Journal of The Korea Society of Computer and Information
Vol. 21 No. 2, pp. 1-7, February 2016

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2016.21.2.001

Correlated Locality Data Distribution Policy for Improving Performance in SSD

Jung Kyu Park*

Abstract

In this paper, we propose in this paper present a novel locality data allocation policy as COLD(Correlated Locality Data) allocation policy. COLD is defined as a set of data that will be updated together later. By distributing a COLD into a NAND block separately, it can preserve th locality. In addition, by handling multiple COLD simultaneously, it can obtain the parallelism among NAND chips. We perform two experiment to demonstrate the effectiveness of the COLD data allocation policy. First, we implement COLD detector, and then, analyze a well-known workload. And we confirm the amount of COLD found depending on the size of data constituting the COLD. Secondly, we compared the traditional page-level mapping policy and COLD for garbage collection overhead in actual development board Cosmos OpenSSD. Experimental results have shown that COLD data allocation policy is significantly reduces the garbage collection overhead. Also, we confirmed that garbage collection overhead vary depending on the COLD size.

▸ Keyword : SSD, performance, Flash, NAND

I. Introduction

최근 각광받고 있는 저장매체 중 하나인 플래시 메모리는 전기적으로 데이터에 접근할 수 있는 비휘발성 저장매체로써 빠른 임의 접근 시간과 높은 데이터 처리량, 적은 소모 전력을 가지고 있다는 장점이 있다. 더불어 소형에다가 강한 내구성을 가지고 있어 개인 컴퓨터, 스마트 폰, 태블릿 PC, 노트북 등 다양한 곳에서 폭 넓게 사용되고 있다. 최근 플래시 메모리는 20nm 이하의 공정으로 만들어지고 있으며, 3D NAND 기술과 MLC(Multi-Level Cell) 기술을 도입하여 집적도를 높이고 있다[1]. 이러한 기술 발전으로 인해 512GB 크기의 플래시 칩이 출시되고 있으며, TB 이상 고용량 저장 장치 개발이 가능하게 되어 대용량 서버와 데이터 센터 등의 환경에서도 채택되어 사용되고 있다.

SSD는 플래시 메모리로 이루어진 대표적인 저장 장치로 플래시 메모리의 장점을 모두 가져 기존에 사용되던 하드디스크를 빠르게 대체하고 있다. 그러나 SSD는 플래시 메모리의 하드웨어적인 단점을 가지고 있기 때문에 FTL(Flash Translation Layer)이라는 소프트웨어 계층을 이용하여 데이터 주소 변환, 가비지 컬렉션(garbage collection), 마모 평준화(wear-leveling) 등의 기능을 수행해 이를 극복하고 있다.

가비지 컬렉션은 out-of-place 업데이트로 인해 생겨난 유효하지 않은 데이터(invalid data)들을 정리해주는 기법으로 사용자가 요청한 작업과 별도로 추가적인쓰기와 삭제 연산이 발생한다. 읽기 및 쓰기 연산의 단위가 페이지인 것과 비해 삭제 연산은 훨씬 큰 블록 단위로만 가능하기 때문에 가비지 컬렉션은 상당한 성능 오버헤드를 가지고 있다. 또한 플래시 메모리의 블록은 제한된 수명을 가지고 있어서 삭제 연산이 발생하는 블록은 점점 마모되기 때문에 잦은 삭제 연산은 SSD의 수명을 크게 감소시킬 수 있다. 그러므로 가비지 컬렉션의 오버헤드를 줄이는 일은 SSD의 성능 및 수명을 향상시키기 위해 가장 효과적인 방법이라 할 수 있다.

SSD는 성능 향상 및 용량 확장을 위해 다수의 NAND 칩에 데이터를 분산 저장 시키는 인터리빙(interleaving) 방식을

• First Author: Jung Kyu Park, Corresponding Author: Jung Kyu Park
*Jung Kyu Park(jkpark@dankook.ac.kr), Department of Computer Engineering, Dankook University
• Received: 2016. 01. 06, Revised: 2016. 01. 10, Accepted: 2016. 01. 28.

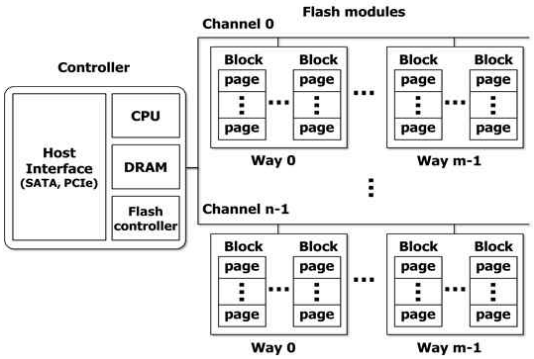

Fig. 1. Internal architecture of SSD

활용하는데 이러한 인터리빙 방식은 지역성을 가지는 데이터들 또한 여러 칩에 분산 저장시킨다. 지역성을 가지는 데이터들은 이 후 업데이트 시 함께 업데이트 될 가능성이 높고 이러한 데이터들이 흩어져 있다면 가비지 컬렉션이 수행 될 때 페이지 이동 및 블록 삭제가 증가하여 가비지 컬렉션의 오버헤드가 커지게 된다.

본 논문에서는 이러한 가비지 컬렉션으로 발생하는 오버헤드를 줄이기 위해 지역성을 가지는 연관 데이터 배치 정책인 COLD(Correlated Locality Data) 할당 정책을 제안한다. 만약 호스트(host)에서 전달된 데이터가 지역성을 가진다면 인터리빙을 사용하여 흩어지게 하지 않고 한 칩에 저장함으로서 이후의 가비지 컬렉션 수행 시 오버헤드를 크게 줄일 수 있다.

제안된 기법은 SSD 개발 보드에 구현하여 기존의 페이지 레벨 맵핑 방식(page level mapping)과 성능을 비교한다.

논문의 구성은 다음과 같다. 2장에서는 플래시 메모리와 SSD에 대해 관련 연구를 살펴보고 3장에서는 COLD의 아이디어에 대해 설명한다. 다음 4장에서는 COLD의 설계 및 구현 방법에 대해서 알아본다. 5장에서는 실험결과를 설명하고 마지막 6장에서 결론을 맺는다.

II. Related Works

1. FTL (Flash Translation Layer)

NAND 플래시 메모리의 하드웨어적인 특성 때문에 이를 관리하기 위한 특별한 소프트웨어 계층이 필요한데 그림 3과 같이 호스트 시스템과 NAND 플래시 메모리 사이에서 NAND 플래시 메모리의 하드웨어적인 특성을 관리해주는 소프트웨어 계층을 FTL(Flash Translation Layer)이라고 한다.

FTL은 호스트에서 전달되는 논리적 주소를 NAND 플래시 메모리에 맞게 물리적 주소로 맵핑한다. 이러한 주소 맵핑 방식은 맵핑하는 단위에 따라 페이지 맵핑(page mapping), 블록 맵핑(block mapping), 하이브리드 맵핑(hybrid mapping) 등의 여러 가지 맵핑 방식이 있다[3,4,5].

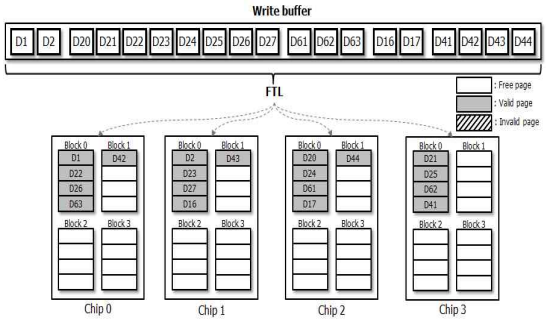

Fig. 2. Data Allocation Method Using Interleaving

또한 NAND 플래시 메모리는 out-of-place 업데이트를 하기 때문에 계속해서 업데이트가 발생하면 유효하지 않은 데이터가 쌓이게 되고 NAND 플래시 메모리내부에 데이터를 쓸 영역이 점차 사라지게 된다. 이런 이유로 FTL은 유효하지 않은 페이지를 다시 사용 가능한 페이지로 바꿔서 새로운 데이터를 쓸 수 있는 빈 공간을 확보하는 작업을 수행 하는데 이러한 작업을 가비지 컬렉션 (garbage collection)이라고 한다. 가비지 컬렉션은 빈 공간을 만들기 위해 삭제 할 블록 (victim block)을 선정하고 해당 블록에 있는 유효한 페이지들은 모두 다른 블록들로 이동 시킨 후 삭제 연산을 수행한다. 이렇듯 가비지 컬렉션이 수행되면 사용자가 요청하는 작업 이외에 추가적인 쓰기/삭제 연산이 수행되기 때문에 가비지 컬렉션의 오버헤드는 SSD의 전체 성능 저하에 상당한 영향을 끼친다.

NAND 플래시 메모리는 제한된 삭제 횟수를 가지고 있기 때문에 만약 블록의 삭제 횟수가 특정 블록들에만 집중되어 베드 블록이 급격하게 증가하게 되면 NAND 플래시 메모리를 더 이상 사용할 수 없게 되는 문제가 발생한다. 이를 해결하기 위해 FTL은 마모 평준화 기법(wear-leveling)을 사용하여 삭제 연산이 모든 블록에 골고루 될 수 있게 하여 모든 블록이 균등하게 사용될 수 있도록 하는 방법이다[6].

2. SSD

SSD(Solid State Drive)는 컨트롤러(controller)와 여러 개의 NAND 플래시 칩으로 구성되어 있고 그림 1에서 이를 자세히 나타내고 있다.

SSD 컨트롤러는 호스트 인터페이스(host interface), 프로세서(processor), DRAM 과 플래시 컨트롤러(Flash controller)로 구성되어 있는데 SSD의 전체적인 동작을 제어하는 역할을 한다. 호스트 인터페이스는 파일 시스템에서 요청한 작업들을 SSD로 전달하는 역할을 하고 프로세서와 DRAM은 FTL을 수행하는데 사용되고 DRAM의 상당 부분은 호스트에서 전달된 쓰기 연산을 위해 데이터를 저장하기 위한 버퍼로 사용된다.

DRAM 버퍼는 SSD의 성능 향상에 중요한 영향을 주는 부분이다. 버퍼는 쓰기 요청을 최대한 지연시켜 플래시 내부에 발생하는 쓰기와 가비지 컬렉션을 줄일 수 있다. 이를 위해 내부 버퍼를 그룹핑하여 관리하는 기법과 버퍼의 교체 기법이 연구되

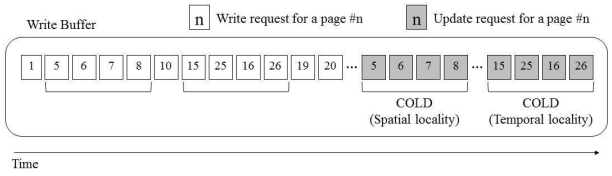

Fig. 3. COLD Detecting Method in Write Buffer

었다[13-15].

NAND 플래시 칩은 플래시 컨트롤러를 통해서 SSD 컨트롤러와 연결되고 멀티-채널(multi-channel), 멀티-웨이(multi-way)구조를 가진다. 따라서 서로 다른 칩으로 배치되는 데이터들은 동시에 접근이 가능하여 I/O 성능이 향상될 수 있는데 이 방법을 채널/웨이 인터리빙(interleaving)이라고 한다[7].

인터리빙은 한 페이지 이상의 데이터를 다수의 채널/웨이에 분산시켜서 저장할 경우 각 요청에 대한 처리를 병렬적으로 수행 가능하므로 SSD의 전체적인 성능을 높일 수 있다. 하지만 만약 지역성을 가지는 페이지들이 인터리빙에 의해 여러 칩에 분산 저장되게 되면 이후 가비지 컬렉션 오버헤드가 상당히 커질 수 있다. 이를 설명하기 위해 그림2는 4개의 채널과 각 채널은 4개의 블록, 각 블록은 4개의 페이지로 이루어진 SSD 내부에서 일반적인 인터리빙을 이용한 데이터 배치를 나타낸 그림이다.

초기에는 NAND 플래시 메모리에 데이터가 쓰이지 않은 상태로 가정했고 만약 호스트에서 데이터 요청이 발생하면 호스트에서 전달된 쓰기 요청들은 SSD 내부의 쓰기 버퍼에 먼저 쓰이게 된다. 그림의 쓰기 버퍼 내부에서 네모 상자들은 페이지 크기의 데이터이고 서로 붙어 있는 네모 상자는 하나의 쓰기 요청을 의미한다. 쓰기 버퍼에 있는 데이터들은 FTL에 의해 NAND 플래시 메모리에 맵핑되는 주소를 할당받고 인터리빙을 통해 각 칩에 병렬적으로 쓰이게 되어 그림과 같이 NAND 플래시 메모리 내부에 배치되게 된다. 예를 들어 D1이 칩1에 쓰이게 되면 D2는 칩2에 쓰이고 이런 식으로 다른 데이터들도 각 칩에 쓰이게 된다.

이렇게 여러 칩에 병렬적으로 데이터를 쓰면 쓰기 작업 수행 시의 SSD 성능을 향상 시킬 수 있다. 하지만 이후에 수정 또는 삭제 연산을 포함하는 가비지 컬렉션이 수행되게 되면 심각한 성능 저하를 야기 시킬 수 있다.

그림 2에서 D20~D27, D41~D44 데이터가 업데이트 요청이 요청되었을 때의 NAND 플래시 메모리의 내부 상태를 가정한다. 이러한 파일의 삭제 또는 업데이트 연산은 워드 파일이나 파워포인트 파일과 같은 파일에서는 매우 빈번하게 발생한다[8]. 그리고 같은 요청에 속해 있는 데이터들은 지역성에 의해 함께 삭제 또는 업데이트 될 가능성이 높다. 이후 NAND 플래시 메모리 내부에 공간이 부족하면 FTL은 가비지 컬렉션을 수행시킨다.

가비지 컬렉션 기법은 유효하지 않은 페이지들을 포함한 블록들 내부의 유효한 페이지들은 빈 영역에 복사되고 블록을 삭제함으로서 공간을 확보한다. 그림 2에서 D20~D27, D41~D44 페이지는 유효하지 않은 페이지가 되고, 공간을 확보하기 위해서는 이 페이지들이 다른 영역에 복사 된 후 삭제 연산이 수행되어야 한다. 그런데 만약 유효하지 않은 페이지들이 여러 칩에 분산되어 있으면 수행해야하는 페이지 복사와 블록 삭제 연산이 많아지기 때문에 상당한 가비지 컬렉션 오버헤드가 발생한다. 그림 2에서 Chip 0의 D1, D63, Chip 2의 D2, D16, Chip 2의 D61, D17, Chip 3의 D62가 유효한 페이지이기 때문에 빈 영역으로 복사되어야 그 후 4개의 블록이 삭제될 수 있다. 이때 가비지 컬렉션 오버헤드는 7개의 페이지 복사 연산과 4개의 블록 삭제 연산이 발생한다.

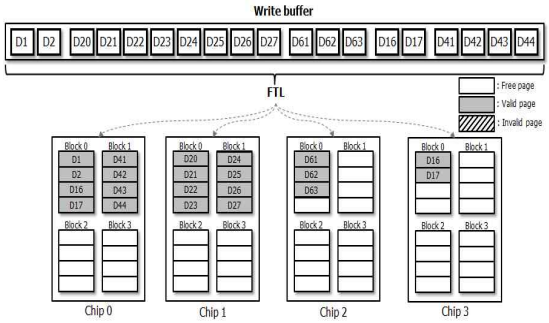

Fig. 4. Example of COLD Allocation in SSD

III. COLD(Correlated Locality Data) 할당 정책

1. 제안 기법

본 논문에서 제안하는 기법은 지역성을 가지는 데이터들이 인터리빙으로 인해 여러 칩에 흩어지지 않게 한 곳에 모아 가비지 컬렉션 수행 시 오버헤드를 최소화 할 수 있도록 한다. 본 논문에서는 지역성을 가지는 데이터들을 COLD(Correlated Locality Data)라 정의하였다. 지역성을 가지는 데이터들은 이후 함께 업데이트 될 가능성이 많기 때문에 COLD를 한 곳에 저장하면 데이터들의 지역성을 유지 할 수 있다.

SSD에 데이터가 전달되면 먼저 DRAM의 IO 버퍼에 데이터들이 저장 되므로 버퍼 내의 데이터들을 분석하여 COLD를 찾을 수 있다. COLD를 찾는 방법은 두 가지로 구분 할 수 있다. 첫 번째로는 공간적 지역성(spatial locality)을 이용하는 방법이고 또 두 번째는 시간적 지역성(temporal locality)을 이용하는 방법이다. 그림 3은 쓰기 버퍼 내에서 COLD를 발견하는 방법을 보여주고 있다.

그림 3과 같이 호스트에서 쓰기 요청을 SSD로 전달하게 되면 쓰기 요청들은 먼저 SSD 내의 DRAM에 쌓이게 된다. 공간적 지역성을 이용하는 방법은 쓰기 버퍼 내에 쓰기 요청들의 논리적 주소가 연속적이고 일정 크기 이상을 가진다면 공간적

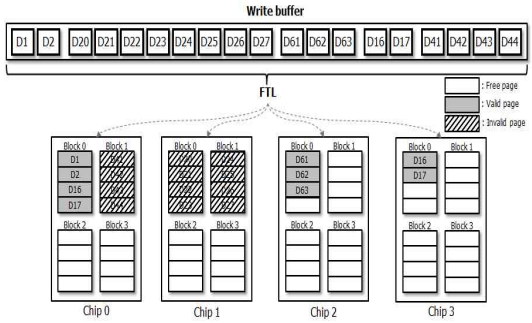

Fig. 5. Data Arrangement Using COLD Allocation

지역성을 가질 것이라 예상하고 COLD로 정의 한다. COLD로 정의된 요청은 NAND 플래시 메모리에 할당될 때 인터리빙으로 흩어지지 않고 한 칩에 모두 배치시킨다. 시간적 지역성을 이용하는 방법은 쓰기 버퍼 내에 시간 순서대로 들어온 요청들을 일정 크기 이상의 시간 윈도우(time window)단위로 구분한다. 시간 윈도우로 구분된 데이터를 COLD로 정의하고 이후 NAND 플래시 메모리에 쓰일 때 한 칩에 배치시킨다. 이렇게 시간적·공간적 지역성을 이용해 데이터를 배치시키면 잦은 업데이트로 인해 유효한 페이지가 많이 발생하더라도 가비지 컬렉션 오버헤드를 크게 줄일 수 있다.

2. SSD 내에서 COLD 할당

그림 4는 SSD 내부에서 COLD가 어떤식으로 NAND 플래시 메모리에 할당되는지를 나타내고 있다. 앞서 설명했던 일반적인 인터리빙에서의 예와 같이 초기에는 데이터가 쓰이지 않은 상태라 가정하고 호스트에서 SSD로 데이터가 전달되면 먼저 쓰기 버퍼에 머물게 된다. 그리고 이 때 새롭게 제안된 배치 정책은 쓰기 버퍼 내에서 COLD를 탐색한다. 예를 들어 일반 데이터는 인터리빙으로 각 칩에 흩어져 저장되지만 COLD로 가정하는 D1~D2는 하나의 칩에 모여서 쓰여 지게 된다. 그림 4의 쓰기 버퍼 내에 가깝게 붙어 있는 데이터들을 COLD 데이터로 판정하였고 COLD 할당 정책에 따라 데이터를 배치하면 그림 4와 같이 데이터들이 NAND 플래시 메모리에 배치된다.

앞서 언급한 예시와 마찬가지로 D20~D27, D41~D44의 데이터가 수정 또는 삭제가 되었을 때의 SSD 상태를 그림 5에서 표현하고 있다. 업데이트가 요청된 페이지들은 데이터 수정 후 빈 영역에 다시 쓰여 지고 원래 데이터가 있던 페이지들은 모두 무효화 처리 된다. 무효화 처리된 데이터(invalid data)들이 흩어져 있게 되면 가비지 컬렉션 수행 시 무효화 처리된 데이터가 포함된 블록의 유요한 데이터들의 이동 연산이 많아지고 블록의 삭제 연산도 증가하기 때문에 무효화 처리된 데이터들이 모여 있다면 가비지 컬렉션의 오버헤드가 상당히 줄어든다. 그림 5의 쓰기 버퍼에서 D20~D27, D41~D44는 COLD로 판정되어 한 곳에 쓰여 지고 지역성을 가져 함께 업데이트 될 가능성이 높다. 그러므로 업데이트가 발생하면 무효화되는 데이터들이 모여 있게 되고 이후 가비지 컬렉션 수행 시 기존의 기법보다 상당한 성능 향상을 기대할 수 있다. 그림 5에서는 페이지 이동 오버헤드는 없고 3번의 삭제 연산만 추가적으로 발생하므로 그림 2의 7번의 페이지 이동 연산과 4번의 삭제 연산이 수행되는 것과는 대조적으로 상당히 적은 추가 연산이 발생하는 것을 알 수 있다.

Table 1. Information of MSR-Cambridge Trace

Name	Total Req. size(GB)	Read Req. size(GB)	Write Req. size(GB)
proj 0	59.09	13.12	45.97
prxy 0	56.84	3.05	53.80
rsrch 0	12.21	1.39	10.82
src1 0	1538.34	729.18	809.16
src1 2	52.97	8.82	44.15
stg 0	22.42	7.33	15.09
ts 0	15.47	4.13	11.34

인터리빙에 의해 분산 저장된 데이터는 이후 읽기 연산이 수행되면 마찬가지로 병렬적으로 읽기가 수행 될 수 있기 때문에 COLD는 읽기 연산에 상당한 약점을 가지고 있다고 생각될 수 있다. 하지만 NAND 플래시 메모리에 데이터를 읽어 들어오는 오버헤드가 FTL의 연산 오버헤드 보다 훨씬 크기 때문에 기존 기법과 마찬가지로 병렬 효과를 누릴 수 있다. 예로, 읽기 연산 오버헤드가 FTL 오버헤드보다 3배 오래 걸리는 것으로 가정한다. 이때 칩 1에서 읽기 연산이 발생하면 FTL은 칩 1에서 읽기 연산이 처리 되는 동안 다음 요청에 대한 FTL 연산을 수행이 가능하여 COLD로 인한 읽기 연산의 성능 저하는 작을 것으로 예상된다.

그러나 인터리빙을 사용함으로써 성능과 전력 소비의 트레이드 오프(trade off)가 발생할 수 있다. SSD에서 멀티 채널/웨이 구조를 이용한 인터리빙을 사용하게 되면 전체적인 성능 향상을 얻을 수 있다. 하지만 전력 소비 측면에서 적은 채널/웨이 사용을 요구한다[9]. 그러므로 COLD 배치 정책은 가비지 컬렉션의 추가적인 쓰기/삭제 연산과 멀티 채널/웨이 구조로 이한 전력 소비와 성능 향상을 효율적으로 제어할 수 있는 기법이다.

3. COLD 검출 및 분석 결과

실제 환경에서 얼마나 COLD가 발견되는 지를 알아보기 위해 본 논문에서는 잘 알려진 시스템 워크로드인 MSR-Cambridge 트레이스를 분석하였다[10]. 표 1은 실험에서 사용된 MSR-Cambridge의 트레이스들의 정보를 나타낸 표이다.

실험은 LRU 정책 기반의 캐시 시뮬레이터에서 수행 하였다. 버퍼에서 COLD를 찾기 위해 COLD_THRESHOLD (δ) 라는 값을 새롭게 정의 하였는데 버퍼 내에서 지역성을 가지는 데이터 집합 중 크기가 δ 이상이 되어야 COLD로 정의하였다. Ⅲ-1에서 설명한 것처럼 공간적 지역성과 시간적 지역성을 이용하여 COLD를 검출 하였다.

실험에서는 δ 값을 4, 16 ,64로 바꿔가면서 진행하였다. 그림6과 그림7은 캐시 시뮬레이터를 이용하여 각 트레이스를 수

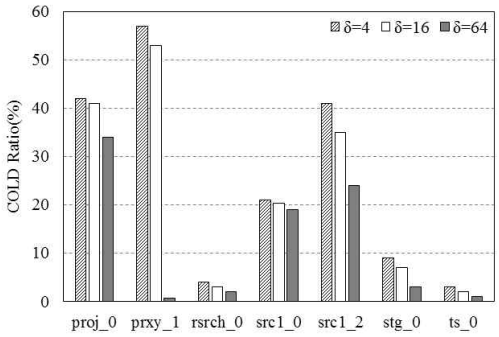

Fig. 6. Spatial Locality COLD Rate

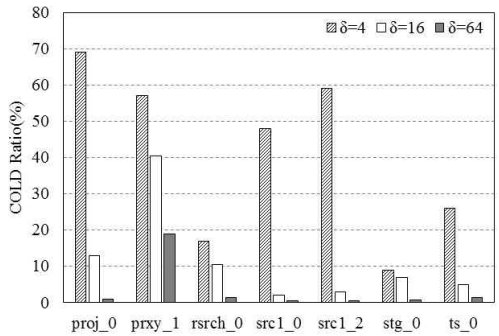

Fig. 7. Temporal Locality COLD Rate

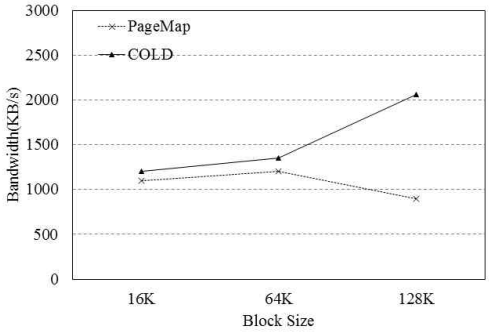

Fig. 8. Bandwidth of Read Request

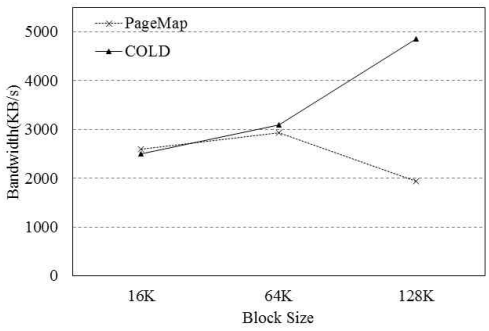

Fig. 9. Bandwidth of Write Request

행한 실험 결과를 나타내는 그림으로 트레이스에서 업데이트 되는 데이터들 중 지역성을 가지면서 δ에 따라 발견된 COLD의 비율을 나타내고 있다. 공간적 지역성을 가지는 COLD를 발견하기 위해서는 RB-Tree를 이용해 논리적 주소 순서대로 정렬을 시킨 후 연속되는 논리적 주소를 가짐과 동시에 δ 보다 큰 데이터 집합을 COLD로 정의 하였고 발견된 COLD의 비율을 그림 6에서 나타내고 있다. 시간적 지역성을 가지는 COLD을 발견하기 위해서 시간 윈도우의 크기를 δ로 정하고 δ 단위로 데이터를 구분하여 COLD로 정의 하였고 이를 그림 7에서 나타내고 있다.

여기서 주의 깊게 관찰할 사실은 발견되는 COLD가 많은 δ 가 무조건 적절한 값은 아니라는 것이다. 가비지 컬렉션의 오버헤드를 줄이기 위해서는 COLD과 많이 발견되어야 하는 것과 동시에 COLD의 크기가 커서 무효한 데이터가 한 곳에 최대한 많이 있어야하기 때문이다. 그러므로 이러한 점을 고려하였을 때 그림 6에서는 16이 가장 적절한 δ 값이고 그림 7에서는 4가 가장 적절한 δ 값이다.

IV. Results

1. 실험 준비

논문에서는 제안하는 COLD 할당 정책의 성능을 검증하기 위해 SSD 개발 보드인 Cosmos OpenSSD 개발 보드를 사용하였다[11,12].

Cosmos OpenSSD는 PCIe 인터페이스 기반의 SSD 개발보드로서 ARM Cortex-A9 core를 프로세서로 가지고 1GB DRAM과 최대 256GB의 NAND 플래시 메모리를 지원한다. 펌웨어는 기본적으로 PCIe 컨트롤러와 플래시 컨트롤러 관련 소스를 제공하고 페이지 레벨 맵핑 FTL 또한 제공한다.

OpenSSD의 기본 펌웨어는 페이지 매핑 FTL로 플레시 메모리 기반 저장장치의 핵심 소프트웨어 이다. 페이지 매핑 FTL은 데이터의 논리 주소를 실제 플레시 메모리의 물리 주소로 변경시키는 역할을 한다[4]. 개발 보드에서 제공하는 FTL은 채널 인터리빙을 지원하는 하드웨어에 맞춰 호스트에서 전달되는 쓰기 요청을 페이지 단위로 나눠 각 칩에 분산 저장 시킨다. 본 논문에서는 COLD 할당 정책을 구현하고 실험하기 위해서 페이지 레벨 FTL을 수정하였다. 또한 COLD 할당 정책 사용하기 위해 호스트에서 전달되는 논리적 주소를 변환하는 함수를 구현하고 변환된 주소를 저장할 수 있는 맵핑 테이블을 추가 하였다. 그리고 빠른 실험을 위해 Cosmos OpenSSD의 용량을 8GB로 조절하여 실험을 진행 하였다.

COLD 할당 정책의 성능 측정은 결국 가비지 컬렉션의 오버헤드를 측정하는 것과 같다고 말할 수 있다. 그러므로 SSD가 가비지 컬렉션이 수행될 수 있도록 유효한 공간이 부족하게 하여야 한다. 이를 위해 리눅스에서 제공하는 dd 명령을 이용해 SSD의 이용률을 거의 100%로 만든 다음 fio 벤치마크를 이용하여 성능을 측정하였다. fio 벤치마크는 연속적인 I/O 요청이 발생하게 하고 읽기와 쓰기 I/O가 3:7의 비율로 요청되도록 설

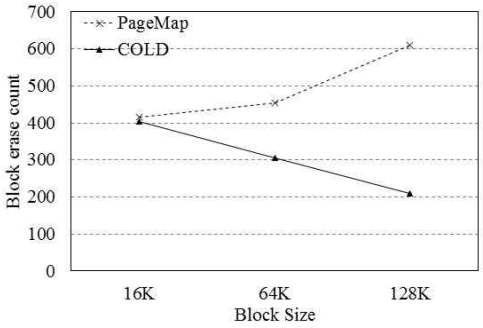

Fig. 11. Counts of Page Erase

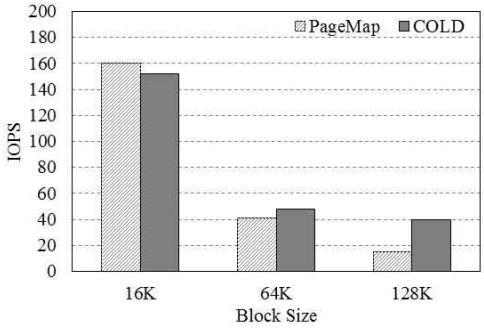

Fig. 12. IOPS of SSD

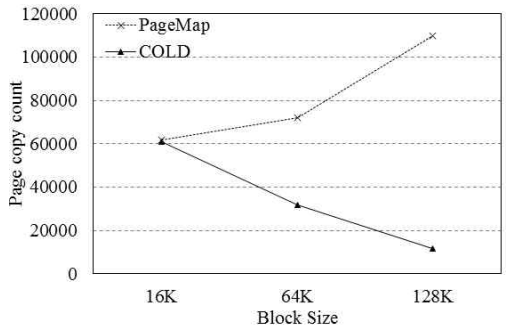

Fig. 10. Counts of Page Copy

정하였다. 그리고 블록의 크기를 여러 가지로 조절하면서 워크로드가 생성되도록 하였다. 또한 500MB 크기의 파일의 쓰기가 두 번 반복되도록 설정 하였는데 이렇게 함으로서 업데이트가 발생하여 가비지 컬렉션 이 빈 영역을 만들어내기 위해 수행되도록 하였다.

fio 벤치마크로의 실험을 통해 Cosmos SSD의 IOPS, 대역폭(bandwidth), 가비질 컬렉션으로 발생하는 추가적인 페이지 이동 연산, 삭제 연산을 측정할 수 있었다. 또한 추가적으로 발생한 페이지 이동 연산과 삭제 연산으로 인해 생기는 SSD의 에너지 소비에 대한 결과도 예측 할 수 있었다.

2. 실험 결과

그림 8, 그림 9는 I/O 요청 블록 크기에 따른 SSD의 대역폭 측정 결과를 나타내는 그림이다. 그림 8는 읽기 요청, 그림 9는 쓰기 요청에 대한 대역폭을 나타내고 있고, 그림의 증감의 모양 거의 비슷하지만 쓰기 요청의 대역폭이 읽기 요청에 비해 더 크게 측정되었다.

블록 사이즈가 작을 때는 COLD의 크기가 페이지 크기와 큰 차이가 없기 때문에 인터리빙에 의한 성능 저하의 정도는 적다. 하지만 요청되는 블록의 크기가 64K가 되면 COLD 할당 정책은 인터리빙에 인한 성능 저하가 심해진다. 하지만 그와 동시에 가비지 컬렉션 오버헤드도 줄어들기 때문에 기존의 방식과 COLD 할당 정책은 비슷한 속도를 나타내는 것이다. 하지만 인터리빙에 의한 성능저하보다 가비지 컬렉션으로 인해 발생하는 연산을 줄여서 얻을 수 있는 성능 향상이 더 크기 때문에 블록 사이즈가 128K일 때 COLD 할당 정책이 41% 더 높은 대역폭을 나타낸다.

그림 10과 그림 11은 가비지 컬렉션에 의해 발생되는 오버헤드를 나타내는 그림이다. 그림 10은 가비지 컬렉션에 의해 유요한 페이지의 이동 횟수를 나타낸 그림이고 그림 11은 블록 내의 유효한 페이지를 모두 이동 시킨 후 삭제 연산이 수행된 횟수를 나타낸다. 그림 10을 보면 요청된 블록의 크기가 16K일 때는 두 정책이 비슷한 성능을 내는 것을 볼 수 있는데 이는 COLD의 크기가 페이지 크기와 크게 차이가 없기 때문이다. 같은 맥락으로 그림 11의 삭제 횟수도 이유를 설명할 수 있다. 또한 요청된 블록의 크기가 64K일 때 COLD 할당 정책에서 페이지 이동 연산과 삭제 연산이 하락하였는데 이로 인해 그림 8과 그림 9에서 대역폭이 높아졌다는 것을 유추할 수 있다.

그림 12는 쓰기 요청에서 블록 크기에 따른 SSD의 IOPS를 나타낸 것이다. 그림 19는 블록 사이즈가 16K일 때, 블록 사이즈가 64K, 블록 사이즈가 128K일 때의 IOPS를 나타낸 그림으로 앞서 봤던 결과와 마찬가지로 전체적으로 COLD 할당 정책이 기존의 기법 보다 뛰어난 성능을 보인다.

진행된 실험이 연속적인 데이터를 이용한 실험이었기 때문에 공간적 지역성의 영향을 많이 받아 COLD 할당 정책이 월등한 성능을 보이고 있는 것이다. 하지만 랜덤한 데이터를 이용한 실험도 시간적 지역성을 나타낼 수 도 있기 때문에 COLD 정책은 다양한 워크로드에서 좋은 성능을 보일 것이라 예상된다.

V. Conclusions

본 논문에서는 성능 향상을 위해 사용되어 왔던 인터리빙 방식을 효율적으로 사용하고 지역성을 이용한 새로운 데이터 할당 정책을 제안하였다. I/O 요청이 들어 왔을 때 인터리빙을 통해 데이터를 병렬적으로 분산 처리하면 그 순간의 성능을 향상될 수 있지만 지역성을 가지는 데이터들이 분산되면 이후 가비지 컬렉션이 발생하였을 때 치명적인 오버헤드가 생길 수 있다. 본 논문에서 지역성을 가지고 이 후 업데이트 될 가능성이 높은 데이터를 COLD라 정의하였고 이러한 COLD들은 인터리빙 시키지 않고 한 곳에 저장하는 COLD 할당 정책을 제안하였다.

논문에서 제안한 COLD 할당 정책은 지역성을 가지는 데이터들은 한 곳에 저장하고 지역성을 가지지 않는 데이터들은 인터리빙으로 처리함으로써 가비지 컬렉션의 오버헤드를 크게 줄일 수 있었다.

REFERENCE

[1] 3D NAND, http://semimd.com/blog/2014/01/29/3d-nand-to10-nm-and-beyond/2014

[2] L.A. Barroso, "Warehouse-scale computing," SIGMOD'10, 2010.

[3] A. Ban, "Flash File System," United States Patent, No. 5,404,485, 1995.

[4] A. Gupta, Y. Kim, B. Urgaonkar, "DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings," ASPLOS, pp.229-240, 2009.

[5] J. Kim et al. "A specification flash translation layer for compactflash systems," IEEE Trans. on Consu. Electro. Vol. 48, No. 2, pp.366-375, 2002.

[6] D. Jung et al. “A Group-based Wear-Leveling Algorithm for Large-Capacity Flash Memory Storage Systems,” CASES 2007, pp.160-164, 2007.

[7] N. Agrawal et al. "Design tradeoffs for SSD performance," USENIX ATC, pp.57-70, 2008

[8] J. Kim, C et al. "Deduplication in SSDs: Model and quantitative analysis," MSST'12, pp.1-12, 2012

[9] Y. Oh, J. Choi, D. Lee, S. Noh, "Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems," FAST'12, 2012

[10] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-Loading: Practical Power Management for Enterprise Storag," FAST'08, 2008

[11] Cosmos OpenSSD project, www.openssd-project.org/wiki/The_OpenSSD_Project

[12] Y. Song, S. Jung, S. Lee, and J. Kim, "Cosmos OpenSSD: A PCIe-based Open Source SSD Platform," Flash Memory Submit, 2014

[13] H. Choi, Y. 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, 2015.

[14] B. Jung and J. Lee, "The buffer Management system for reducing write/erase operations in NAND flash memory," Journal of the Korea Society of Computer and Information, vol. 16, no. 10, pp.1-10, 2011.

[15] S. M. Huang and L. P. Chang, "A Locality-Preserving Write Buffer Design for Page-Mapping Multichannel SSDs," HPCC, pp.713-720, 2014.

Authors

Jung Kyu Park received the M.S. and Ph.D. degrees in computer engineering from Hongik University in 2002 and 2013, respectively.

He has been a research professor at the Dankook University since 2014.

His research interests include operating system, new memory, embedded system and robotics theory and its application.