Journal of The Korea Society of Computer and Information
Vol. 22 No. 7, pp. 1-7, July 2017

www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.07.001

STP-FTL: An Efficient Caching Structure for Demand-based Flash Translation Layer

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

Abstract

As the capacity of NAND flash module increases, the amount of RAM increases for caching and maintaining the FTL mapping information. In order to reduce the amount of mapping information managed in the RAM, a demand-based address mapping method stores the entire mapping information in the flash and some valid mapping information in the form of cache in the RAM so that the RAM can be used efficiently. However, when cache miss occurs, it is necessary to read the mapping information recorded in the flash, so overhead occurs to translate the address. If the RAM space is not enough, the cache hit ratio decreases, resulting in greater overhead. In this paper, we propose a method using two tables called TPMT(Translation Page Mapping Table) and SMT(Segmented Translation Page Mapping Table) to utilize both temporal locality and spatial locality more efficiently. A performance evaluation shows that this method can improve the cache hit ratio by up to 30% and reduces the extra translation operations by up to 72%, compared to the TPM scheme.

▸Keyword: Solid State Drive, Flash Memory, Flash Translation Layer, Demand-based address mapping, Cache Management

I. Introduction

NAND 플래시 메모리는 기존 하드디스크에 비해 읽기/쓰기 속도가 빠르고 적은 소음, 높은 내구성과 낮은 전력 소모량 등의 장점이 있다. NOR 플래시 메모리에 비해 제조 단가가 싸고 대용량화가 가능하기 때문에 모바일 기기 및 일반 PC 등에서 하드디스크를 대체하는 저장장치로 활용되고 있다. 하지만 하드디스크를 대체하기 위해서 고려해야 할 플래시 메모리의 특성이 있다.

가격이 기존 하드디스크에 비해 상대적으로 비싸고, 지우고 재기록 가능한 횟수에 제한이 있으며 제자리 쓰기를 할 수 없다는 점이다. 기술이 발달함에 따라 가격이 낮아지고 재기록 가능 횟수의 성능이 높아지고 있다. 하지만 하드디스크와 달리 플래시 메모리는 제자리 쓰기 전 삭제 연산을 해야 하는데, 쓰기 연산은 페이지 단위이지만 삭제 연산은 블록 단위로 이루어진다. 따라서 데이터를 갱신하기 위해서 해당 블록의 내용을 삭제 후 써야하고 삭제와 쓰기 연산은 읽기 연산에 비해 많은 시간이 필요하기 때문에 발생하는 오버헤드가 커지게 된다. 이러한 제약들로 인해 플래시 메모리를 효율적으로 사용하기 위한 여러가지 알고리즘이 제안되었다[1-4].

제자리에 기록을 할 수 없는 문제는 플래시 메모리에 맞는 YAFFS, JFFS, F2FS [5, 6 ,7] 등의 파일시스템을 사용하여 해결하거나 기존 파일시스템을 사용할 수 있도록 주소변환 계층인 FTL(Flash Translation Layer)을 통해 요청된 논리 주소를 실제 데이터가 있는 물리 주소로 변환하여 맵핑시켜 줌으로써 해결한다. FTL은 주소변환 뿐만 아니라 전체 블록이 균등하게 사용되도록 조절하여 플래시 메모리의 수명을 늘려주는 기능(wear leveling)과 유효하지 않은 페이지들을 효과적으로 관리하기 위한 기능(garbage collection)이 포함되는데 본 논문에

• First Author: Hwan-Pil Choi, Corresponding Author: Yong-Seok Kim
*Hwan-Pil Choi (rdzmoon@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: 2017. 05. 17, Revised: 2017. 06. 12, Accepted: 2017. 07. 18.
• This study is supported by 2016 Research Grant from Kangwon National University (No. 520160202)

서는 주소 변환 방법에 대해 초점을 맞춘다.

기존 페이지 레벨 맵핑 방법은 플래시 메모리의 용량이 커질수록 필요한 RAM의 크기가 커지게 된다. 플래시 메모리의 용량이 늘어남에 따라 많은 수의 페이지가 선형적으로 증가하며 각각의 페이지를 맵핑하기 위한 페이지 테이블의 크기 또한 크게 증가하게 되면서 매우 큰 용량의 RAM 공간을 필요로 하게 된다. 전체 256MB 용량의 플래시 메모리에 각 페이지의 크기가 2KB인 경우 128K개의 맵핑 정보가 필요하며, 하나의 맵핑 정보에 기록되는 주소를 4 Byte로 표현한다면 512KB(128K*4B)가 필요하다.

전체 용량이 GB, TB로 커지게 되면 필요한 맵핑 테이블의 크기 또한 MB, GB로 커지게 되어 부담이 커지게 된다. 이러한 문제가 있음에도 페이지 레벨 맵핑 방법은 캐시 적중시 바로 주소를 변환할 수 있고 쓰기 요청된 논리 주소에 대해 남아있는 사용가능한 페이지가 있다면 삭제나 복사 연산없이 바로 쓰기를 할 수 있다. 따라서 논리 주소를 어느 물리 주소에나 맵핑할 수 있어 유연하게 맵핑 정보를 관리 할 수 있게 된다.

다양한 맵핑 방법에서 발생하는 문제들을 해결하기 위해 많은 연구들이 이루어져 왔다[1-4, 10, 11]. 그 중 페이지 맵핑의 장점을 활용하면서도 RAM 소요량을 줄이는 방법으로 DFTL(Demand-based Page-mapped FTL)[2]이 제안되었다. DFTL은 최근 요청된 주소의 맵핑 정보를 RAM에 캐시 형태로 관리하며, 전체 맵핑 테이블 정보를 플래시 내에 기록하여 관리하고 맵핑 테이블 정보가 기록된 페이지들을 RAM상에서 맵핑시켜 관리한다. 따라서 상대적으로 적은 RAM 공간을 이용하여 페이지 맵핑 구조를 사용할 수 있는 방법을 제안했다. 하지만, 맵핑 테이블에 요청된 하나의 논리 주소와 그에 대응하는 물리 주소만 기록하는 것은 공간 지역성을 활용하지 못하여 연속된 요청이 들어온 경우 캐시 적중률이 떨어지게 되고 캐시 미스가 발생한 경우 플래시에 기록된 맵핑 정보가 들어있는 TP (Translation Page)를 읽어와야 하는 오버헤드가 발생하게 된다.

본 논문에서는 페이지 레벨의 요구 기반 주소 맵핑 방법에 대해 논리 주소와 그에 대응하는 하나의 물리 주소만 기록하는 것이 아니라 TP 전체를 기록하거나 일부분을 기록할 수 있게 2개의 맵핑 테이블을 사용하고 함으로써 캐시 적중률을 높이고 발생하는 추가 오버헤드를 줄일 수 있는 방법을 제안한다.

논문의 구성은 2장에서 기존에 제안된 요구 기반의 주소 맵핑방법들에 대해 설명하고 기존 맵핑 방식의 문제점을 살펴본다. 3장에서는 제안하는 방법에 대해 구체적으로 기술하며, 4장에서 제안한 방법의 성능을 평가하고 5장에서 결론 및 향후 연구에 대해 설명한다.

II. Related Works

전체 맵핑 정보를 플래시 내에 기록하고 일부의 맵핑 정보를 RAM상의 페이지 맵핑 테이블에 사용하는 요구 기반의 주소 맵핑 방식은 기존의 페이지 맵핑을 위한 테이블의 크기를 줄이면서도 좋은 성능을 보여주었다. DFTL[2]은 최근에 요청된 페이지를 CMT(Cached mapping Table)라는 페이지 레벨 맵핑 테이블을 사용하여 관리한다. CMT는 LRU(least recently used) 알고리즘 정책으로 관리되며, 요청이 CMT에 적중한 경우 바로 처리하게 되지만 적중하지 못한 경우 플래시 내의 TP를 읽어와서 갱신해야 하는 오버헤드가 발생하기 때문에 CMT의 적중률은 DFTL의 성능에 많은 영향을 미치게 된다.

DFTL은 요청이 들어오면, CMT에 적중된 경우 일반적인 페이지 맵핑과 동일하게 논리 주소에 맵핑된 물리 주소를 바로 이용할 수 있게 된다. CMT에 해당 맵핑 정보가 존재하지 않으면 GTD(Global Translation Directory)로부터 해당 맵핑 정보가 있는 TP의 위치를 검색하고, TP를 읽어온 뒤 해당 논리 주소에 대한 물리 주소의 맵핑 정보를 CMT의 빈자리에 기록하고 요청을 처리하게 된다. 이때, CMT가 가득 찬 경우라면 CMT에서 가장 오래된 맵핑 정보를 제거하는 처리를 한다. 만약 제거하는 맵핑 정보가 수정된 경우라면 수정된 맵핑 정보를 플래시에 기록하고 연관된 GTD를 갱신시켜준다.

요구 기반의 주소 맵핑에서 플래시에 기록되는 TP는 맵핑 정보를 가지고 있는 하나의 페이지를 의미하고, 한 페이지의 크기가 커질 수록 기록되는 주소의 수가 증가하게 된다. 예를 들어 하나의 주소를 4B로 나타내면, 512B 크기의 페이지는 128개의 주소를 기록할 수 있고 2KB 크기의 페이지는 512개, 4KB 크기의 페이지는 1024개의 주소를 기록 할 수 있게 된다.

DFTL의 경우 요청된 논리 주소에 해당하는 물리 주소의 정보만 맵핑 테이블에 기록하게 된다. 이러한 경우 캐시 미스가 발생하게 되면 요청된 하나의 주소를 얻기위해 한 페이지를 읽어오게 된다. 그리고 하나의 맵핑 정보만 가져오게 되면 공간지역성을 활용할 수 없기 때문에 연속된 요청이 들어오게 되면 연속적으로 캐시 미스가 발생하게 되고 추가적인 오버헤드가 커지게 된다. 이러한 경우 성능을 개선하기 위해 캐시에 하나의 주소를 기록하지 않고 TP를 기록하는 방법이 제안되었다[3]. 공간 지역성을 잘 이용할 수 있어 연속된 요청에 대한 성능이 크게 좋아지게 되지만 페이지 크기가 커질 수록 맵핑 테이블의 엔트리 개수가 크게 줄어들게 되어 시간 지역성에 대해서 적절하지 않게 된다. RAM 크기가 작아질수록 이러한 문제가 더욱 심각해진다.

시간 지역성을 활용하기 위해서는 충분한 엔트리 개수가 확보되어야 하고 공간 지역성을 위해서 연속된 주소 정보가 필요하게 된다. 따라서 TP만 맵핑 테이블에 사용하는 것이 아니라 일부의 TP를 맵핑 테이블에 기록하게 함으로써 엔트리 개수를 확보하고 연속된 주소정보를 기록하여 공간지역성과 시간지역성을 모두 활용할 수 있게 할 수 있다.

본 논문에서 제안한 방법과 비슷하게 2개의 맵핑 테이블을 사용하는 방법 또한 제안되었다[4]. 논리 주소와 물리 주소를 1:1로 맵핑하는 방법과 TP를 맵핑하는 방법을 같이 사용 하였는데, 기본적으로 논리 주소와 물리 주소를 하나씩 맵핑하는 방법을 함께 사용함으로써 공간지역성을 보다 잘 활용할 수 없었고

엔트리 제거시 연관된 TP를 관리하기에 적절하지 않게 된다.

요구 기반 주소 맵핑 방식의 경우, 전체 주소정보를 각각의 TP에 기록하고 있으며 TP는 플래시 내에 저장되어 있다. 캐시 미스가 발생하여 새로운 주소를 기록해야 할 때 테이블이 가득 차 있으면 특정 엔트리를 선택하여 제거해야 하는데, 해당 엔트리가 수정된 경우 수정된 정보를 TP에 기록해 주어야 한다.

하나의 엔트리가 제거될 때마다 연관된 TP를 플래시에 기록하게 하면 플래시에 잦은 쓰기 연산이 필요하게 되어 성능이 떨어지게 된다. 따라서 엔트리에 하나의 주소를 맵핑하는 경우, 해당 주소에 대한 정보만 TP에 다시 쓰는 것이 아니라 TP에 연관되어있는 논리 주소가 맵핑 테이블에 있으면 한번에 같이 기록하는 방법을 사용한다. 이 과정에서 맵핑 테이블에 연관된 주소가 있는지 검사하는 추가적인 오버헤드가 발생하게 된다.

맵핑 테이블의 엔트리에 한 페이지를 기록하는 경우에 연관된 주소가 있는지 확인할 필요가 없게 된다. 따라서 하나의 주소를 맵핑하거나 일부의 TP 정보를 기록하는 맵핑 테이블에서는 연관된 주소 정보를 관리할 수 있는 방법이 필요하다.

III. The Proposed Scheme

본 논문에서는 주소변환캐시에 TP와 함께 TP를 분할한 부분 단위로 관리하는 STP-FTL(Segmented Translation Page Flash Translation Layer)을 제안한다. 페이지 레벨의 주소 맵핑 정보가 NAND 플래시에 저장되는 각 TP에는 여러개의 맵핑 정보를 가지고 있게 되는데 주소를 4 Byte로 나타내는 경우, 하나의 페이지 크기가 512B, 2KB, 4KB인지에 따라 각각 128, 512, 1024개의 맵핑 정보를 기록할 수 있게 된다. TP의 내용을 캐시에 관리하는 방법에 있어서 기존에 제안된 맵핑 방법은 요청된 1개의 논리주소에 대해 1개의 물리주소를 캐시에 기록하거나 연관된 TP 전체를 기록하는 방법이 제안되었다.

1개의 물리주소를 캐싱하는 방법은 공간 지역성을 활용하지 못해 연속된 요청을 제대로 처리할 수 없기 때문에 적중률이 크게 낮아지게 된다. 반면 TP전체를 캐싱하는 방법은 연속된 요청을 처리하는데 좋은 성능을 보이지만 페이지의 크기가 커짐에 따라 맵핑 엔트리 개수가 크게 낮아지면서 시간 지역성에 약해진다. 따라서, 본 논문에서는 TP전체와 TP의 일부를 맵핑하는 2가지 종류의 캐시 맵핑 테이블을 사용하여 요청으로 인해 읽어들인 TP를 맵핑하여 시간, 공간 지역성을 보다 효과적으로 활용함으로써 적중률을 높이고 추가적으로 발생하는 플래시에 읽기, 쓰기 연산을 줄일 수 있는 방법을 제안한다.

1. Structure of STP-FTL

STP-FTL의 전체적인 구조는 그림 1과 같다. 캐시 맵핑 테이블(CMT)은 SMT(Segmented Translation Page Mapping Table)와 TPMT(Translation Page Mapping Table)로 구성되어 있으며, 플래시 내에 있는 전체 TP 정보들을 관리하기 위해 GTD가 사용된다.

SMT는 각각의 엔트리에 TP의 일부분인 일정 개수의 주소를 기록한다. 기록되는 주소의 숫자는 TP를 얼마나 나눌지 결정하는 D(Divisor)에 따라 달라지게 된다. 만약, 페이지의 크기가 4KB인 경우에 각 TP는 1024개의 주소를 기록하게 된다. 이 때, D가 8이면 SMT의 엔트리 하나에 기록되는 주소 수는 128개가 되고 16이면 64개가 된다. SMT 엔트리 하나에 기록되는 주소 수는 TP에 기록되는 주소 수를 D로 나눈 값을 가지게 된다.

SMT는 TP의 일부분인 STP를 기록하며 각 엔트리 별로 s-lpn(start logical page number)을 가지고 있다. s-lpn은 SMT에 기록된 맵핑 정보(STP)의 시작 논리 주소를 기록한 것으로 요청이 발생하면 캐시 적중 여부를 판단하는데 사용하며, SMT에서 TPMT로 이동하거나 맵핑 테이블에서 제거 될 때, 연관된 TP의 주소를 GTD에서 찾는데 사용된다.

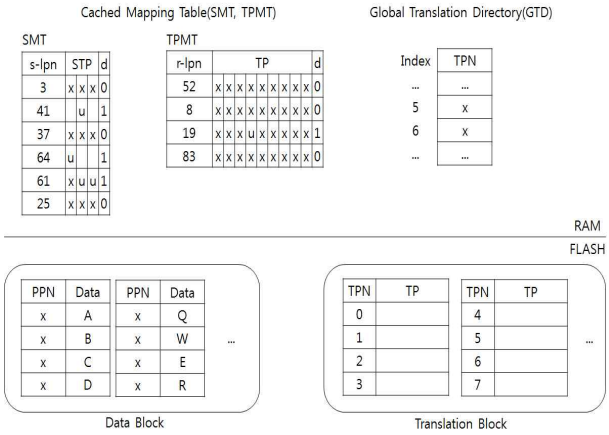

Fig. 1. Structure of STP-FTL

TPMT는 전체 TP를 기록하며 r-lpn(recently used logical page number)을 가지고 있다. r-lpn은 최근 사용된 논리 주소를 기록한 것으로 해당 엔트리에 요청된 주소가 있는 경우 지속적으로 갱신되며 캐시 적중 여부를 판단하거나 TPMT에서 SMT로 이동할 때 사용된다.

SMT와 TPMT의 각 엔트리에는 해당 엔트리가 수정되었는지 여부를 확인하는 비트가 있다. 쓰기 요청시 캐시에 적중하게 되면 새로 쓰여진 물리주소로 맵핑 정보를 갱신하면서 해당 엔트리가 수정되었음을 표시한다.

읽기와 쓰기 요청에 대해서 캐시 미스가 발생하여 새로운 엔트리를 추가해야 하는 경우, 읽기 요청은 미스가 발생하면 하나의 TP를 읽어오기 때문에 읽어온 정보를 모두 사용하기 위해서 TPMT에 기록하게 한다. 쓰기 요청은 TP를 읽어올 필요가 없어서 이후 주소의 정보를 알 수 없기 때문에 TPMT에 기록하게 되면 사용할 수 있는 공간 낭비가 커지게 되므로 SMT에 우선 기록하게 된다.

새로운 요청이 캐시 미스가 발생했을 때 해당 요청의 LPN이

SMT에 기록된 LPN과 같은 TP에 연관되어 있는 경우 TPMT에 함께 기록하게 된다. 이동되는 SMT 엔트리는 제거되며 TPMT가 가득 찬 상태라면 SMT와 TPMT간 엔트리 교환이 발생한다. TPMT에서 SMT로 이동시에는 r-lpn을 사용하여 최근 사용된 부분을 SMT에 기록하게 된다. 이렇게 연관된 주소를 관리해줌으로써 하나의 TP는 SMT나 TPMT 내에 하나의 엔트리에 기록되게 함으로써 특정 LPN의 캐시는 둘 중 하나에만 등록되게 된다.

SMT와 TPMT가 가득 찬 상태라면 새로운 주소 정보를 엔트리에 추가하기 위해 하나의 엔트리를 제거해야 한다. TPMT에 추가하는 경우는 TPMT의 한 엔트리를 r-lpn을 사용하여 SMT로 이동하게 되고, 만약 SMT가 가득 찬 경우 기본적으로 SMT의 마지막 엔트리를 희생(victim) 엔트리로 선택하여 제거하게 된다. 추가적으로 일정 범위 내의 수정되지 않은 엔트리를 우선 선택하게 하는 기능을 추가로 사용할 수 있다. 만약 TPMT의 엔트리가 수정된 상태라면 플래시에 기록하고 수정되지 않은 상태로 SMT로 이동한다. SMT에 추가하는 경우는 그대로 SMT에서 제거하게 되며, 제거시 해당 엔트리가 수정된 상태라면 GTD를 사용해 연관된 TP를 읽어온 후 변경된 내용을 적용하여 플래시에 기록하게 된다.

SMT에 TP의 일부분을 기록하고 TPMT에 전체 TP를 기록하는 방법을 사용함으로써 시간, 공간 지역성을 효과적으로 사용할 수 있게 된다. 전체 TP만 캐싱하는 방법을 사용하게 되면 페이지의 크기가 커짐에 따라 캐시에 기록 할 수 있는 엔트리의 개수가 줄어들기 때문에 시간 지역성에 대한 성능이 낮아질 수 있지만 TP의 일부를 기록하는 SMT를 같이 사용하여 페이지 크기가 커져도 충분한 엔트리 개수를 확보할 수 있기 때문에 시간 지역성을 효과적으로 활용할 수 있게 된다.

2. Caching Mechanism

STP-FTL은 쓰기, 읽기 요청인지에 따라 각각 SMT, TPMT에 우선 기록하게 된다. 들어온 요청은 엔트리 수가 적은 TPMT를 우선 살펴보고 이후 SMT를 검색하여 캐시 적중 여부를 판단한다. s-lpn과 r-lpn을 활용하여 엔트리 단위로 검색하기 때문에 검색에 필요한 시간을 줄일 수 있다. 그림 2는 STP-FTL에서 주소를 변환하는 전체적인 과정을 간략하게 나타낸 것이다.

SMT를 검색할 때는 요청된 lpn이 해당 엔트리와 같은 TP에 기록되는지 여부를 함께 확인하며 해당 엔트리의 요청된 위치에 물리 주소가 기록되어 있지 않은 경우 캐시 미스로 판단한다. 캐시에 적중된 경우는 바로 주소를 변환하여 사용할 수 있게 된다.

만약 읽기 요청으로 TPMT에 기록된 TP에 쓰기 요청이 적중되면 요청된 lpn을 해당 엔트리의 r-lpn 값으로 교체하고 TP내의 해당 위치의 값을 요청에 의해 플래시에 기록된 물리 주소로 변경한 뒤 해당 엔트리가 수정되었음을 표시한다. 반대로 쓰기 요청 후 SMT에 새로 쓰여진 STP의 엔트리에 읽기 요청이 발생했을 때 물리 주소가 기록된 위치에 요청된 경우 라면 해당 위치의 물리 주소를 바로 반환한다. 만약 해당 엔트리에 물리 주소가 기록되어 있지 않다면 캐시 미스가 발생하고 요청된 lpn에 해당하는 TP를 GTD를 검색하여 플래시에서 읽어온다. 연관된 엔트리가 SMT에 있으므로 해당 엔트리의 내용으로 읽어온 TP를 갱신 후 TPMT에 추가하게 된다.

```
Input: Request's Logical Page Number(requestlpn)
Read Request:
 if requestlpn miss in CMT then
   TPrequest ← consult_GTD(requestlpn)
   if associated entry exist in SMT then
     update_TP(TPrequest, SMT_associated_entry)
     erase_entry(SMT_associated_entry)
   else
     if TPMT is full then
       erase_entry(SMT_last_entry)
       SMT ← TPMT(TPMT_last_entry)
   TPMT ← TPrequest
   update_entry(TPMT_new_entry)
 return ppn

Write Request:
 if requestlpn miss in CMT then
   if associated entry exist in SMT then
     SMT(SMT_associated_entry)                    ↔
TPMT(TPMT_last_entry)
   else
     if SMT is full then
       erase_entry(SMT_last_entry)
       SMT ← STPnew
   update_entry(SMT_new_entry)
 return ppn
```

Fig. 2. STP-FTL Address Translation

캐시 미스가 발생한 경우 쓰기 요청은 비어있는 SMT 엔트리에 바로 기록하며 비어있는 SMT가 없는 경우 LRU 알고리즘으로 희생 엔트리를 선택하여 제거한다. 읽기 요청인 경우는 미스가 발생하면 물리 주소를 얻기 위해 GTD를 사용하여 플래시 내에 저장되어 있는 연관된 TP를 읽어와야 한다. 요청에 대응하는 주소 하나만 읽어 올 수 있는 것이 아니라 한 페이지를 읽어 와야 하므로 하나의 TP전체를 읽어오게 되는데, 읽어온 TP에서 하나의 주소만 캐시에 기록하는 것은 공간 지역성을 활용할 수가 없으므로 읽어온 TP를 낭비없이 TPMT에 기록하게 한다.

쓰기 요청을 TPMT에 기록하지 않고 SMT에 기록하는 것은 쓰기 요청은 TP를 읽어오지 않기 때문에 다른 물리 주소를 알 수가 없게 된다. TPMT에 기록하게 되면 TP전체를 기록할 수 있는 엔트리에서 하나의 주소만 기록되기 때문에 낭비되는 비어있는 주소 공간이 커지게 되고, 이는 효과적으로 캐시를 사용할 수 없게 한다. 따라서 상대적으로 낭비되는 공간이 적은 SMT에 기록한다.

캐시 미스가 발생했을 때, 요청된 lpn과 연관된 TP의 일부가 SMT에 기록된 요청일 경우 해당 SMT의 엔트리를 TPMT로 이동시켜 맵핑 정보를 기록하게 한다.

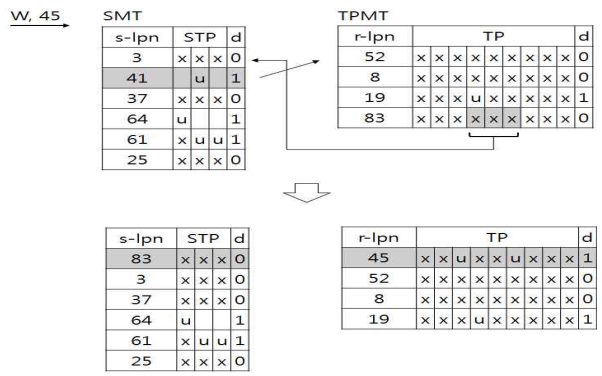

Fig. 3. Exchange of entries between SMT and TPMT

만약 SMT에 기록된 내용이 수정된 경우라면 TPMT에 수정된 내용을 기록하게 하고 수정되지 않은 경우라면 바로 SMT 엔트리에서 제거한다. TPMT가 비어있으면 이동만 하면 되지만 가득 찬 경우 그림 3과 같이 SMT와 TPMT간 엔트리 교환이 발생하게 되고 이동하는 TPMT의 엔트리가 수정된 상태가 아니므로 바로 이동하게 된다. 수정된 상태라면 플래시에 기록하고 GTD를 수정한다.

연관이 없는 요청인 경우에는 SMT에서 LRU 알고리즘으로 희생 엔트리를 선택하여 제거한다. 읽기 요청에 의해 제거될 경우, 읽기 요청은 TPMT에 추가되어야 하므로 SMT의 엔트리 하나를 제거하고, TPMT의 희생 엔트리의 일부가 SMT로 이동하게 된다. 이때 사용되는 것이 r-lpn으로 해당 값의 위치에서부터 STP 크기 만큼의 부분을 SMT에 기록하며 해당 엔트리의 s-lpn의 값이 된다. 만약 해당 TPMT 엔트리가 수정된 상태가 아니라면 바로 제거할 수 있지만, 수정된 상태라면 플래시에 기록한 후에 제거하게 된다.

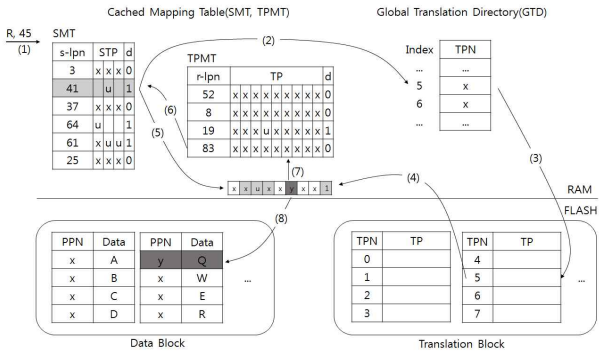

Fig. 4. Address Translation in STP-FTL

그림 4는 읽기 요청이 들어온 경우의 예를 보여준다. 각각의 TP에 기록될 수 있는 주소의 수가 8개이고 SMT에 엔트리당 주소 수가 3일 때 LPN 45에 대한 읽기 요청이 오면 다음과 같이 처리된다. 캐시 맵핑 테이블을 탐색하고 캐시 미스가 발생하고 SMT에 연관된 주소가 있는 것을 알 수 있다. LPN 45에 대한 PPN값을 얻기위해 TP를 읽어와야 하고 $\lfloor 45/8 \rfloor = 5$이므로 GTD의 인덱스 5에 해당하는 TPN에 기록되어 있는 TP를 읽어오게 된다. SMT에 연관된 엔트리가 있으므로 읽어온 TP에 SMT의 값을 갱신한다. TPMT가 가득 찬 상태이고 SMT에 연관된 주소가 있으므로 엔트리 교환 처리과정을 진행하고 TPMT에 SMT의 내용이 추가된 정보를 기록한다.

각각의 맵핑 테이블은 LRU 알고리즘 정책으로 관리되며 TPMT와 SMT간 이동을 통해 하나의 엔트리는 하나의 TP와 연관을 맺게 된다. 따라서 제거 과정에서 같은 TP에 기록되는 엔트리를 찾아야 하는 과정이 필요 없고 수정되지 않은 엔트리는 바로 제거할 수 있게 된다.

IV. Performance Evaluation

제안된 STP의 성능을 평가하기 위해 하나의 주소를 맵핑하는 DFTL과 TP를 맵핑하는 TPM을 캐시의 주소 맵핑에 관련된 부분을 시뮬레이터로 구현하여 비교하였다. 사용된 트레이스 파일은 UMass 트레이스 파일의 Financial 1, 2 [8]와 Web Search 1[9]을 사용하였으며 특성은 표 1과 같다. Financial1은 쓰기 요청 비율 다른 트레이스 파일과 비교했을때 높으며, Financial 2는 쓰기와 읽기가 섞여 있다. WebSearch 1은 대부분 읽기로 이루어져있으며 요청되는 크기가 다른 두 트레이스 파일보다 크다.

Table 1. Characteristics of Traces

Traces	Write Request Ratio	Average Requset Size	Number of Request
Financial1	76.8%	3.38KB	5,334,984
Financial2	17.7%	2.39KB	3,699,194
WebSearch1	0.02%	15.15KB	1,055,448

요구 기반의 페이지 맵핑에서 캐시 미스가 발생하는 경우 요청된 논리 주소에 해당하는 물리 주소를 얻어오거나 캐시 관리를 위해서 플래시에 추가적인 요청이 발생한다. 따라서 캐시 적중률은 응답시간에 영향을 주기때문에 성능 측정에서 중요한 요소가 된다.

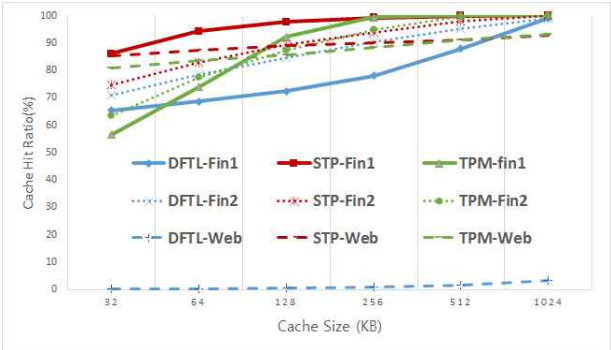

Fig. 5. Cache Hit Ratio under Different Cache Size

그림 5는 한 페이지의 크기가 4KB인 경우 캐시에 요청된 주소를 맵핑하는 DFTL[2]과 TP를 맵핑하는 TPM[4]을 제안한 STP-FTL과 각각의 트레이스를 적용했을 때 캐시 사이즈에 따른 적중률을 보여준다. 제안한 방식의 경우 D는 8로, SMT와 TPMT는 전체 캐시의 크기를 4:6 비율로 설정한 경우이다. 쓰기 요청이 많은 Financial 1에서 TP와 분할된 TP를 함께 사용하는 STP-FTL의 경우 TP만 사용하는 TPM과 비교했을때 캐시 사이즈가 작을수록 더 좋은 캐시 적중률을 보이고 32KB의 캐시 사이즈에서 약 30%더 높은 적중률을 보여준다. 쓰기 연산이 적은 Financial 2와 WebSearch 1의 경우는 Financial 1에 비해 차이가 줄어들긴 하지만 제안한 STP-FTL이 다른 맵핑 방식보다 더 좋은 성능을 보여준다.

TP만 맵핑하는 TPM은 캐시 사이즈가 작은 경우 하나의 주소를 맵핑하는 것보다 더 낮은 캐시 적중률을 보이는 경우도 있는데, 이것은 캐시에서 시간지역성을 이용하기에 충분하지 못한 맵핑 테이블의 엔트리 개수를 가지게 되기 때문이다. 하지만 맵핑 테이블의 엔트리 개수가 늘어나면 주소 하나를 맵핑하는 방식보다 TP를 맵핑하는 경우가 더 좋은 성능을 나타낸다. 일정 개수의 맵핑 테이블 엔트리를 확보할 수 있게 되면 TP를 맵핑하는 것이 공간지역성을 잘 활용할 수 있기 때문에 더 좋은 성능을 보일 수 있다. 하지만 페이지의 크기가 커지면 더 많은 주소가 TP에 기록되고 TP를 맵핑하기 위해 필요한 공간이 늘어나면서 엔트리 개수가 줄어들기 때문에 페이지의 크기가 커질수록 성능은 떨어지게 된다.

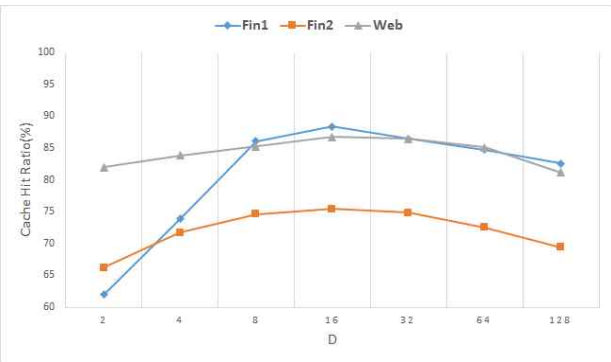

Fig. 6. Effect of D size on STP-FTL

그림 6은 TP의 일정부분을 기록하는 SMT의 한 엔트리에 대한 페이지 맵핑 개수를 결정하는 D 값에 따른 비교를 보여준다. D값이 너무 낮으면 맵핑 테이블의 엔트리의 수가 줄어들게 되고 낮은 캐시 적중률을 보이게 된다. 반면 값이 높으면 엔트리의 수는 많아지지만 연속적인 요청에 대한 공간지역성 효율이 낮아지면서 성능이 떨어지게 된다.

특정 트레이스나 캐시 크기에 따라 차이를 보이지만 대부분의 경우 실험적인 평가로 D값은 8이나 16일때 좋은 성능을 보여준다. 이와 비슷하게 SMT와 TPMT의 비율의 경우 낮은 캐시를 사용하는 경우 SMT의 비율이 높을수록 좋은 성능을 보이지만 캐시의 크기가 커질수록 TPMT의 비율이 높은 경우가 더 좋은 성능을 보이게 된다. 실험결과 D값이 8이고 SMT와 TPMT는 전체 캐시의 크기를 4:6 비율로 가진 경우 대부분의 상황에서 좋은 성능을 보이는 것을 확인할 수 있었다.

그림 7은 한 페이지의 크기가 4KB 이고 캐시 크기가 128KB일때 요청에 대한 캐시 맵핑 테이블의 관리에서 발생하는 읽기, 쓰기에 대한 추가 시간의 평균을 정규화한 그래프이다. 맵핑 테이블의 각 엔트리에 주소하나를 기록하는 경우 TP를 기록하는 것에 비해 낮은 캐시 적중률을 보이기 때문에 발생하는 추가 시간이 커지게 된다. TP만 사용하는 경우와 비교했을 때 STP-FTL의 SMT, TPMT 맵핑 테이블 구조를 사용하는 것과 제안한 맵핑 테이블 관리 기법은 쓰기가 많은 Financial 1의 경우 약 72% 낮은 추가 시간 오버헤드가 발생하고 Financial 2와 WebSearch 1에서 각각 12%, 24% 낮은 추가 시간이 발생한다.

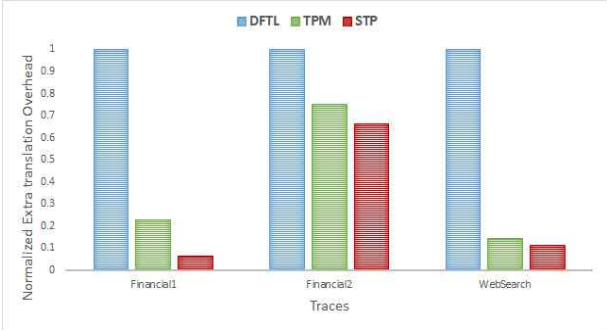

Fig. 7. The Normalized Average Time of Extra Address Translation Overhead

제안한 방법은 랜덤 쓰기 요청이 많을 수록 TP기록에 비해 그 효과가 커지며 좋은 성능을 보이기 위해 필요한 캐시의 크기가 낮다. 또한 다양한 트레이스에 대해서 대체적으로 좋은 성능을 보이는 TP 분할 수인 D 값과 SMT, TPMT의 비율을 실험을 통해 확인하였다. 만약 특수한 트레이스 파일에 대해서는 D값과 SMT, TPMT의 비율을 조절함으로써 성능을 향상시킬 수 있을 것이다.

IV. Conclusions

본 논문에서는 낮은 캐시에서 성능을 높이기 위해 TP를 맵핑하는 TPMT와 TP의 일부분을 맵핑하는 SMT를 사용하는 캐시 구조 및 관리 방법에 대해서 기술하였다. 플래시의 용량이 늘어나고 한 페이지의 크기가 커지게 되면서 맵핑 테이블에 TP를 기록할때의 장점을 유지하면서 그에 따라 발생하는 문제점에 대한 해결책을 제시하였다. 공간 오버헤드 또한 기존에 사용하는 논리주소와 물리주소 쌍으로 구성된 요구 기반 주소 맵핑 방식의 RAM 크기보다 적으며 캐시의 크기가 작아질 때의 성능을 개선하였다.

STP-FTL에서 사용되는 D, SMT와 TPMT의 비율과 같은 특성 값들을 실험을 통해 D는 8, 16 정도가 적절하며 RAM의 용량이 작을 수록 D의 크기가 큰 경우 더 좋은 성능을 보이지만, 너무 크게 할 경우 오히려 성능이 떨어지게 됨을 확인하였다. 따라서 엔트리에 주소 하나를 맵핑하는 방식보다 적절한 크기의 연속된 주소를 함께 맵핑하는 방법이 더 효율적임을 확인할 수 있다. 전체 캐시의 크기에서 SMT와 TPMT의 비율은 4:6 정도가 적절하며 전체 엔트리의 수가 시간지역성을 활용하기에 충분하게 확보된 상태라면 TPMT의 비율이 높아질수록 더 좋은 성능을 보이는 것을 확인하였다. 특수한 경우 트레이스의 성질에 맞게 값을 조절하면 성능을 높일 수 있을 것이다.

REFERENCES

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

[2] GUPTA, A., KIM, Y., AND URGAONKAR, B, “DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings,” In Proceedings of ASPLOS’09, pp. 229-240, March 2009.

[3] Zhiwei Qin , Yi Wang , Duo Liu , Zili Shao, "A Two-Level Caching Mechanism for Demand-Based Page-Level Address Mapping in NAND Flash Memory Storage Systems," Proceedings of the 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium, p.157-166, April 11-14, 2011.

[4] Qi Zhang, Xuandong Li, Linzhang Wang, Tian Zhang, Yi Wang, and Zili Shao, "Optimizing translation information management in NAND flash memory storage systems," In Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC'13). pp. 326—331, 2013.

[5] "Yet another flash file system", http://yaffs.net

[6] A. B. Bityutskiy, "JFFS3 design issues". http://www.linux-mtd.infradead.org

[7] 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.

[8] OLTP Trace from UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

[9] Websearch Trace from Umass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

[10] Z. Qin. Y. Wang, D. Liu, Z. Shao, and Y. Guan, "MNFTL: an efficient flash translation layer for MLC NAND flash memory storage systems," In DAC '11, pp. 17-22, 2011.

[11] You Zhou , Fei Wu , Ping Huang , Xubin He , Changsheng Xie , Jian Zhou, "An efficient page-level FTL to optimize address translation in flash memory," Proceedings of the Tenth European Conference on Computer Systems, Article No. 12, Bordeaux, France, April 21-24, 2015.

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.