이 높기 때문에 TP를 읽거나 쓰는 데 소요되는 오버헤드는 합리적인 수준으로 제한된다. 캐시의 성공률이 높은 이유는 호스트 쪽에서 요청하는 LPN들에 대하여 시간적 지역성이 있기 때문이다.

DFTL을 개선하여 캐시를 위한 RAM의 사용량을 줄이기 위해서 HP-FTL에서는 해시를 활용하여 캐시를 압축하여 표현하는 방법을 제안하였다[4]. DFTL이 시간적 지역성에 기반을 두고 있는데 반해서 공간적 지역성 까지 고려한 방법들도 여러 가지가 제안되었다[5,6]. 그 중에서 TPC-FTL은 캐시를 TP 단위로 관리함으로써 공간적 지역성을 활용하여 캐시의 성공률을 높이면서도 캐시를 검사하는 시간을 대폭 줄일 수 있음을 보여주었다[6]. DFTL은 캐시에 <LPN,PPN> 쌍으로 기록하는데 하나의 쌍을 위해서 8바이트 (LPN 4 바이트와 PPN 4 바이트)가 필요하다. 만약 32KB 용량의 캐시를 사용한다면 32KB/8B = 4096개의 항목이 있다. 이 캐시를 일반 RAM으로 구성한다면 매핑 때마다 4096개를 검사해야 하는 것이다. 이에 비해서 TPC-FTL은 32KB 메모리에 16개 페이지의 TP들만 관리하므로 매핑 때마다 16개만 검사하면 충분하다. 그러면서도 공간적 지역성으로 인해서 동일한 TP가 반복적으로 사용되어서 캐시 성공률이 매우 높게 나온다. SHRD는 쓰기 주소가 순차적이지 않은 요청을 FTL 내부에서 적절하게 변형하여 공간적 지역성을 높이도록 제안한 것인데 추가적인 매핑 단계를 필요로 하므로 구조가 매우 복잡하다[7]. AALRU는 호스트의 워크로드가 변함에 따라 캐시를 관리하는 방법도 적응적으로 변화시키는 방법을 제안하였다[8]. 이러한 방안들의 목표는 가능한한 RAM의 소요량을 줄이면서도 매핑 과정의 오버헤드를 줄이는 것이다. 호스트에서 요청한 데이터 페이지의 읽기/쓰기 작업에 소요되는 시간 이외에 매핑 정보를 플래시 메모리 페이지에 기록하거나 읽어오는 추가적인 작업에 소요되는 시간은 모두 오버헤드에 해당한다.

DFTL과 이것에 근거한 모든 FTL들은 그림 1과 같이 TP가 기록된 페이지들의 주소(TPN: TP Number)를 RAM에 GDT(Global Directory Table)로 관리하므로 LPN을 PPN으로 매핑하는 정보는 GDT와 TP를 거치는 2단계로 구성된다. 그런데 SSD의 용량이 커짐에 따라서 GDT를 위한 RAM의 용량도 매우 커지게 된다. 한 페이지의 크기가 2KB라고 가정하고 PPN을 4 바이트로 기록하면 TP 당 512개의 PPN들을 기록할 수 있다. SSD의 용량이 32TB라면 32TB/2KB = 16G개의 매핑 항목이 필요하고, 16G/512 = 32M 개의 TP가 사용된다. 따라서 GDT를 저장하기 위해서 32M x 4B = 128MB의 RAM이 필요하다. 본 논문에서는 대용량 SSD에서 GDT를 위한 RAM 용량을 획기적으로 줄이는 방법으로서 DFTL의 2단계를 확장한 3단계 매핑 테이블을 제안하고 그 유용성을 검증한다. 3단계로 확장하면 GDT를 위한 RAM의 용량을 1/512로 대폭 줄일 수 있고, 단계가 늘어남으로 인한 추가 오버헤드는 캐시를 적절히 활용함으로써 매우 작은 수준으로 제한할 수 있음을 확인하였다.

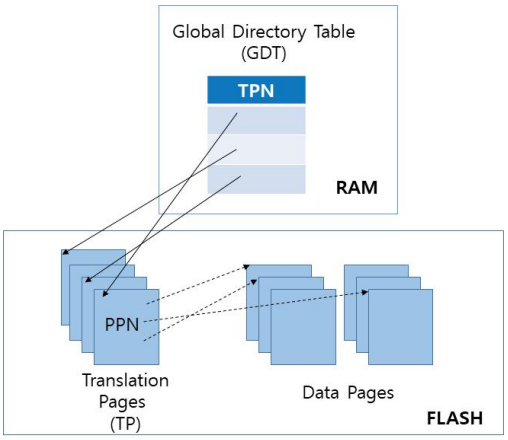

Fig. 1. 2-Level Mapping Structure of DFTL

III. Proposed Multi-Level FTL

1. Basic Structure

본 논문에서 제안하는 ML-FTL(Muti-Level FTL)은 기본적으로 그림 2와 같이 3단계로 구성된다. TP들의 페이지 주소인 TPN을 플래시 메모리에 DP(Directory Page)로 기록하고 GDT에는 DP들의 페이지 주소(DPN: DP

www.kci.go.kr

Number)를 기록한다. 이렇게 함으로써 RAM에 관리하는 GDT의 크기를 대폭 줄일 수 있고, 따라서 대용량 SSD에서 소요되는 RAM의 용량을 합리적인 수준으로 줄일 수 있게 된다. RAM의 소요량은 하나의 DP에 기록되는 TPN들의 개수 비율로 줄어든다.

호스트에서 한 페이지의 읽기/쓰기 요청이 오면 그 LPN을 PPN으로 매핑하는 과정은 GDT에서 DPN을 확인하고 그 DP를 읽어서 TPN을 확인하고 그 TP를 읽어서 PPN을 결정한다. DFTL의 2단계 매핑을 3단계로 확장한 방법은 필요에 따라서 4단계 이상으로 확장하는 데에도 그대로 적용될 수 있다. 이렇게 3단 구조이면 2단 구조에 비해서 DP 읽기/쓰기의 오버헤드가 추가된다. TP를 읽기 위해서는 먼저 DP를 읽어서 TPN을 확인해야 하고, TP를 기록하면 변경된 TPN을 DP에도 반영해야 하므로 DP도 변경된다. 이러한 오버헤드는 TP를 캐시에 관리하듯이 DP도 캐시에 관리함으로써 해결할 수 있다.

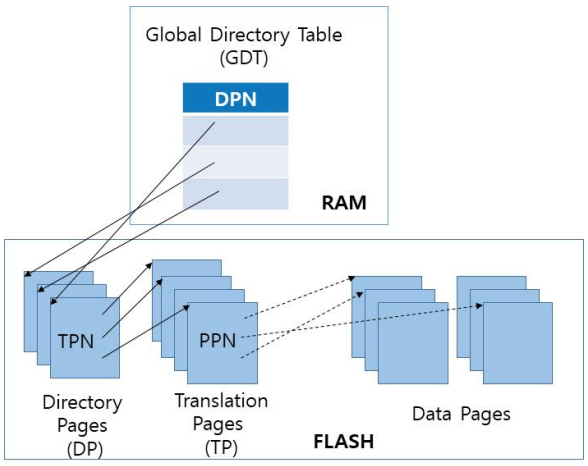

Fig. 2. 3-Level Mapping Structure of ML-FTL

호스트에서 요청한 LPN으로부터 플래시 메모리에 기록된 페이지인 PPN을 결정하는 과정은 그림 3과 같다. LPN과 PPN은 32비트(4바이트)라고 가정한다. 3단계로 구성되므로 LPN은 GDT 인덱스, DP 인덱스, 그리고 TP 인덱스의 3부분으로 구분한다. 한 페이지의 크기가 2KB 이고 그래서 한 페이지당 512개의 페이지 번호를 기록한다면 TP 인덱스와 DP 인덱스는 각각 9비트이고 나머지 상위 14비트가 GDT 인덱스이다. 먼저 LPN의 GDT 인덱스 부분을 활용하여 GDT에서 DP가 기록된 페이지 번호 DPN을 확인하고 DP를 읽어온다. 이 DP에서 LPN의 DP 인덱스 부분을 활용하여 TP가 기록된 페이지 번호 TPN을 확인하고 TP를 읽어온다. 이 TP에서 LPN의 TP 인덱스 부분을 활용하여 데이터가 기록된 페이지 번호인 PPN을 확인한다.

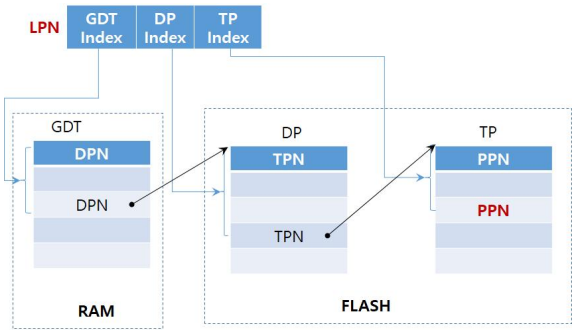

Fig. 3. Determining PPN from LPN

DP와 TP를 읽어오는 오버헤드를 줄이기 위해서 자주 사용되는 것들을 캐시에 관리하는데 기본 구조는 그림 4와 같이 3가지의 캐시들로 구성된다. TP를 위한 캐시는 기본적으로 TPC-FTL과 동일하게 최근에 사용된 TP들을 보관한다. 특정 TP를 구별하기 위해서는 LPN에서 TP 인덱스 부분을 제외한 값을 사용하면 되므로, 이 값을 캐시에서 특정 TP를 구별하기 위한 Base 값으로 사용한다. TP 인덱스 부분은 특정 TP 내에서 몇 번째 항목인지를 구별하는 용도이기 때문이다.

DP를 위한 캐시도 최근에 사용된 DP들을 보관한다. 특정 DP를 구별하기 위해서는 LPN에서 DP 인덱스 이후 부분을 제외한 값을 사용하면 된다. 이것은 곧 GDT 인덱스 부분만을 의미하며, DP 캐시에서 특정 DP를 구별하기 위한 Base 값으로 사용된다.

하나의 DP는 512개의 TP들을 커버하므로 매우 큰 영역에 걸쳐있다. 따라서 DP 캐시와 별도로 최근에 사용된 TPN들을 TPN 캐시에 보관함으로써 DP 캐시를 거치지 않고도 TPN을 확인할 수 있도록 한다. TPN 캐시의 각 항목은 TP를 구분하는 Base와 플래시 메모리에 기록된 주소인 TPN의 쌍으로 관리한다. TP 캐시의 각 항목에는 Base 값 외에 TPN도 함께 기록하며, 이 정보를 TPN 캐시에 활용한다.

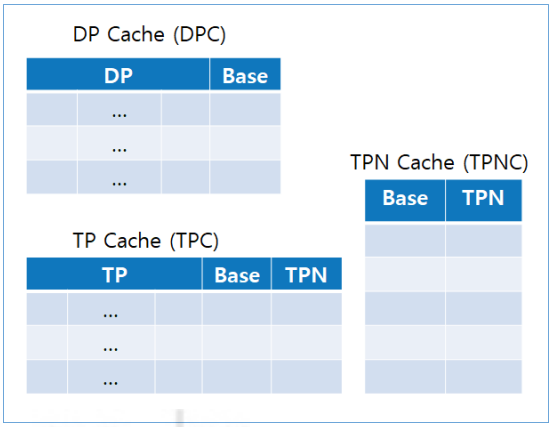

Fig. 4. DP Cache, TP cache and TPN Cache

www.kci.go.kr

2. Page Number Mapping Algorithm

캐시는 LRU(Least Recently Used) 정책에 따라서 최근에 사용된 것들을 보관한다. DP 캐시와 TP 캐시에 새로운 항목을 추가하기 위해서는 희생 대상 항목을 선택해야 하는데, 플래시 페이지에서 캐시로 읽어 온 이후에 수정되지 않은 항목들 중에서 사용한지가 가장 오래된 것을 선택한다. 수정되지 않은 항목이 없다면 수정된 항목들 중에서 가장 오래된 것을 선택하여 그 내용을 플래시 페이지에 기록한 후에 사용한다. TPN 캐시에 새로운 항목을 추가할 때에는 사용한지가 가장 오래된 항목을 선택한다.

호스트에서 LPN을 지정하여 읽기/쓰기 요청이 오면 PPN을 결정하는 과정은 그림 5의 알고리즘과 같다. 먼저 LPN으로부터 얻은 TP의 Base 값을 활용하여 TP 캐시를 검사한다. 해당 TP가 캐시에 없으면 플래시 페이지에서 읽어서 캐시에 추가한다. 이 때 TPN 캐시에도 이 항목을 반영한다. 읽기 요청에 대해서는 해당 TP로부터 TP 인덱스를 활용하여 PPN을 확인하여 사용한다. 쓰기 요청에 대해서는 새로운 페이지를 할당하여 PPN으로 결정하고 TP에 반영한다.

TP 캐시에 필요한 항목이 없으면 TP를 읽어서 캐시에 추가하는데, 그 과정은 그림 6의 알고리즘과 같다. TP가 기록된 페이지 번호 TPN을 결정하기 위해서 먼저 TPN 캐시에서 TPN을 검사한다. 여기에 없으면 DP 캐시를 검사하는데, LPN으로부터 얻은 DP의 Base 값을 활용한다. 여기에도 없으면 DP를 플래시 페이지에서 읽어서 DP 캐시에 추가한다. DP를 확보하면 DP 인덱스를 활용하여 TPN을 결정한다. TPN이 결정되면 TP 읽기를 실행하는데, TP 캐시에 빈 공간이 없으면 캐시에 등록된 이후에 수정되지 않은 TP들 중에서 사용한지가 가장 오래된 것을 선택하여 이 공간을 사용한다. 수정되지 않은 것이 없다면 사용한지가 가장 오래된 것을 선택하여 플래시 페이지에 기록하고 그 TPN을 DP의 해당 항목에 갱신한다.

```
on Read/Write Request of LPNi:
if the TPj covering LPNi is found on TPC
  select the TPC slot;
else
  load the TPj on a victim slot of TPC;
if the request is Read
  determine PPNi corresponding to the LPNi
       based on TPj;
else       // the request is Write
   allocate a free flash page PPNi;
   update the corresponding LPNi entry
       of TPj as PPNi;
   set the TPC slot as dirty;
update access time of the TPC slot and
       the corresponding TPNC entry;
return PPNi;
```

Fig. 5. Algorithm of Determining PPN from LPN

```
load TPj on a victim slot of TPC:
if TPj is found on TPNC
  get TPN of TPj from TPNC;
else if the DPk covering TPj is found on DPC
  get TPN of TPj from the DPk;
else
  load the DPk on a victim slot of DPC;
  get TPN of TPj from the DPk;
if (TPC has empty slots)
  select any empty slot as victim;
else
  if (TPC has clean slots)
    select the oldest clean slot as victim;
  else
    select the oldest dirty slot as victim;
    save the TP of the victim slot to a free page;
    update the corresponding entry of the
        DP covering the TP of victim slot;
load the TPj on the victim TPC slot from
      flash page of TPN;
set the TPC slot as clean;
```

Fig. 6. Algorithm of TP Cache Loading

DP 캐시에 필요한 항목이 없으면 DP를 읽어서 캐시에 추가하는데, 그 과정은 그림 7의 알고리즘과 같다. DP가 기록된 페이지 번호 DPN을 GDT로부터 확인하고 그 페이지를 읽어 들인다. 이때 DP 캐시에 빈 항목이 없으면 TP 캐시와 같은 방법으로 희생대상 항목을 결정한다. 이 항목이 수정된 것이었다면 먼저 플래시 페이지에 기록하고 그 DPN을 GDT의 해당 항목에 갱신한다.

```
load DPk on a victim slot of DPC:
if (DPC has empty slots)
  select any empty slot as victim;
else
  if (DPC has clean slots)
    select the oldest clean slot as victim;
  else
    select the oldest dirty slot as victim;
    save the DP of the victim slot to a free page;
    update the corresponding entry of GDT;
get DPN of DPk from GDT;
load DPk on the victim DPC slot from DPN page;
set the DPC slot as clean;
```

Fig. 7. Algorithm of DP Cache Loading

IV. Performance Evaluation

FTL에서는 호스트에서 요청한 데이터 페이지를 읽고 쓰는 작업 이외에 TP와 DP를 읽고 쓰는 오버헤드가 수반된다. ML-FTL에서 발생하는 오버헤드에 대한 성능 평가를 하기 위해서 실제 시스템에서 수집한 트레이스 정보를 활용하였다. UMass 트레이스는 이러한 목적으로 많이 사

www.kci.go.kr

용되는 것인데 그 중에서 Financial 1과 WebSearch 1을 사용하였다[9]. 표 1과 같이 Financial 1은 온라인 트랜잭션 처리 서버에서 수집한 것으로서 읽기와 쓰기가 적절히 섞여있지만 쓰기 요청이 훨씬 많다. WebSearch 1은 웹 서버에서 수집한 것으로서 읽기가 압도적으로 많이 이루어지며, 평균 요청 간격이 Financial 1보다 훨씬 짧다. UMass 트레이스에 함께 포함된 Financial 2는 SSD의 크기가 고작 0.6GB로 작아서 제외하였고, WebSearch 2와 3은 특성이 WebSearch 1과 거의 유사하므로 생략하였다. UMass 트레이스에서는 한번에 읽기/쓰기를 요청하는 크기가 다양한데, 한번의 요청에 대하여 여러 페이지에 걸쳐서 읽기/쓰기를 해야 한다. Financial 1에서 읽기 요청은 평균 2.3페이지, 쓰기 요청은 평균 3.7 페이지에 걸쳐서 이루어진다. WebSearch 1의 경우에는 요청 단위가 더욱 커서, 읽기 요청은 평균 15.1 페이지, 쓰기 요청은 8.6 페이지에 걸쳐서 이루어진다.

Table 1. Characteristics of UMass Traces

Trace	Write Request Ratio	Average Inter-arrival Time	SSD Capacity
Financial 1	76.8%	8.2ms	644GB
WebSearch 1	0.02%	3.0ms	17GB

일반적으로 플래시 메모리의 한 페이지의 크기는 보통 2KB이거나 그 이상이다. Financial 1은 650GB 크기의 SSD를 반영하고 있어서 대용량 SSD에 대한 조건을 어느 정도 만족한다. 그러나 WebSearch 1의 경우에는 17GB 정도에 그친다. 이 트레이스들을 활용하면서도 대규모 SSD에 대한 ML-FTL의 특성을 정확하게 보여주기 위해서 페이지의 크기에 1KB를 적용하여 시뮬레이션을 진행하였다. 페이지의 크기가 2KB이면 TP 하나에 PPN을 512개, DP 하나에 TPN을 512개 기록할 수 있고, 그러면 GDT의 최대 인덱스가 Financial 1은 1300 정도밖에 되지 않고, WebSearch 1의 경우에는 고작 33 정도로 작다. 페이지의 크기에 1KB를 적용하면 한 페이지 당 기록할 수 있는 항목의 수가 TP와 DP가 모두 절반으로 줄어들고 페이지의 크기도 절반이 되므로, GDT의 최대 인덱스는 8배로 늘어난다. 그러면 WebSearch 1의 경우에는 260 정도로 늘어나고, Financial 1의 경우에는 10,200까지 늘어나서 DP 캐시에 대한 효과를 보다 정확하게 검증할 수 있게 된다. 시뮬레이션에 적용한 플래시 메모리의 파라미터로는 가장 보편적인 것으로서 한 페이지 읽기에 25us, 한 페이지 쓰기에 200us를 적용하고, 연속적인 요청으로 인한 큐잉 지연도 반영하였다[10].

TPC-FTL은 DFTL에 비해서 캐시를 위한 RAM의 크기를 동일하게 적용하더라도 캐시 검사 시간을 단축하면서도 캐시의 성공률을 높일 수 있다고 하였다[6]. ML-FTL의 성능을 평가하기 위해서 TPC-FTL과 비교하였다. 한 페이지의 크기가 1KB이므로 DP 하나에 256개의 TPN들이 기록되고, 따라서 GDT의 크기는 기존의 DFTL이나 TPC-FTL에 비해서 1/256로 대폭 줄어든다. 비교 기준은 한 번의 읽기/쓰기 요청에 대하여 처리 완료에 소요되는 오버헤드이다. RAM의 소요량을 대폭 줄이면서도 오버헤드가 별로 늘어나지 말아야 하는 것이다.

TP 캐시는 8개를 적용하였다. ML-FTL은 DP 캐시를 8개 추가하고 TPN 캐시를 위해서 1KB의 RAM을 추가한 경우에 대한 평가 결과이다. 표 2와같이 ML-FTL의 오버헤드는 TPC-FTL에 비해서 늘어나기는 하지만 읽기/쓰기를 처리하는 전체 소요 시간에 비하면 아주 미미하다. 늘어나는 정도가 Financial 1의 경우에는 읽기 요청에 1.00%, 쓰기 요청에 0.53% 늘어나고, WebSearch 1의 경우에는 읽기 요청에 2.51%, 쓰기 요청에 0.02% 정도 늘어나는 데에 그친다. 이렇게 ML-FTL은 GDT를 위한 RAM의 크기를 TPC-FTL에서 비해서 1/256로 획기적으로 줄이면서도 오버헤드 증가는 매우 미미한 것을 확인하였다.

Table 2. Comparison of Overhead

Trace	Request	(a) Average Service Time (us)	Average Overhead (us)		$\frac{b-c}{a}$
			(b) ML-FTL	(c) TPC-FTL	
Financial 1	Read	713	94.9	87.7	1.00%
	Write	2,154	112.3	100.9	0.53%
WebSearch 1	Read	575	34.7	20.2	2.51%
	Write	2,499	1.1	0.6	0.02%

적절한 DP 캐시의 개수를 확인하기 위해서 TP 캐시의 개수를 8개로 고정하고 DP 캐시 개수에 따른 오버헤드 추이를 확인하였다. 그림 8와 같이 Financial 1의 경우에 8개 까지는 오버헤드가 줄어들지만 그 이상은 효과가 별로 없다. WebSearch 1은 한 번에 읽기/쓰기를 요청하는 크기가 매우 크기 때문에 공간적 지역성 효과가 커서 오버헤드가 Financial 1보다 훨씬 작게 나오는데, DP캐시 개수가 늘어나도 개선의 효과가 없지는 않지만 매우 미미하다. 따라서 DP 캐시의 개수는 8개 정도만 사용해도 충분한 것으로 나타났다. 물론 SSD의 용량, 페이지의 크기, 기타 플래시 메모리의 특성 값에 따라서 적절한 캐시의 개수에 차이가 있을 것이다.

www.kci.go.kr

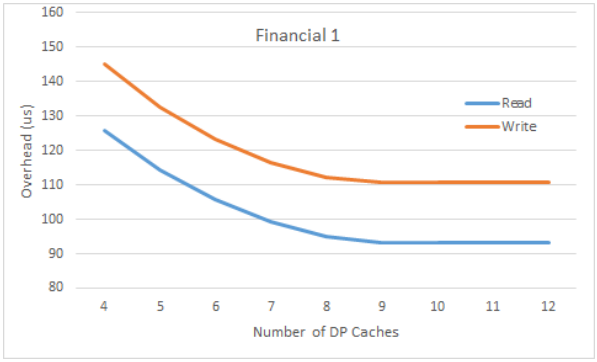

Fig. 8. Overhead According to the Number of DP Caches

캐시를 위한 RAM의 용량이 정해져 있을 때 DP 캐시와 TP 캐시의 적절한 배분을 확인하기 위해서, TP와 DP를 위한 전체 캐시의 수를 16개로 고정하고 TP를 위한 캐시와 DP를 위한 캐시의 배분에 따른 오버헤드를 확인하였다. Financial 1은 그림 9과 같이 오버헤드가 가장 적은 경우는 DP를 위해서 7~10개, 나머지를 TP를 위해서 배분하는 경우이다. WebSearch 1의 경우에는 DP 캐시가 3개 이상이면 큰 차이가 없다. 따라서 DP를 위해서 8개 정도를 배분하고 나머지는 모두 TP를 위해서 사용하는 것이 좋은 성과를 낼 수 있는 것으로 나타났다.

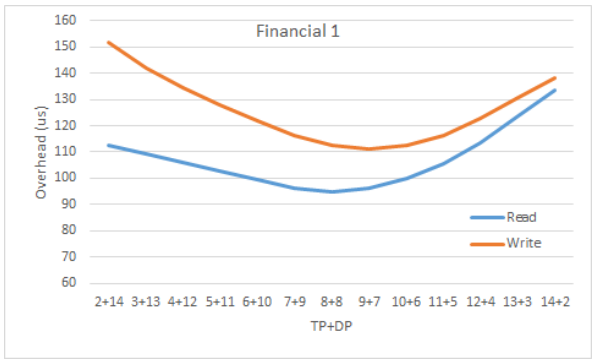

Fig. 9. Dividing Cache RAM for TP and DP

V. Conclusions

SSD 내부에는 호스트에서 요청한 페이지 번호인 LPN을 실제로 데이터가 기록된 플래시 메모리 페이지 번호인 PPN으로 매핑하는 부분인 FTL이 있다. 순수 페이지 매핑 방법으로 FTL을 구현하려면 LPN마다 대응되는 PPN을 RAM에 테이블 형태로 기록해야 하므로 소요되는 RAM의 용량이 매우 방대하다. 기존의 요구기반 FTL에서는 RAM의 용량을 줄이기 위해서 매핑 정보도 플래시 메모리 페이지에 TP로 기록하고 그 TP들의 주소만 RAM에 테이블로 관리하는 2단계 방법을 적용하였다. 그러나 SSD의 용량이 테라바이트 수준이 보편화되고 있으므로 TP들의 주소를 기록하는 RAM의 용량도 지나치게 커진다.

본 논문에서는 소요되는 RAM의 용량을 획기적으로 줄이기 위해서 매핑 정보를 3단계로 관리하는 방법인 ML-FTL을 제안하고 그 성능을 평가하였다. 또한 3단계로 확장하는데 따른 오버헤드를 줄이기 위해서 캐시를 관리하는 방법도 제안하였다. 매핑 정보는 제일 상단의 GDT, 중간단의 DP, 그리고 하단의 TP로 구성된다. 캐시는 DP 캐시와 TP 캐시를 사용하고, 이와 별도로 최근에 사용한 TP들에 대하여 플래시 페이지의 위치 정보를 캐시로 관리한다. UMass 트레이스를 적용하여 평가해본 결과 ML-FTL은 기존의 요구기반의 FTL들에 비해서 RAM 소요량은 대폭 줄이면서도 오버헤드는 미미하게 늘어나는 것을 확인하였다. 캐시를 적용할 때 중간 단에 해당하는 DP 캐시에는 8개 정도만 사용하고 나머지는 모두 제일 하단에 해당하는 TP 캐시에 사용하는 것이 좋은 것도 확인하였다.

수십 테라바이트 용량의 SSD가 보편화되고 있으므로 이러한 방법으로 RAM의 소요량을 줄이는 것은 매우 중요하다. RAM 소요량이 줄어드는 핵심 이유는 매핑 정보의 최상단에 해당하는 GDT의 크기가 줄어드는 것이다. 플래시 메모리의 한 페이지 크기가 2KB나 그 이상으로 커지면 GDT를 위한 RAM의 크기도 그 비율만큼 더 줄어든다. 또한 필요에 따라서는 이러한 확장 원리를 적용하여 4단계 이상으로 구성할 수도 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] S. Lee, et. al., “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.

[3] 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, pp. 1464-1477, Sep. 2017.

[4] F. Ni, et. al., "A Hash-Based Space-Efficient Page-Level FTL for Large-Capacity SSDs," 2017 International Conference on Networking, Architecture, and Storage (NAS), Shenzhen, 2017, pp. 1-6

[5] S. Jiang, et. al., “S-ftl: An efficient address translation for flash

www.kci.go.kr

memory by exploiting spatial locality," in Mass Storage Systems and Technologies (MSST), IEEE, 2011, pp. 1-12.

[6] H.-P. Choi, Y.-S. 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

[7] H. Kim, D. Shin, Y. Jeong, and K. Kim, "Shrd: Improving spatial locality in flash storage accesses by sequentializing in host and randomizing in device," in Proceedings of the 15th USENIX Conference on File and Storage Technologies, Berkeley, USENIX Association, 2017.

[8] Y. Yao, et. al., "An Advanced Adaptive Least Recently Used Buffer Management Algorithm for SSD," in IEEE Access, vol. 7, pp. 33494-33505, 2019,

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

[10] 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 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 flash memory file systems.

www.kci.go.kr