Journal of The Korea Society of Computer and Information
Vol. 24 No. 2, pp. 1-7, February 2019
www.ksci.re.kr
https://doi.org/10.9708/jksci.2019.24.02.001

A Compact Representation of Translation Pages for Flash Translation Layers of Solid State Drives

Yong-Seok Kim*

Abstract

This paper presents CTP (Compact Translation Page), a compact representation of translation pages, for page mapping-based flash translation layers to improve RAM utilization and reduce the response time of solid state drives. CTP can store translation information twice in a translation page and the total number of translation pages stored in flash is reduced to half. Therefore, CTP halves the RAM size of the directory of translation pages and uses the saved RAM space for translation cache. CTP shows the best response time when compared to existing page mapping-based flash translation layers.

▸Keyword: Flash translation layer, Solid state drive, Page mapping, NAND flash memory

I. Introduction

NAND 플래시 메모리 칩을 사용하는 SSD (Solid State Drive)는 휴대용 정보기기뿐만 아니라 일반 PC나 데이터 센터 서버까지 광범위하게 사용되고 있다. 기존의 하드디스크에 비해서 크기나 무게, 속도, 전력소모, 진동에 대한 내성 등의 여러 면에서 유리하고 이제는 용량대비 가격도 개선되어가고 있다. 단점으로는 기존의 데이터를 새로운 데이터로 다시 기록하기 위해서는 같은 영역에 그대로 기록할 수가 없으므로 지우기를 한 다음에 쓰기를 하거나 다른 영역에 기록해야 하는 점이다. 또한 지우고 쓰기를 반복하면 플래시 메모리의 물리적 특성이 변화되므로 최대 지우기 회수에 제한이 있어서 하드디스크에 비해서 수명이 짧다는 단점도 있다.

제자리 쓰기의 문제를 해결하기 위해서는 두 가지 방법이 있는데 하나는 기존의 하드디스크를 기반으로 설계된 파일시스템을 플래시 메모리의 특성에 맞게 완전히 다시 개발하는 것이다. 다른 하나는 기존의 파일시스템을 그대로 사용하되 SSD 내에서 데이터를 기록할 페이지 위치를 적절히 결정하고 그 위치 정보를 자체에서 관리하는 것이다. 즉 파일시스템에서 요청된 논리적 페이지 번호(LPN: Logical Page Number)를 SSD 내부에 실제로 기록된 물리적 페이지 번호(PPN: Physical Page Number)로 적절히 매핑하여 처리하도록 한다. 본 논문에서는 후자의 방법에 대하여 기존의 방법들을 개선한 효율적인 방안을 제시한다.

NAND 플래시 메모리의 읽기나 쓰기를 위한 기본 단위는 페이지이고, 지우기는 일정한 개수의 페이지들의 집합인 블록 단위로만 가능하다. 즉, SSD는 블록들의 집합으로 구성되며 하나의 블록은 일정한 개수의 페이지들로 구성된다. 파일시스템에서 요청하는 특정 논리적 페이지 (LPN) 쓰기 요청에 대하여 SSD는 내부의 빈 물리적 페이지 (PPN)을 선택하여 기록한 후에 그 매핑 정보를 등록한다. 특정 LPN의 내용을 새로운 데이터로 재기록하려면 새로운 페이지에 기록하고 이전의 페이지는 무효화한다. 특정 블록의 페이지들이 모두 무효화되면 이 블록은 지우기를 실시한 다음에 페이지들에 새로운 데이터를 기록할 수 있게 된다. SSD 내부의 소프트웨어 계층으로서 FTL (Flash Translation Layer)은 LPN을 PPN으로 매핑하는 기능, 무효화된 페이지들이 포함된 블록을 지워서 재사용할 수 있도록 처리하는 기능(garbage collection) 및 SSD 내부의 모든 블

• 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: 2018. 07. 24, Revised: 2018. 10. 26, Accepted: 2018. 10. 27.
• This study was supported by 2017 Research Grant from Kangwon National University(No. 520170078)

록들이 골고루 사용되도록 하여 SSD의 수명을 연장시키는 기능(wear leveling) 등을 담당한다.

SSD에서 주소 매핑 정보를 페이지 단위로 관리하기 위해서는 필요한 RAM의 용량이 방대한데 소형 임베디드 시스템에서는 RAM의 용량을 줄이는 것이 매우 중요하다. 페이지 단위로 매핑정보를 관리하면서도 RAM의 용량을 줄이기 위해서는 매핑 정보를 플래시 메모리에 기록하고 자주 사용되는 매핑 정보만 RAM에 캐시형태로 관리해야한다. 본 논문에서는 플래시 메모리에 기록하는 매핑 정보를 표현하는 방법을 개선하여 압축하여 표현함으로써 기존의 방법에 비해서 두 배의 정보를 기록하도록 한다. 이것을 통해서 소요되는 RAM의 용량을 줄이면서도 SSD의 성능을 개선할 수 있는 방안을 제시한다.

II. Related Works

LPN을 PPN으로 매핑하는 방법으로는 크게 페이지 단위로 처리하는 페이지 매핑과 블록단위로 처리하는 블록 매핑, 그리고 이 둘을 절충하여 사용하는 하이브리드 매핑이 있다[1]. 페이지 매핑은 LPN 별로 PPN을 바로 매핑하기 때문에 단순해 보이지만 LPN마다 대응되는 PPN 정보를 관리해야 하므로 그 정보가 너무 방대해서 RAM에 관리하기가 곤란하다. 블록 매핑은 주소 매핑을 블록 단위로 처리하는 것으로서 논리적 블록 번호를 물리적 블록 번호로 변환하고, 해당 블록내의 특정 페이지의 위치는 동일한 오프셋을 적용한다. 따라서 RAM에 관리해야 할 매핑 정보의 양이 페이지 매핑 방식에 비해서 한 블록 당 페이지 개수의 비율로 줄일 수 있다.

예를 들어서 64G 바이트 용량의 SSD를 가정해보자. NAND 플래시 메모리 칩은 다양한 구성으로 설계되는데, 여러 가지 FTL들 간의 성능 비교를 위해서, 대표적인 것으로서 한 페이지의 크기는 2K 바이트이고, 한 블록은 64개의 페이지들로 구성된다고 가정한다[2]. 페이지 매핑은 LPN마다 대응되는 PPN 정보를 관리해야 한다. 64G 바이트 용량의 SSD에는 2K 바이트 크기의 페이지가 32×10^6 개이고 한 항목 당 PPN을 4 바이트로 기록한다면 변환 정보로만 128M 바이트의 RAM이 소요된다. 반면에 블록 매핑을 적용하면 한 블록은 64개의 페이지들로 구성되므로 변환 정보를 위한 RAM의 용량은 1/64인 2M 바이트로 줄어든다.

블록 매핑의 문제점은 각 페이지의 블록 내 위치가 그대로 유지되어야 하므로 한 페이지만 갱신해도 새로운 블록에 나머지 페이지들도 전부 복사해야하는 것이다. 이 문제를 해결하기 위해서 하이브리드 매핑에서는 일정한 분량의 로그 블록들을 두고 쓰기 작업은 이 로그 블록들에 페이지 단위로 기록한다. 로그 블록들이 다 소모되면 로그 블록들의 유효한 페이지들을 기존의 데이터 블록들에 한꺼번에 반영하여 새로운 블록에 기록함으로써 유효 페이지들의 복사부담을 줄인다 [3-6].

페이지 매핑을 적용하면서도 RAM의 용량을 줄이는 방법으로서 DFTL에서는 페이지 매핑 정보를 플래시 메모리 페이지 (TP: Translation Page)에 저장한다[7]. RAM에는 TP들의 주소 목록만 기록한다. 주소 매핑이 필요할 때마다 플래시 메모리에서 TP를 읽어오려면 시간이 많이 소모되므로 자주 사용되는 논리적 페이지들에 대한 정보만 RAM에 캐시 형태로 관리함으로써 시간을 단축한다. 이때 캐시의 성공률을 높이는 것이 매우 중요한데 응용 프로세스들이 파일을 접근할 때는 대체로 시간적 지역성과 공간적 지역성을 가지고 있으므로 높은 성공률이 가능하다. 2K 바이트 크기의 한 페이지에는 4 바이트 크기의 PPN을 512개 기록할 수 있으므로, 앞에서 언급한 64G 바이트 용량의 SSD에는 32×10^6 개의 페이지가 있으므로 $32 \times 10^6 / 512 = 64 \times 10^3$ 개의 TP들이 플래시 메모리에 기록된다. 따라서 TP들의 주소 목록을 위해서는 TP 별로 PPN 4바이트를 사용하면 $64 \times 10^3 \times 4$ 바이트 = 256K 바이트의 RAM이 사용된다. 여기에 적절한 크기의 주소 변환 캐시를 추가하더라도 필요한 RAM 용량이 일반적인 페이지 매핑 방식의 128M 바이트나 블록 매핑 방식의 2M 바이트에 비해서 대폭 줄일 수 있는 것이다.

DFTL은 주소변환 캐시를 단순히 <LPN, PPN> 쌍의 배열로 관리하는데 비해서 CFTL은 동일한 크기의 캐시에 보다 많은 항목들을 표현하는 방법을 제안하였다[10]. CFTL은 연속된 LPN들을 하나의 캐시 항목에 표시할 수 있도록 캐시의 각 항목을 <LPN, PPN, k> 형식으로 표현하는데, LPN 부터 연속된 k 개의 페이지들이 플래시의 PPN부터 연속된 k개의 페이지들에 기록되어 있음을 표시하도록 하였다. 또한 하나의 논리적 블록이 하나의 물리적 블록에 기록된 것들은 별도로 블록 매핑 캐시에 기록하였다. 자료구조가 매우 복잡하긴 하지만 이렇게 함으로써 캐시에 더 많은 정보를 기록할 수 있고 따라서 캐시의 성공률도 일정부분 향상시키는 효과가 있다.

DFTL은 주소 변환 캐시를 페이지 단위로 관리하므로 시간적 지역성에 대한 특성은 잘 활용하지만 공간적 지역성을 충분히 활용하지 못하고 있다. 프로세스들이 파일을 접근하는 데 있어서 특정 논리적 페이지가 사용되었다면 인접한 논리적 페이지들도 사용될 가능성이 많으므로 이들을 묶어서 관리할 필요가 있는 것이다. TPC-FTL에서는 이러한 관점에서 주소변환 캐시를 TP 단위로 관리함으로써 캐시의 성공률을 높였다[8]. 하나의 TP에는 512개의 인접한 LPN을 위한 항목들이 포함되므로 캐시에 등록된 후에는 인접한 LPN들의 정보가 함께 저장되어 있어서 공간적 지역성에 대한 특성이 반영되는 것이다. TPC-FTL이 시간적 지역성도 유용하게 활용되기 위해서는 주소변환 캐시를 위한 RAM은 충분한 개수의 TP들을 저장할 수 있는 용량을 확보해야한다.

본 논문에서는 DFTL이나 TPC-FTL과 유사하게 페이지 매핑을 적용하면서도 TP에 매핑정보를 기록하는 방식을 개선하여 하나의 TP에 두 배의 정보를 압축하여 기록하도록 하였다. 따라서 전체 TP의 개수는 반으로 줄어들고 따라서 TP들의 주소를 기록하기 위한 RAM의 용량도 반으로 절약할 수 있다.

III. Compact Representation of Translation Pages

1. Representation of Translation Pages

본 논문에서 제안하는 CTP(Compact Translation Page) FTL에서 주소 매핑 정보는 그림 1과 같이 플래시 메모리 페이지에 기록되는 TP들과, 이들의 목록인 TPD (TP Directory)의 2 단계로 관리한다. TPD (Translation Page Directory)에는 TP들이 기록된 플래시 메모리 페이지 번호인 TPN (Translation Page Number)을 기록하고, TC (Translation Cache)에는 자주 사용되는 매핑 정보를 관리한다. 파일시스템으로부터 요청된 LPN에 대하여 TC에 포함되어 있는지 검사하고, 있으면 그 정보를 PPN으로 결정한다. TC에 존재하지 않으면 TPD에서 해당 TP가 기록된 플래시 메모리 페이지 주소인 TPN을 확인하고 해당 TP를 읽어서 TC에 갱신한 후에 이 정보를 이용하여 PPN을 확정한다. TC의 각 항목에는 TP와 함께 그것의 TPD 인덱스 (TPDI)도 함께 기록하는데, 이 정보를 이용하여 TC에 존재하는지를 검사한다. 이러한 2 단계 구조는 DFTL이나 TPC-FTL과 유사하지만 TP에 정보를 기록하는 방식과 캐시를 관리하는 방식에서 차이가 있다.

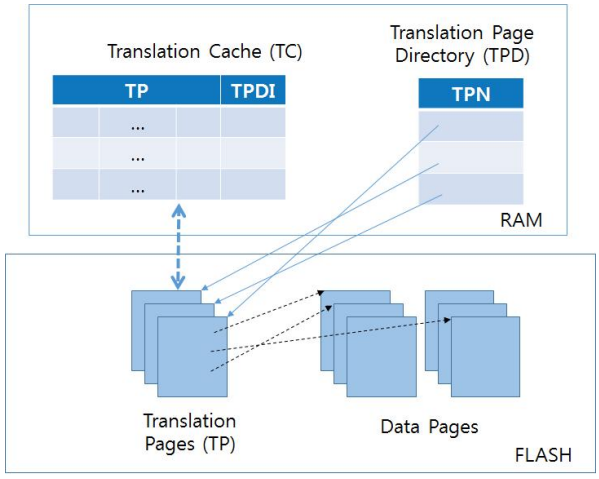

Fig. 1. Two Level Structure of Address Translation

LPN으로부터 PPN을 결정하는 과정은 그림 2와 같다. LPN의 비트들을 두 부분으로 나누어서 상위 비트 부분을 TPD의 인덱스 (TPD Index)로 사용하고 하위 비트 부분은 TP의 인덱스 (TP Index)로 사용한다. 예를 들어서 하나의 TP에 512개의 항목을 기록한다면 LPN의 하위 9비트는 TP내의 512개를 구별하는 TP Index이고 나머지 상위 비트들은 TPD Index가 된다. 특정 LPN의 읽기나 쓰기 요청이 오면 TPD에서 TPD Index를 적용하여 TP의 주소인 TPN을 확인하고 그 TP를 읽은 후에 TP Index를 활용하여 PPN을 확정한다.

파일시스템에서 읽기 요청이 오면 그 PPN을 결정하는 방법은 앞에서 설명한 바와 같다. 특정 LPN 페이지의 쓰기 요청에 대해서는 새로운 페이지에 쓰기를 실행하고 그 PPN을 TP에 반영한다. 읽기 요청 때와 마찬가지로 그 TP가 TC에 없으면 플래시 메모리에서 읽어서 TC에 등록한 후에 PPN을 반영한다. TC가 넘치게 되면 일부 항목을 선택하여 TP를 플래시 페이지에 기록하고 그 TPN을 TPD에 등록한다. TC의 관리 정책으로는 최근에 사용된 것 위주로 유지하는 LRU (Least Recently Used)를 적용한다. 이러한 구조는 기본적으로 TPC-FTL과 동일하지만 CTP에서는 TP에 관리하는 정보를 압축하여 표현하는 방식을 정용한다. DFTL에서는 TC를 TP 단위가 아니라 LPN 단위로 관리하며 <LPN, PPN> 쌍으로 기록한다.

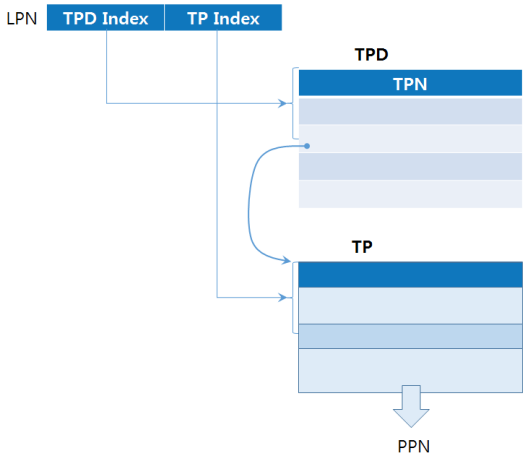

Fig. 2. Translation of LPN to PPN

TP에 매핑 정보를 표현하는 방법은 다음과 같다. 기존의 DFTL과 TPC-FTL은 그림 3 (a)와 같이 TP를 단순히 PPN의 배열로 기록한다. 즉 TP의 TP Index 번째 항목이 바로 PPN이 되는 것이다. PPN은 물리적 블록 번호(PBN: Physical Block Number)와 그 블록 내의 페이지 오프셋(PO: Page Offset)으로 구성되어 있다. 하나의 블록은 64개의 2K 바이트 페이지들로 구성된다면 PO는 PPN의 하위 6비트이고 PBN은 나머지 상위 비트들이 된다.

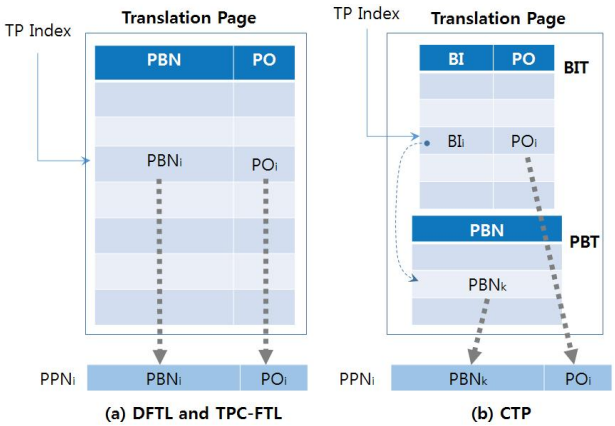

Fig. 3. Representation of Translation pages

CTP에서는 매핑 정보를 압축하여 표현하기 위해서 그림 3 (b)와 같이 TP를 PBT (Physical Block Table)와 BIT (Block Index Table)의 두 부분으로 나누어 기록한다. PBT는 TP 내에서 사용하는 물리적 블록들의 목록을 기록한다. BIT에는 항목별로 PBT의 인덱스 (BI: Block Index)와 그 블록 내의 PO를 기록한다. LPN으로부터 PPN을 결정하는 과정은 다음과 같다. LPN의 TP Index를 적용하여 BIT에서 BI와 PO를 결정하고 BI를 활용하여 PBT에서 PBN을 결정한다. 결정된 PBN과 PO를 결합하면 PPN이 된다. 만약 TP 하나당 64개의 PBN들을 기록할 수 있다면 BI는 이들을 구별할 수 있는 6비트로 표현한다.

CTP에서는 2K 바이트 크기의 TP에 1024개의 PPN을 기록할 수 있다는 것을 다음과 같이 확인할 수 있다. PPN은 32비트 (4 바이트)로 표현된다고 가정한다. 한 블록은 64개의 페이지들로 구성되므로 PO는 6비트로 표현되고 PBN은 나머지 상위 26비트이다. PBT에는 64개의 PBN을 기록한다고 가정하면 BI는 6비트로 표현된다. 따라서 BIT 한 항목당 BI와 PO를 합쳐서 12비트가 필요하다. 1024개의 항목을 기록하기 위해서 BIT는 1024 x 12 비트 = 12,288 비트, 즉 1,536 바이트가 사용된다. 여기에 PBT를 위해서 64 x 26 비트 = 1,664 비트, 즉 208 바이트를 추가하면 총 1,744 바이트이므로 2K 바이트 크기의 TP 한 페이지에 충분히 기록할 수 있는 것이다. DFTL이나 TPC-FTL은 TP를 단순히 PPN들의 배열 형태로 기록하므로 2K 바이트 크기의 TP 한 페이지에 PPN을 512개 기록할 수 있다. 따라서 CTP는 기존의 방식에 비해서 하나의 TP에 두 배의 PPN 항목들을 기록할 수 있는 것이다.

TPD는 RAM에 관리하는데, 하나의 TP에 2배의 PPN들을 기록할 수 있다면 총 TP 개수는 반으로 줄어들고 따라서 TPD의 크기를 반으로 줄일 수 있다. 이렇게 절약한 RAM을 TC에 활용함으로써 비교적 적은 RAM 용량으로도 TC의 성공률을 높일 수 있게된다. 또한 동일한 크기의 TC에는 2배의 정보를 기록하므로 TC의 크기를 2배로 늘린 효과도 있다. 64G 바이트 용량의 SSD에 대하여 DFTL과 TPC-FTL은 TPD를 위해서, 2장에서 분석했듯이, 256K 바이트의 RAM이 사용된다. 이에 비해서 CTP에서는 그 절반인 128K 바이트를 사용하므로 나머지 128K 바이트는 TC에 할당하여 사용함으로써 캐시의 성공률을 높일 수 있게 된다.

2. Block Merge Operation

CTP에서는 하나의 TP에 등록하여 사용할 수 있는 블록들의 개수가 제한된다. 한 페이지 크기의 TP 내에서 PBT의 크기를 얼마정도 할당할 지를 정해야 하기 때문이다. 예를 들면 PBT를 최대 64개의 블록까지만 기록할 수 있도록 제한한다. 따라서 데이터 페이지들의 쓰기가 진행되면서 PBT에 등록된 블록들이 다 사용되면 기존의 항목들 중에서 일부를 제거해야 한다. 데이터 페이지들의 쓰기가 진행되면서 기존의 데이터 페이지들이 무효화되므로 PBT에 등록된 블록들에는 무효페이지들이 많이 포함되어 있게 된다. 따라서 가비지 컬렉션 작업을 통해서 유효 페이지 개수가 작은 블록들을 선택하여 새로운 블록에 복사하는 작업을 진행하면 PBT에는 등록된 블록수를 줄일 수 있다. 무효 페이지들만 남은 기존의 블록들은 지워서 재사용한다.

PBT에 블록들이 전부 등록된 상태에서 새로운 블록을 등록해야 하면 가비지 컬렉션에 해당하는 블록 병합 작업을 실시한다. 블록 병합에서는 기존의 블록들 중에서 유효페이지 개수가 가장 작은 블록을 선택하여 유효 페이지들만 새로운 블록에 복사하고, PBT에는 기존 블록을 제거하고 그 자리에 새로운 블록을 등록한다. 새로운 블록에는 아직 많은 수의 빈 페이지들이 남아있어서 새로운 데이터 페이지 쓰기에 계속 사용된다. 가비지 컬렉션 작업에서 유효 페이지 개수가 가장 작은 블록을 선택함으로써 유효페이지를 복사하기 위한 오버헤드를 줄일 수 있으며, 실제로는 유효 페이지가 전혀 없어서 페이지 복사 작업을 하지 않는 경우도 많이 발생한다.

여기서는 확률적인 방법으로 PBT 항목 갱신 시점의 가비지 컬렉션을 실행할 때 블록 병합을 위해 복사해야 하는 유효 페이지 수를 분석하고, 다음 장에서는 시뮬레이션을 통해서 그 성과를 확인하도록 한다. 하나의 TP에는 1024개의 페이지들이 기록되고, PBT에는 64개의 항목을 등록하며, 쓰기 대상인 페이지는 TP내의 페이지들 중에서 임의로 선택된다고 가정한다. 첫 번째 블록 PB_1의 첫 페이지 P_1에 기록된 데이터는 두 번째 페이지 P_2를 기록한 시점에 여전히 유효한 것으로 남아있을 확률은 p = 1023/1024 이다. 세 번째 페이지 P_3을 기록한 시점에는 확률이 p^2이다. 이제 PB_1의 64 페이지를 모두 기록하고 난 뒤에 두 번째 블록 PB_2의 첫 페이지에 기록하는 시점에 PB_1의 페이지들에 대하여 유효한 것으로 남아 있을 확률을 계산해 보면 P_1은 p^{64}이고, P_2는 p^{63}이고, k 번째 페이지 P_k는 이 확률이 p^{64-k+1}이다. 따라서 이 시점에 PB1에 남아 있는 유효 페이지 수는 $v_1 = p^{64} + p^{63} + \ldots + p^1 = (p^1 - p^{65})/(1 - p) = 62.01$ 이다. 64개의 페이지들 중에서 평균적으로 2개 정도가 무효 페이지이다.

PB_2의 64 페이지를 모두 기록하고 세 번째 블록 PB_3의 첫 페이지를 기록하는 시점에 PB_1에 남아 있는 유효 페이지 수는 $v_2 = v_1 \times r$ 이고, $r = p^{64}$이다. 따라서 64 개의 블록들을 모두 기록하고 난 뒤에 PB1에 남아있는 유효 페이지 수는 $v_{64} = v_1 \times r^{63} = 1.21$ 이다. 따라서 64개의 블록들을 모두 사용한 후에 블록 병합을 실시할 때 유효 페이지 수가 가장 작은 블록을 선택한다면 평균적으로 1.21개의 유효 페이지만 복사하면 되는 것이다. 이 정도의 오버헤드는 전체 응답시간에 미치는 영향이 미미할 것이다. 만약 임의의 페이지 쓰기가 아니라 순차 쓰기라면 이전 블록의 페이지들이 모두 순차적으로 무효화되므로 이전 블록들은 유효 페이지 수가 0일 것이다. 이러한 분석 결과는 다음의 성능 평가 부분에서 확인할 수 있다.

IV. Performance Evaluation

특정 FTL의 성능은 파일시스템에서 요청되는 페이지 읽기와 쓰기의 특성에 많이 좌우된다. 공정한 평가를 위해서 대표적으로 많이 사용되는 UMass 트레이스 파일을 적용하였다[9]. 이것은 대표적인 몇 가지 응용들을 실제 시스템에서 실행하면서 수집한 정보로서 통계적인 특성은 표 1과 같다. Financial 1과 2는 OLTP 처리를 하는 응용에서 수집한 것으로서 읽기와 쓰기가 적절히 섞여 있으며, 그 중에서도 1이 쓰기 요청 비율이 높다. WebSearch 1은 웹의 검색 위주로 이루어지는 응용에서 수집한 것으로서 읽기가 압도적으로 많이 이루어지며, 작업 요청 간의 평균 간격은 Financial 1과 2보다 훨씬 짧다. UMass 트레이스에서 WebSearch 2와 3은 특성이 WebSearch 1과 거의 유사하므로 생략하였다.

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

비교 대상 FTL로는 페이지 매핑을 기반으로 하는 DFTL과 TPC-FTL을 적용하였다. OPT는 단지 최적의 비교대상으로 삼기위한 것으로서 페이지 매핑 정보를 전부 RAM에 저장한 것을 가정하였으며, 따라서 매핑 정보를 위한 플래시 메모리 페이지 읽기나 쓰기를 위한 오버헤드가 전혀 없는 경우이다. 적용 대상 플래시 메모리 칩으로는 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[2].

동일한 크기의 RAM을 사용하여 성능을 비교해야 하므로 소요되는 RAM의 용량을 비교하기 위해서 SSD의 용량으로 64G 바이트를 적용하였다. 한 페이지의 크기가 2K 바이트이므로 총 32×10^6 개의 페이지들이 존재한다. DFTL과 TPC-FTL의 경우에는 TP 하나에 512 개의 PPN이 기록되므로 TPD에는 $32 \times 10^6 / 512 = 64 \times 10^3$ 개의 항목이 필요하고, 항목당 4바이트의 TPN을 기록하므로 TPD를 위해서 256K 바이트가 사용된다. CTP에서는 하나의 TP에 1024개의 항목이 기록되므로 TPD를 위한 RAM의 용량은 절반인 128K 바이트가 사용된다. 따라서 CTP에서는 남는 128K 바이트를 주소 변환 캐시에 활용하였다. 참고로 DFTL은 주소 변환 캐시에서 LPN 항목마다 <LPN, PPN> 쌍의 정보 외에도 수정여부를 표시하는 비트와 사용 시점을 기록하기 위한 상당한 용량의 RAM이 추가로 필요하지만 이 비교 평가에서는 포함하지 않았다. CTP와 TPC-FTL에서는 변환 캐시가 TP 단위로 구성되므로 수정여부 비트와 사용시점 기록이 TP 단위로만 하면 되므로 추가로 소요되는 RAM의 용량이 미미하다.

FTL들 간의 성능 비교를 위해서 응답시간을 적용하였다. 응답시간은 파일시스템에서 특정 페이지의 읽기나 쓰기 요청이 온 시점부터 해당 작업이 완료된 시점까지의 시간을 측정한 것이다. 오직 플래시 메모리 칩의 읽기와 쓰기를 위한 시간만 반영하였으며 캐시를 검사하는 시간 등의 SSD 제어기가 알고리즘을 수행하면서 소모하는 시간은 반영하지 않았다. 특히 DFTL의 경우에는 캐시 용량이 크면 항목수가 대폭 늘어나서 캐시 검사에 많은 시간을 소모하지만 본 응답시간 평가에는 포함하지 않았다. CTP의 경우에는 PBT에 새로운 항목을 등록하기 위해서 실시하는 블록 병합 과정에서 필요한 유효 페이지들의 복사에 필요한 시간은 반영하였다. 이 블록을 지우기 위한 시간은 반영하지 않았는데, 블록을 지우는 작업은 가비지 컬렉션 작업의 일부에 해당하며 별도의 시점에 실행하면 되기 때문이다. 비교대상 FTL들은 모두 가비지 컬렉션을 위한 시간을 전혀 반영하지 않았다.

트레이스 별로 시뮬레이션을 통한 평균적인 응답시간은 그림 4와 같다. 여기에 적용한 RAM 용량은 DFTL과 TPC-FTL의 TPD를 위한 256K 바이트에 캐시를 위한 8K 바이트를 추가하여 총 264K 바이트이다. 모든 트레이스에서 CTP가 DFTL과 TPC-FTL보다 응답시간이 단축되었으며 최적의 비교 대상인 OPT에 근접한 특성을 보여주고 있다. 특히 읽기 요청 위주의 트레이스인 WebSearch 1의 경우에는 응답시간이 DFTL의 절반 정도로 개선된다. 참고로 TPC-FTL은 읽기 위주의 트레이스에서는 DFTL보다 우수한 특성을 보여주지만 쓰기 요청이 많은 트레이스에서는 DFTL에 비해서 성능이 떨어진다.

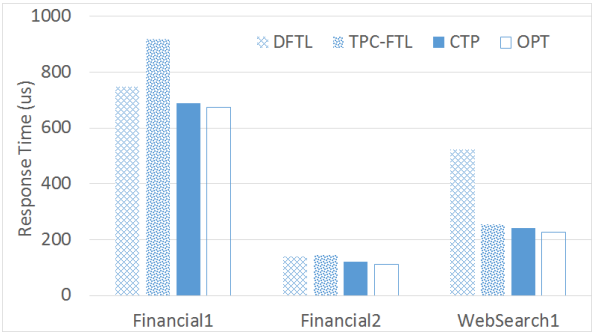

Fig. 4. Comparison of Response Times

그림 5는 RAM 용량에 따른 평균 응답시간의 변화를 보여준다. RAM 용량은 DFTL과 TPC-FTL에서 TPD를 위한 256K 바이트에서 추가된 RAM의 용량으로 표시하였다. OPT는 페이지 매핑 정보 전부를 저장하기 위한 충분한 RAM을 사용하는 것으로 하였다. CTP는 최적에 해당하는 OPT에 근접한 성능을 보여주며, 4K 바이트만 추가해도 (총 260K 바이트) 충분하다. 참고로, DFTL과 TPC-FTL의 경우에 Financial 1과 같이 페이지 쓰기가 많은 응용에서는 RAM의 용량을 늘려주면 응답시간이 개선되지만 WebSearch 1과 같이 페이지 읽기 위주이며 순차적인 읽기가 많은 응용에서는 RAM의 용량을 늘려도 별 효과가 없다.

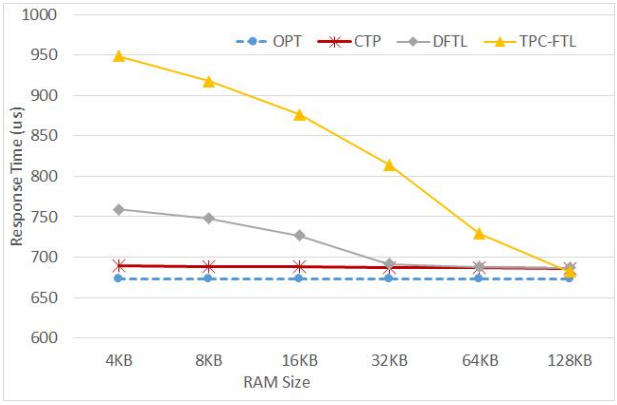

Fig. 5. Response Time According to RAM Size (Financial 1)

블록 병합이 실시되는 경우에 복사하는 유효 페이지 수는 표 2와 같이 Financial 1의 경우에는 평균 1.56 페이지, Financial 2는 평균 0.23 페이지로서 매우 작다. 이것은 앞에서 임의의 페이지를 기록한다는 가정하여 예측한 평균 1.21 페이지와 유사한 결과를 보여준다. Financial 2가 더 작게 나오는 것은 순차적인 페이지 쓰기가 일부 포함되어 있는 것으로 추정된다. WebSearch 1의 경우에는 유효 페이지 복사가 0으로 나타나는데 페이지 쓰기 작업이 드문데다가 대부분 순차적으로 쓰기가 이루어진 것으로 추정된다.

Table 2. Average Number of Valid Pages on Block Merge

Trace	Average Number of Valid Pages
Financial 1	1.56
Financial 2	0.23
WebSearch 1	0.00

V. Conclusions

용량이 큰 SSD에 대하여 페이지 매핑 정보는 매우 방대해서 RAM에 저장하기에는 RAM의 용량이 지나치게 크다. 이러한 문제를 해결하기 위해서 페이지 매핑 정보는 플래시 메모리의 페이지들에 기록하고 그 주소를 목록으로 RAM에 관리하는 방법을 사용한다. 여기에 매핑정보에 대한 캐시를 활용함으로써 매핑이 필요할 때마다 플래시 메모리 페이지를 읽어오는 오버헤드를 줄여준다. 본 논문에서 제안한 CTP는 TP에 매핑정보를 표현하는 방법을 개선하여 기존의 DFTL이나 TPC-FTL에 비해서 두 배의 정보를 기록함으로써 이 TP들의 목록을 위한 RAM의 용량을 반으로 절약하고 이 부분을 캐시에 활용함으로써 전체적인 성능을 높일 수 있었다. 대표적인 트레이스들을 활용하여 성능평가를 해본 결과 기존의 DFTL이나 TPC-FTL 에 비해서 응답시간을 대폭 단축할 수 있음을 확인하였다.

REFERENCES

[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] 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*

[3] 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.

[4] 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.

[5] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, “A superblock-based flash translation layer for NAND flash memory,” in Proc. 6th ACM IEEE Int. Conf. Embedded Softw., pp. 161-170, Oct. 2006

[6] 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.

[7] 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.

[8] 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

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

[10] 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.

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.