논문 2019-14-09

대용량 SSD를 위한 요구 기반 FTL 캐시 분리 기법

(Demand-based FTL Cache Partitioning for Large Capacity SSDs)

배 진 욱, 김 한 별, 임 준 수, 이 성 진*

(Jinwook Bae, Hanbyeol Kim, Junsu Im, Sungjin Lee)

Abstract : As the capacity of SSDs rapidly increases, the amount of DRAM to keep a mapping table size in SSDs becomes very huge. To address a Demand-based FTL (DFTL) scheme that caches part of mapping entries in DRAM is considered to be a feasible alternative. However, owing to its unpredictable behaviors, DFTL fails to provide consistent I/O response times. In this paper, we a) analyze a root cause that results in fluctuation on read latency and b) propose a new demand-based FTL scheme that ensures guaranteed read response time with low write amplification. By preventing mapping evictions while serving reads, the proposed technique guarantees every host read requests to be done in 2 NAND read operations. Moreover, only with 25% of a cache ratio, the proposed scheme improves random write performance and random mixed performance by 1.65x and 1.15x, respectively, over the traditional DFTL.

Keywords : DFTL, Address translation, Cache partitioning, Tail latency, Write amplification

Ⅰ. 서 론

높은 I/O 처리량과 빠른 응답시간 등의 장점을 바탕으로 SSD는 기존의 HDD를 빠르게 대체하고 있다. 또한 3D 낸드 플래시와 같이 플래시 공정 기술이 발전하면서 [1] SSD 용량도 빠른 속도로 커지는 중이다. 하지만 HDD와 다르게, SSD는 용량이 커지면 논리 주소를 물리 주소로 변환해야하는 SSD 컨트롤러의 메모리 요구량도 함께 증가한다. 예를 들어, 현재 시판되는 기업용 SSD의 경우 2.5인치 폼팩터에 30TB의 용량까지 지원하는데 [2], 전체 주소 영역의 변환 테이블을 메모리에 올리기 위해서는 30GB의 DRAM이 필요하다. 하지만 SSD와 같은 제한된 폼팩터의 임베디드 시스템에 대용량의 DRAM을 적재하는 것은 물리적으로 한계가 있다. 따라서 전체 테이블은 SSD에 저장하는 동시에, 자주 사용되는 주소 사상 관계를 메모리에 유지하는 요구 기반 FTL (Demand-based FTL, DFTL) [3]이 새로운 대안으로 주목 받고 있다.

현재 많은 응용 프로그램에서는 사용자의 체감 성능을 높이기 위해, 일정 수준 이상의 성능과 응답 시간을 보장하도록 제한을 두고 서비스한다. 이것은 서비스 수준의 협의 (Service Level Agreement, SLA)라고 일컬어지는데, 클라우드 컴퓨팅이나 가상화 서비스 등 데이터센터 수준의 시스템에서 가장 중요한 요구 사항 중 하나다 [4, 5]. 아쉽게도, 요구 기반 FTL은 접근되는 주소 사상 정보가 DRAM에 캐싱되어 있지 않을 경우 사용자 요청의 응답시간이 매우 느려지는 문제를 발생시킨다. 아울러 워크로드의 지역성에 따라 매우 상이한 성능을 제공하기 때문에 일관된 성능을 보장받기 힘들다는 단점을 지닌다.

본 논문은 요구 기반 FTL 동작의 정량적인 분석을 통해 읽기 요청 수행 시 발생하는 긴 응답 시간 문제의 원인을 찾았으며, 해당 문제를 해결하기 위한 캐시 분리 기법을 제안한다. 제안된 캐시 분리 기법은 기존 요구 기반 FTL의 주소 사상 영역을 읽기와 쓰기를 위한 영역으로 나눈다. 분리된 영역을 통해 읽기 요청의 동작 과정을 낸드 플래시 쓰

*Corresponding Author (sungjin.lee@dgist.ac.kr)

Received: Feb. 24, 2019, Revised: Mar. 28, 2019, Accepted: Apr. 03, 2019.

J. Bae, H. Kim, J. Im, S. Lee: DGIST.

※ 이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (NRF-2017R1E1A1A01077410).

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2019.14.2.71

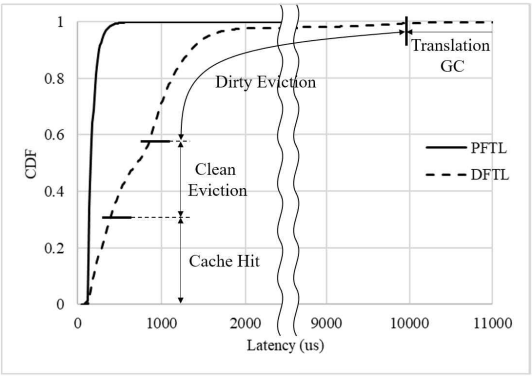


그림 1. 요구 기반 FTL 읽기 응답시간 분석
Fig. 1 DFTL read latency analysis

기연산 없이 읽기연산 2번으로 제한하여 응답시간을 보장한다. 이를 통해 쓰기 성능을 미량 희생시키면서, 기존 요구 기반 FTL의 늘어지는 꼬리 응답시간 문제를 해결한다. 더불어 요구 기반 FTL에서 선택될 수 있는 두 가지 캐시 형태의 장단점을 분석하면서, 쓰기 성능을 함께 최적화하였다.

본 논문에서는 해당 기법의 성능을 실제 SSD로 설계한 임베디드 시스템에서 측정하여 페이지 수준 FTL (Page-level FTL) 및 기존 요구 기반 FTL과 성능을 비교하였다. 64GB 용량의 전체 사상 영역 중에서 25%를 메모리로 캐싱하여 실험한 결과, 읽기 응답시간은 짧게 유지하면서 요구 기반 FTL 대비 임의 쓰기 성능을 1.65배, 임의 읽기/쓰기 성능을 1.15배 향상시켰다.

본 논문의 구성은 다음과 같다. 2장에서는 요구 기반 FTL의 꼬리 응답시간이 늘어나는 원인을 분석함과 동시에 캐싱 방식에 대해 다루고, 3장에서는 우리가 제안하는 캐시 분리 기법을 설명한다. 4장에서는 이 기법의 성능을 측정한 결과를 보여주고, 5장에서는 논문을 마무리한다.

II. 연구 동기

1. 요구 기반 FTL 읽기 응답시간

요구 기반 FTL은 메모리 환경에서 가상 주소를 물리 주소로 빠르게 변환하기 위해 사용하는 TLB (Translation Look-aside Buffer)의 동작 방식으로부터 차용되었다. 기본적으로 자주 사용되는 주소 사상 관계는 CMT (Cached Mapping Table)라는 테이블 형태로 메모리에 유지하면서 빠르게 주소 변환에 사용한다. 메모리에 올라오지 않은 주소에 대한 사용자 요청이 들어오면 다음과 같이 요구 기반 방식으로 진행된다. 먼저 1) 각 테이블의 정보를 담고 있는 GTD (Global Table Directory)를 참조한다. 그리고 2) 저장 장치로부터 해당되는 사상 페이지를 읽어 메모리의 주소 변환 테이블을 갱신한다. 주소 변환 관계를 저장하기 위한 메모리 공간이 제한적인 만큼, 새로운 페이지 (혹은 엔트리)를 메모리에 올리기 위해서는 이미 저장된 것을 비워 메모리 공간을 확보해야할 필요가 있다. 이 과정에서 비워지는 페이지가 메모리로 올라온 이후 갱신되었다면 dirty 상태로 간주하고, SSD에 부가적인 쓰기 작업을 요청하여 새롭게 저장해야한다.

이러한 부가적인 쓰기는 쓰기 요청이 많은 워크로드에서 빈번하게 일어날 수밖에 없다. 데이터가 덮어써질 때마다 주소 변환 테이블의 엔트리도 함께 갱신되기 때문이다. 메모리에서 희생되는 페이지들이 계속해서 저장 장치의 새로운 페이지를 할당받으면, NAND 플래시의 특성 상 쓰레기 수집 (Garbage collection) 작업이 필요해진다. 플래시 메모리는 in-place 갱신이 되지 않기 때문에, 덮어쓰기가 일어나면 쓰레기 페이지가 생기게 된다. 쓰레기 페이지들은 쓰레기 수집 과정에서 추려지고, 유효한 페이지는 새로운 블록으로 복사된다. 이러한 쓰레기 수집 과정은 단순한 읽기/쓰기 작업과 비교하여 많은 시간이 소모되기 때문에, 사용자 요청 중에 발생할 경우 성능 및 응답시간에 치명적으로 작용하게 된다.

그림 1은 임의 읽기/쓰기 혼합 워크로드에서 페이지 수준 FTL과 CMT의 25%를 페이지 단위로 캐싱하는 요구 기반 FTL의 읽기 응답시간 차이를 보여준다. 모든 주소 변환 관계를 메모리에 올려 참조하는 페이지 수준 FTL은 빠르고 일관된 응답시간을 보인다. 이는 사용자 요청 처리 중에 사상 페이지를 읽고 쓰는 과정이 발생하지 않기 때문이다. 요구 기반 FTL은 사상 페이지 읽기 (0.8ms)만 하는 경우에는 비교적 빠른 응답시간을 보인다. 하지만 사상 페이지 쓰기 (0.8ms~10ms)와 그로 인한 쓰레기 수집 작업 (10ms~)의 영향으로 사용자 읽기 요청의 응답시간이 크게 증가한다. 결론적으로 이를 통해 요구 기반 FTL의 읽기 응답시간을 개선하기 위해서는, 사상 페이지의 부가적인 쓰기 연산을 방지해야함을 알 수 있다.

2. 캐시 형태

요구 기반 FTL에서 주소 변환 테이블을 메모리에 유지하는 방법은 크게 1) 엔트리 단위 저장 방

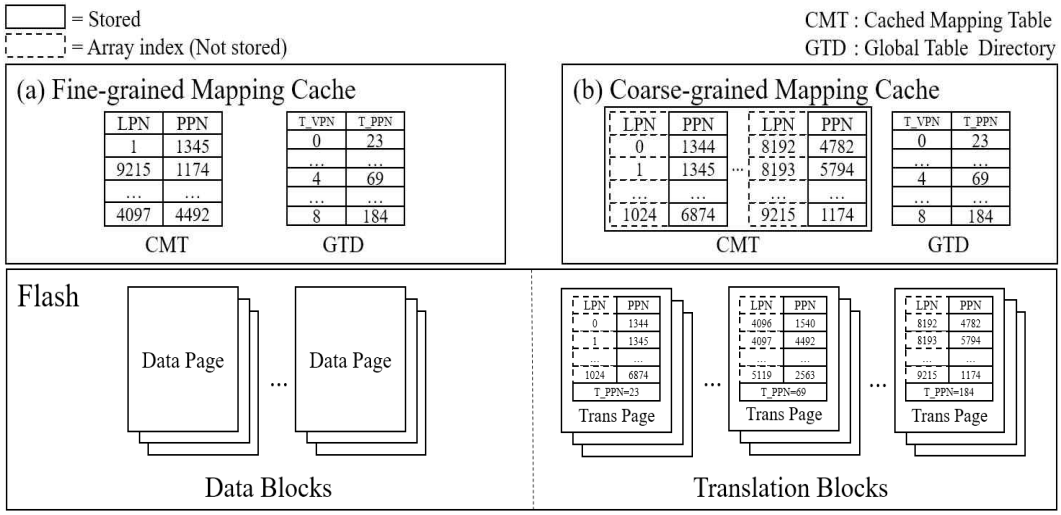


그림 2. 요구 기반 FTL의 두 가지 캐시 형태
(a) 엔트리 단위 저장 방식, (b) 페이지 단위 저장 방식
Fig. 2 Two types of mapping cache on DFTL
(a) Fine-grained mapping cache, (b) Coarse-grained mapping cache

식 (Fine-grained mapping cache)과 2) 페이지 단위 저장 방식 (Coarse-grained mapping cache)으로 나눌 수 있다. 기존 요구 기반 FTL의 경우 캐시 적중에 실패하면 필요한 사상 페이지를 읽은 후 해당되는 엔트리만 메모리에 저장하도록 제안되었다 [3]. 하지만 이후 공간적 지역성과 SSD 컨트롤러의 읽기 단위를 온전히 활용하기 위해 페이지 단위로 저장하도록 발전되었다. 예를 들어, [6, 7]에서는 엔트리 단위의 CMT를 그대로 유지하면서 읽어 올린 사상 페이지를 함께 메모리에 올렸으며, [8, 9]에서는 CMT 자체를 페이지 단위로 구성하여 공간적 지역성을 최대한 활용하고자 하였다.

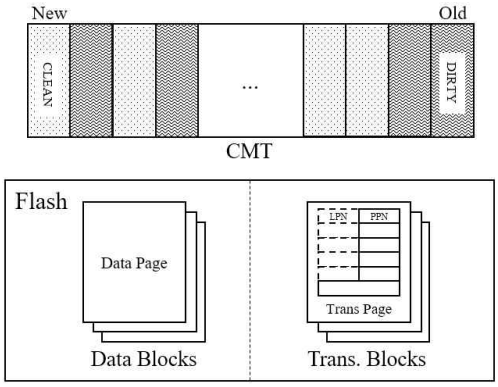
엔트리 단위 저장 방식: 그림 2 (a)에서 볼 수 있듯 엔트리 단위로 CMT를 구성할 경우 (LPA, PPA) 쌍을 함께 기록해야한다. 그리고 만약 엔트리 단위로 LRU 교체 정책을 사용할 경우 캐싱되는 엔트리 개수만큼 포인터 오버헤드가 발생한다. 이 방식은 시간적 지역성은 충분히 활용하지만, 공간적 지역성 측면에서는 제한점이 있다. 이런 형태의 캐시에서 공간적 지역성을 활용하기 위해서는, 한 번에 여러 엔트리를 미리 읽기 (prefetch)해야 한다.

페이지 단위 저장 방식: 그림 2 (b)처럼 페이지 단위로 메모리에 적재시킬 경우, 하나의 사상 페이지가 배열 형태 그대로 올라오게 된다. 따라서 LPA를 배열 인덱스로 사용 가능하기 때문에 앞선 방식보다 2배 더 많은 엔트리를 저장할 수 있다. LRU 또한 페이지 단위로 이루어지기 때문에 LRU 포인터를 유지하는 오버헤드를 줄일 수 있다는 장점을 가진다. 그리고 메모리 저장의 최소 단위가 페이지이므로 공간적 지역성과 SSD 읽기 단위를 쉽게 활용 가능하다.

이 방식의 경우 공간적 지역성을 가진 워크로드에서 큰 성능 향상을 보여준다. 하지만 DRAM의 크기가 지역성을 충분히 커버하지 못하는 경우, 엔트리 단위 저장 방식에 비해 빈번한 사상 페이지 쓰고/읽기 연산이 발생한다. 따라서 임의 트래픽의 워크로드에서 충분한 성능을 내지 못하며 응답시간의 방해가 심해지는 단점이 존재한다.

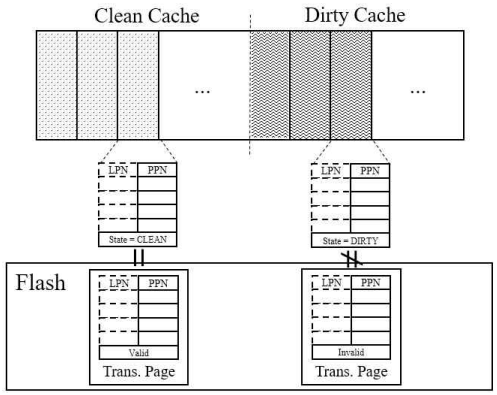
Ⅲ. 캐시 분리 기법 설계

본 논문에서는 제한된 메모리 상황에서도, 늘어지는 읽기 응답시간을 줄이기 위해 캐시를 나누어 독립적으로 관리하는 캐시 분리 기법을 제안한다. 이는 그림 2 (b)와 같이 페이지 단위의 CMT 구조를 기반으로 고안되었다. 이후 페이지 단위 저장 방식의 쓰기 요청에 대한 불필요한 메모리 등재가 존재한다는 맹점을 지적하면서 쓰기 성능을 최적화한 방식에 대해서 설명한다.



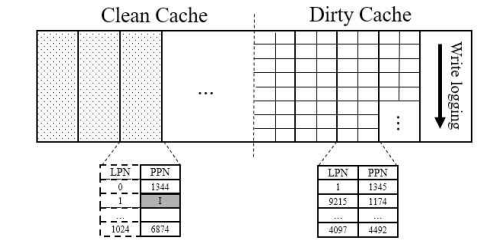
(a) 페이지 단위 저장 방식의 요구 기반 FTL

(a) Coarse-grained DFTL cache



(b) 캐시 분리 기법

(b) Cache partitioning



(c) 캐시 분리 기법 쓰기 성능 최적화

(c) Cache partitioning w / Write optimization

그림 3. DFTL 캐시 분리 기법의 구조

Fig. 3 DFTL cache partitioning architecture

1. 캐시 분리 기법 구조

제안된 캐시 분리 기법은 그림 3 (a)와 같은 요구 기반 FTL의 CMT를 그림 3 (b)와 같이 clean과 dirty의 두 영역으로 나누어 관리한다. SSD 내부 메모리에서 CMT로 사용되는 영역을 논리적으로 나누어, clean 영역에서는 clean 상태의 사상 페이지만, dirty 영역에서는 dirty 상태의 사상 페이지만 저장한다. 그리고 저장된 사상 페이지들은 각 영역에서 LRU 교체 정책을 통해 관리된다. 그림 3 (b)는 기본적인 캐시 분리 기법을 개략적으로 보여준다.

읽기 요청: 읽기 요청이 SSD로 들어오면 먼저 캐시를 살펴본다. 관련된 사상 페이지가 dirty 영역에 저장되어 있더라도 참조는 가능하다. 캐시 참조를 통해 주소 변환 관계를 알 수 있다면, 해당 물리 주소에 저장된 데이터를 읽는다. 하지만 캐시 적중에 실패할 경우, 즉 clean 및 dirty 영역 모두에서 참조를 실패했다면 clean 영역 내의 LRU 알고리즘을 통해 참조된 지 가장 오래된 클린 페이지를 쫓아낸다. 이렇게 하면 읽기 요청 중에 추가적인 쓰기 작업이 발생하지 않는 것을 쉽게 보장할 수 있다. 따라서 해당 기법을 사용할 때, 읽기 요청은 최악의 경우 캐시 적중 실패로 인한 사상 페이지 읽기와 데이터 읽기로 총 2번의 읽기면 완료된다.

쓰기 요청: 반면에 쓰기 요청 중에는 dirty 영역에서 페이지를 비워주게 되기 때문에, 캐시 적중에 실패할 경우 사상 페이지의 쓰기가 항상 동반하게 된다. 이는 기존 요구 기반 FTL에서 읽기 요청 중에 발생하던 추가 쓰기 작업들을, 쓰기 요청 중에 발생하도록 미룬 것과 같다. 그리고 캐시에서 참조에 성공하는 경우, 두 가지 시나리오가 발생한다.

먼저 dirty 영역에서 적중된 경우 새롭게 쓰이는 물리 주소로 엔트리를 갱신한다. dirty 영역에 존재하는 사상 페이지들은 모두 dirty 상태이기 때문에 이 작업 중에 사상 페이지의 상태 변화는 없다. 그리고 해당 페이지를 dirty 영역에서 자체적으로 관리하는 LRU 알고리즘에서 갱신해준다.

반면에 이미 clean 영역로 올라온 사상 페이지에서 캐시 적중이 될 경우에는 추가적인 연산이 발생한다. 쓰기 요청이 발생하면 주소 변환 테이블의 엔트리도 함께 갱신되기 때문에, 쓰기 요청에 한번이라도 적중된 사상 페이지는 dirty 상태로 전환되어 버린다. 그러므로 해당 페이지는 더 이상 clean 영역에 머무를 수 없게 된다. 이 때 캐시 영역을 제한하지 않고 단순히 dirty 영역으로 이전시키는 것은, 모든 캐시 영역이 dirty 상태의 사상 페이지로 채워질 수 있기 때문에 문제가 된다. 이 경우 다음에 들어오는 읽기 요청에서 캐시 적중에 실패하면 쓰기 작업이 수행되고, 곧바로 응답시간에 영향을 미치게 된다. 이는 곧 clean 영역의 공간을 어느 정도 보장해야 한다는 것을 의미한다. 그래서 우리는 clean 영역에서 dirty 영역으로 이전될 때, dirty 영역이 가득 차 있다면 해당 캐시 영역의 희생 페이

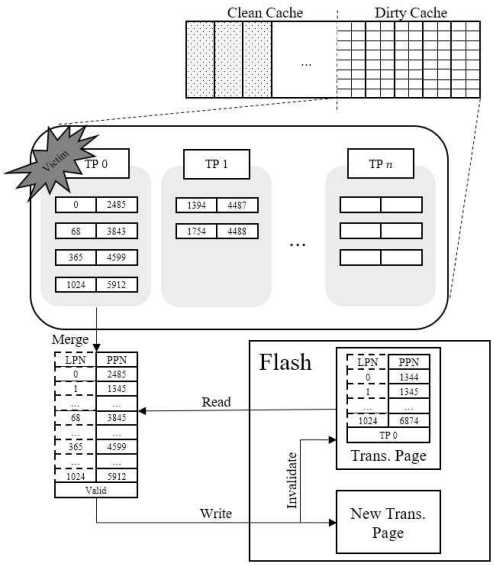


그림 4. Batch-update 동작 과정
Fig. 4 Batch-update process

지를 SSD에 써서 각 영역의 공간을 확보하게끔 하였다.

각 영역의 크기는 읽기나 쓰기 등 목적에 따라 다르게 설정할 수 있지만, 본 논문에서는 50:50의 비율로 설정하여 기법의 효과를 증명하였다 [10].

2. 쓰기 성능 최적화

그림 3 (a)나 (b)와 같이 페이지 단위로 CMT를 구성하면, 임의 트래픽의 워크로드에서 부가적인 쓰기 연산이 많아지고 쓰기 성능이 저하되는 문제가 발생한다. 이는 4절 성능 평가에서도 볼 수 있듯 요구 기반 FTL은 물론이고, 캐시 분리 기법이 적용된 상태에서 더 뚜렷하게 나타난다. 때문에 우리는 기존의 캐시 분리 기법에서 dirty 영역을 엔트리 단위로 이어 쓰도록 하여 쓰기 성능을 최적화하였다. 읽기 요청은 기존 구조와 같기 때문에 이전 흐름을 그대로 따라가지만, 쓰기 요청은 동작 방식이 바뀌게 된다.

쓰기 요청: 쓰기 요청은 발생할 때마다 CMT의 엔트리를 갱신해야 한다. 페이지 단위 저장 방식에서는 한 엔트리를 갱신하기 위해 전체 사상 페이지를 함께 메모리에 올려야했다. 이는 임의 트래픽의 쓰기 워크로드에서 매우 비효율적으로 동작한다. 반면 쓰기 요청에서 갱신하는 엔트리는 사상 페이지를 메모리에 올리지 않아도 변경 사항을 기록할 수 있다. 그림 3 (b)와 같이 dirty 영역에 엔트리 단위로 이어 쓰게 (Logging) 되면 불필요한 쓰기 작업을 없앨 수 있다.

Batch-update: 부족한 메모리를 위해 엔트리를 비워야하는 상황이 되면, 엔트리 단위 저장 방식에서는 batch-update를 통해 희생 엔트리와 같은 사상 페이지의 엔트리들을 함께 써야 한다. 만약 희생 엔트리 1개만 SSD에 써줄 경우 부가적인 쓰기 연산이 매우 빈번하게 발생하기 때문이다. 본 기법에서는 그림 4와 같이 dirty 영역의 엔트리들을 사상 페이지 별로 관리하고, 쫓아내는 상황이 되면 희생 페이지를 정해서 batch-update하였다.

이 경우 희생 페이지를 정하는 데에 있어서 다양한 교체 정책이 사용될 수 있다. 예를 들어, 페이지 단위로 LRU 알고리즘을 사용하거나, dirty 상태의 엔트리가 가장 많은 페이지를 선택하는 탐욕 알고리즘을 사용할 수도 있다. 본 논문에서는 탐욕 알고리즘 방식을 채택하여, batch-update 효과를 최대한 활용하였다.

Ⅳ. 성능 평가

1. 실험 환경

본 기법의 성능 평가는 Xilinx ZCU102 보드에서 구현된 환경을 통해 수행되었다. 이는 최대 512GB 용량의 SSD 역할을 하는 임베디드 보드로, 4GB의 DRAM과 1GHz ARM 코어 4개로 구성되어 있다. 우리는 해당 임베디드 보드 위에서 FTL 플랫폼을 개발하였으며, 플랫폼 자체 벤치마크 툴을 사용하여 성능을 측정하였다. 벤치마크 툴에서 생성하는 워크로드는 균등분포 (Uniform distribution)를 가지는 임의 읽기 및 쓰기이며, 두 요청을 함께 섞어서 생성하는 혼합 (Mixed) 워크로드도 존재한다. 혼합 워크로드의 성능 측정은 쓰레기 수집 과정이 반영된 경우와 반대의 경우를 함께 관찰하였다. 실험에 사용된 데이터 셋의 크기는 64GB이며 CMT의 용량은 데이터 셋의 25%를 캐싱할 수 있도록 설정하였다. 제안된 캐시 분리 기법의 경우 clean과 dirty 영역을 간단하게 각각 12.5%씩 설정하여, 본 기법의 효용성을 보였다. 실험에서는 캐시 분리 기법을 페이지 수준 FTL 및 요구 기반 FTL과 비교하면서, 읽기/쓰기 성능과 읽기 응답시간에 초점을 두었다.

2. 실험 결과

읽기/쓰기 성능: 그림 5는 균등분포의 임의 읽기/쓰기 성능을 각 기법에 대해 비교한 그래프이다. 임의 읽기나 쓰기 성능은 해당되는 요청을 충분한

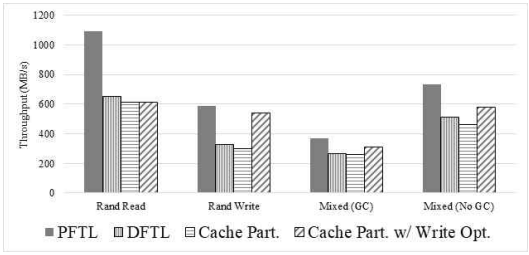


그림 5. 읽기/쓰기 성능
Fig. 5 Read/Write throughput

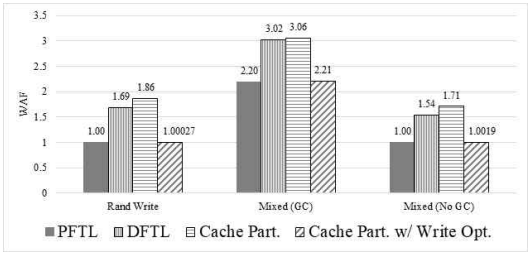


그림 6. 쓰기 증폭 인자
Fig. 6 Write amplification factor

범위 (64GB)에 임의로 생성하여 측정되었다. 또한, 혼합 워크로드는 순차 쓰기로 64GB의 데이터를 적은 후에 해당 범위에 대해서 임의로 읽기/쓰기 요청을 혼합 생성하여 성능을 측정하였다.

임의 트래픽의 워크로드에서 요구 기반 FTL 및 캐시 분리 기법은 사상 페이지의 빈번한 읽기/쓰기로 인해 좋은 성능을 내지 못한다. 하지만 쓰기 성능을 최적화시킨 기법은 요구 기반 FTL보다 1.65배 좋으며 페이지 수준 FTL과는 비슷한 임의 쓰기 성능을 보인다. 그리고 임의 읽기/쓰기를 섞은 혼합 워크로드에서도 임의 쓰기 성능을 개선시킨 만큼 요구 기반 FTL 대비 1.15배 높은 성능을 보여준다.

그림 6에서는 각 기법에서 얻은 쓰기 증폭 인자 (Write Amplification Factor, WAF)를 보여준다. 쓰기 증폭 인자는 1번의 쓰기 요청에 대해 평균 몇 번의 낸드 플래시 쓰기연산이 발생하는지 계산한 값을 나타낸다. 그림 5와 마찬가지로 페이지 단위의 저장 방식을 사용하는 두 요구 기반 FTL은, 캐싱 방식의 특성상 높은 쓰기 증폭 인자 값을 보인다. 반면 dirty 영역을 엔트리 단위로 저장하면 페이지 수준 FTL과 거의 동일한 쓰기 증폭 인자 값을 가진다. 다만 사상 페이지의 읽기가 완료돼야 해당 페이지를 새로 쓸 수 있기 때문에, 페이지 수준 FTL에 비해 조금 떨어지는 성능을 보인다.

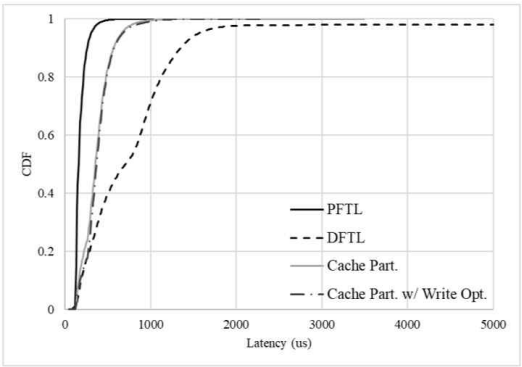


그림 7. 읽기 응답시간 누적분포
Fig. 7 Read latency CDF

읽기 응답시간: 그림 7은 임의 읽기/쓰기가 섞인 혼합 워크로드에서 읽기 요청에 대한 응답시간의 누적 분포 그래프로, 쓰기 요청으로 인해 방해받는 응답 시간을 보여준다. 본 논문에서 제시하는 기법은 페이지 수준 FTL과 비교하면 느리지만 기존 요구 기반 FTL보다는 앞서는 응답시간을 보여준다. 요구 기반 FTL에서 부가적인 쓰기 연산으로 인해 나타났던 꼬리 응답시간 또한 해당 기법에서는 관찰되지 않는다. 오히려 그림 1에서 분석했던 사상 페이지 읽기로 인한 읽기 응답시간 (0.8ms)과 유사한 수치의 꼬리 응답시간을 보인다. 이를 통해 최악의 경우 2번의 읽기 연산으로 읽기 요청이 수행됨을 확인할 수 있다. 또한 쓰기 성능을 최적화시킨 경우에도 캐시 분리 기법은 그대로 유지하였기 때문에 동일하게 늘어지는 꼬리 응답시간을 제거한 모습을 보인다.

V. 결 론

본 논문에서는 요구 기반 FTL의 늘어지는 읽기 꼬리 응답시간을 줄이기 위해 캐시를 두 영역으로 나누어 읽기 응답시간을 보장하는 캐시 분리 기법을 소개했다.

제안된 기법은 요구 기반 FTL의 읽기 응답시간이 느려지는 원인 분석을 바탕으로, 읽기 요청 중 방해받을 수 있는 요소들을 없앴다. 그리고 요구 기반 FTL의 두 가지 테이블 구성 방식의 장점을 함께 취하여 쓰기 성능 또한 최적화시켰다.

그리고 Xilinx ZCU102 임베디드 보드에서 FTL의 성능을 측정하여, 임의 쓰기와 임의 읽기/쓰기 혼합 실험에서 각각 요구 기반 FTL 대비 1.65배,

1.15배 성능이 개선됨을 보였다. 따라서 메모리가 제한적인 상황에서도 읽기 응답시간을 보장하면서 높은 쓰기 성능을 낼 수 있음을 검증하였다.

이후 연구에서는 실험적으로 보인 각 기법의 쓰기 증폭 인자를 수식적으로 정리하여 쓰기 성능의 정량적인 예측에 활용할 예정이다. 또한 본 연구는 실시간으로 워크로드 패턴을 파악하면서, 분리된 캐시 영역의 크기를 동적으로 조절하여 성능을 최대화하도록 확장될 수 있다.

References

[1] Available on : https://www.samsung.com/us/business/oem-solutions/pdfs/V-NAND_technology_WP.pdf

[2] Available on : https://www.samsung.com/semiconductor/ssd/enterprise-ssd/

[3] A. Gupta, Y. Kim, B. Urgaonkar, “DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings,” Journal of ACM, Vol. 44, No. 3, pp. 229-240, 2009.

[4] X. Li, J. Du, “Adaptive and Attribute-based Trust Model for Service-level Agreement Guarantee in Cloud Computing,” Journal of IET Information Security, Vol. 7, No. 1, pp. 39-50, 2013.

[5] D. Serrano, S. Bouchenak, Y. Kouki, T. Ledoux, J. Lejeune, J. Sopena, L. Arates, P. Sens, "Towards Qos-oriented Sla Guarantees for Online Cloud Services," Proceedings of 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, pp. 50-57, 2013.

[6] Z. Qin, Y. Wang, D. Liu, Z. Shao, “A Two-level Caching Mechanism for Demand-based Page-level Address Mapping in NAND Flash Memory Storage Systems,” Proceedings of 17th IEEE Real-Time and Embedded Technology and Application Symposium, pp. 157-166, 2011.

[7] M. Wang, Y. Zhang, W. Kang, “ZFTL: A Zone-based Flash Translation Layer with a Two-tier Selective Caching Mechanism,” Proceedings of 14th IEEE International Conference on Communication Technology, pp. 578-588, 2012.

[8] C. Wang, W. Wong, “TreeFTL: Efficient RAM Management for High Performance of NAND Flash-based Storage Systems,” Proceedings of the Conference on Design, Automation and Test in Europe, pp. 374-379, 2013.

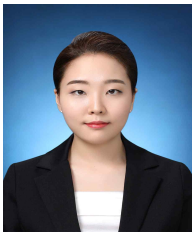
[9] S. Jiang, L. Zhang, X. Yuan, H. Hu, Y. Chen, “S-FTL: An Efficient Address Translation for Flash Memory by Exploiting Spatial Locality,” Proceedings of 27th IEEE Symposium on Mass Storage Systems and Technologies, pp. 1-12, 2011.

[10] H.B. Kim, J.W. Bae, J.S. Im, S.J. Lee, “Address Translation with Bounded Tail Latency for Large Capacity SSDs,” Proceedings of Korea Software Congress (KSC), pp. 1481-1483, 2018 (in Korean).

Jinwook Bae (배 진 욱)

He received the B.S. degree in Computer Engineering from Inha University, Korea, in 2018. He is currently a M.S. student in Information and Communication Engineering at DGIST. His research interests include storage systems, embedded system software and data-center architecture.
Email: jinwook.bae@dgist.ac.kr

Hanbyeol Kim (김 한 별)



She received the B.S. degree in Global Finance and Banking from Inha University, Korea, in 2017. She is currently a M.S. student in Information Communication Engineering at DGIST. Her research interests include storage system software, embedded system and operating system.
Email: hbstella92@dgist.ac.kr

Junsu Im (임 준 수)

He received the B.S. degree in Computer Engineering from Inha University, Korea, in 2018. He is currently a Integrated Ph.D. student in Information and Communication Engineering at DGIST. His research interests include storage system, data-center architecture and embedded software.
Email: junsu_im@dgist.ac.kr

Sungjin Lee (이 성 진)

He is an assistant professor at DGIST in South Korea. He earned the PhD and MS degrees in computer science and engineering from the Seoul National University in 2013 and 2007, respectively, and received the BE degree in electrical engineering from the Korea University in 2005. Before joining DGIST, he was a postdoctoral associate in the Computation Structures Group (CSG) at MIT CSAIL in Boston, MA. His current research interests include storage systems, operating systems, and system software.
Email: sungjin.lee@dgist.ac.kr