DOI: 10.3745/KIPSTD.2010.17D.6.383

SSD 플래시 변환 계층 상에서 논리 주소 매핑의 성능 향상을 위한 HAMM(Hybrid Address Mapping Method)

이 지 원[†] · 노 홍 찬[††] · 박 상 현[†††]

요 약

최근 플래시 메모리 기반 SSD(Solid State Disks)는 데이터 처리 속도가 빠르고, 외부 충격에 강하며 전력소모가 작다는 우수한 특성과 함께 그 용량의 증가와 가격 하락으로 인하여 차세대 저장 매체로 부각되고 있다. 하지만 SSD는 하드디스크와는 달리 읽기, 쓰기 및 지우기의 단위 및 수행 시간이 다르며 덮어쓰기가 불가능하다는 특징이 있다. 이 때문에 SSD는 기존의 하드디스크 기반 시스템 상에서는 그 동작의 효율성이 떨어지며, 이를 보완하기 위해 플래시 변환 계층이 설계되었다. 본 논문에서는 플래시 변환 계층의 역할 중 하나인 논리 주소 매핑 기법을 개선하여 SSD의 성능을 높일 수 있는 HAMM(Hybrid Address Mapping Method)를 제안한다. HAMM은 기존에 존재하는 슈퍼 블록 매핑 기법과 블록 매핑 기법의 단점을 보완하고 장점을 살릴 수 있도록 설계된 논리 주소 매핑 기법이다. SSD 시뮬레이터를 제작하여 실험하였으며, 실험을 통하여 HAMM은 같은 크기의 쓰기 버퍼 상에서 슈퍼 블록 매핑 기법에 비해 SSD의 저장공간을 효율적으로 사용하는 것으로 나타났으며, 또한 블록 매핑 기법에 비해 매핑 테이블을 구성하는데 적은 양의 메모리를 사용하면서 비슷한 성능을 보이는 것으로 나타났다.

키워드 : SSD, 플래시 변환 계층, 논리 주소 매핑

HAMM(Hybrid Address Mapping Method) for Increasing Logical Address Mapping Performance on Flash Translation Layer of SSD

Jiwon Lee[†] · Hongchan Roh[††] · Sanghyun Park[†††]

ABSTRACT

Flash memory based SSDs are currently being considered as a promising candidate for replacing hard disks due to several superior features such as shorter access time, lower power consumption and better shock resistance. However, SSDs have different characteristics from hard disk such as difference of unit and time for read, write and erase operation and impossibility for over-writing. Because of these reasons, SSDs have disadvantages on hard disk based systems, so FTL(Flash Translation Layer) is designed to increase SSDs' efficiency. In this paper, we propose an advanced logical address mapping method for increasing SSDs' performance, which is named HAMM(Hybrid Address Mapping Method). HAMM addresses drawbacks of previous block-mapping method and super-block-mapping method and takes advantages of them. We experimented our method on our own SSDs simulator. In the experiments, we confirmed that HAMM uses storage area more efficiently than super-block-mapping method, given the same buffer size. In addition, HAMM used smaller memory than block-mapping method to construct mapping table, demonstrating almost same performance.

Keywords : Solid State Disks, Flash Translation Layer, Logical Address Mapping

1. 서 론

플래시 메모리는 비휘발성의 반도체로 하드디스크에 비해 전력 소모량이 작으며, 외부 충격에 강하기 때문에 최근에 휴대폰, MP3, PDA와 같은 모바일 기기의 저장 장치로 널리 쓰이고 있다[1]. 또한 플래시 메모리는 하드디스크에 내장된 디스크 헤드나 실린더와 같은 기계 장치가 없기 때문에 데

※ 이 논문은 2008년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(KRF-2008-313-D00849). 또한 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2010-0010689)
† 준 회 원 : 연세대학교 컴퓨터과학과 석사과정
†† 준 회 원 : 연세대학교 컴퓨터과학과 박사과정
††† 종신회원 : 연세대학교 컴퓨터과학과 부교수(교신저자)
논문접수 : 2010년 4월 6일
수 정 일 : 1차 2010년 5월 31일
심사완료 : 2010년 7월 6일

이터 접근 및 처리 속도가 빠르다는 장점이 있다. 이러한 다양한 장점을 바탕으로 플래시 메모리는 오늘날 널리 사용되고 있으며, SSD(Solid State Disks)형태로 개발되어 차세대 저장 장치로서 서버용 컴퓨터나 PC와 같은 컴퓨터의 하드디스크를 대체할 것으로 예상된다[2, 3].

하지만 플래시 메모리는 그 가격이 여전히 하드디스크에 비해 높기 때문에 플래시 메모리 기반 SSD가 PC의 하드디스크를 대체하기 까지는 아직 시간이 더 걸릴 것으로 예상되며, 현재는 데이터 접근 및 처리 속도가 중요시되는 서버용 컴퓨터의 하드디스크를 대체해 가고 있다[4, 5].

이러한 플래시 메모리와 SSD의 동작 원리는 하드디스크의 동작 원리와 상이하기 때문에 기존의 하드디스크 기반 파일 시스템과 데이터베이스 시스템을 플래시 메모리 기반의 저장 장치에 그대로 사용하기에는 어려움이 있다. SSD와 같은 플래시 메모리 기반의 저장 장치를 효율적으로 사용하기 위해서는 플래시 메모리의 특성에 적합한 시스템을 새롭게 설계하는 방법과 기존의 시스템들과 새로운 저장장치 사이에 새로운 계층을 설계하는 방법이 있다. 새로운 시스템을 설계하는 것은 매우 힘들고 어려운 작업이므로, 기존의 시스템들을 플래시 메모리 기반 저장 장치에 사용하기 위하여 새로운 계층인 플래시 변환 계층(FTL, Flash Translation Layer)이 존재하며, 그에 관한 많은 연구가 수행되어 왔다[1-3, 6-8].

오늘날 전자 상거래가 많이 발달함에 따라 그 특성에 따라 데이터베이스에 작은 크기의 임의 쓰기가 많이 요청된다. 플래시 메모리 기반의 저장 장치는 하드디스크에 비해 처리 속도가 빠르지만 플래시 메모리의 특징 때문에 여전히 작은 크기의 임의 쓰기에는 하드디스크와 크지 않은 성능 차이를 보인다. 반면에 작은 크기의 임의 쓰기에 효율적으로 작동하도록 설계된 플래시 변환 계층은 순차적인 대용량의 데이터 처리에 비효율적으로 작동할 우려가 있다.

논리 주소 매핑은 하드디스크 기반의 시스템과 플래시 메모리 또는 SSD 사이의 호환성을 높여주기 위한 방법이다. 논리 주소 매핑 기법 중 기존의 블록 매핑 기법은 플래시 메모리 상에서 널리 사용되었던 방법으로 SSD의 용량이 커짐에 따라 메모리의 필요량이 증가하였다. 반면에 SSD의 쓰기 성능을 높이기 위해 제안되었던 슈퍼 블록 매핑 기법은 임의 쓰기에 느린 성능을 보이며, 이를 보완하기 위한 임의 쓰기 지연 버퍼에 많은 양의 메모리가 필요하게 되었다.

본 논문에서는 블록 매핑 기법과 슈퍼 블록 매핑 기법이 갖는 메모리 필요량과 저장공간의 효율적 사용에 대한 문제점을 보완하고, 각 매핑 기법의 장점을 살릴 수 있도록 새로운 플래시 변환 계층을 설계하여 제시한다.

2장에서는 플래시 메모리와 SSD의 구조와 특징, 플래시 메모리와 SSD에서 사용되는 플래시 변환 계층에 대하여 설명하고 기존에 연구되었던 논리 주소 매핑 기법에 대해 살펴본다. 3장에서는 본 논문이 제안하는 방법을 설명하고, 4장에서는 본 논문에서 제안하는 방법과 기존의 방법들에 대한 실험 결과를 분석한다. 5장에서는 본 논문을 결론 짓고 향후 연구 방향을 제시한다.

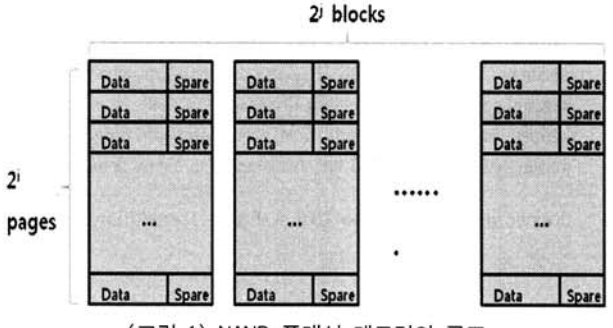
2. 배경 설명

2.1 플래시 메모리와 SSD의 특징

플래시 메모리의 종류에는 크게 NOR와 NAND 두 가지 형태가 있다. NOR 플래시 메모리는 NAND 플래시 메모리에 비해 여러 면에서 뛰어난 성능을 보이지만 그 단가가 높다. 따라서 오늘날 보조 저장 장치에는 단가가 비교적 낮은 NAND 플래시 메모리를 많이 사용하는 추세이다. (그림 1)은 이러한 NAND 플래시 메모리의 구조를 보여준다. 하나의 플래시 메모리 칩은 다수의 블록(block)으로 구성되며, 하나의 블록은 일반적으로 64개의 페이지(page)로 구성된다[9]. 각각의 페이지는 데이터를 저장하는 데이터 영역(data area)과 해당 페이지의 메타데이터(meta data)를 저장하는 예비 영역(spare area)로 이루어진다. 예비 영역은 일반적으로 128바이트(bytes) 정도의 크기를 가지며, 지우기 횟수, ECC(Error Correction Code) 등의 정보들이 저장된다.

또한 최근에 출시되는 플래시 메모리에는 SLC(Single-Level Cell)과 MLC(Multi-Level Cell)가 있다. SLC는 하나의 메모리 셀(cell)에 하나의 비트(bit)를 저장할 수 있으며, MLC는 하나의 메모리 셀에 다수의 비트를 저장할 수 있다. <표 1>에서 SLC와 MLC를 비교하여 보여준다. SLC는 MLC에 비해 쓰기 및 읽기 속도가 빠르고 오래 사용할 수 있으나, 단위 용량 당 가격이 비싸다. 반면에 MLC는 SLC에 비해 쓰기 및 읽기 속도가 느리고, 수명이 짧으나 동일한 가격에 더 큰 용량의 플래시 메모리를 사용할 수 있다.

전기적으로 작동하는 플래시 메모리에는 기계적으로 작동



(그림 1) NAND 플래시 메모리의 구조

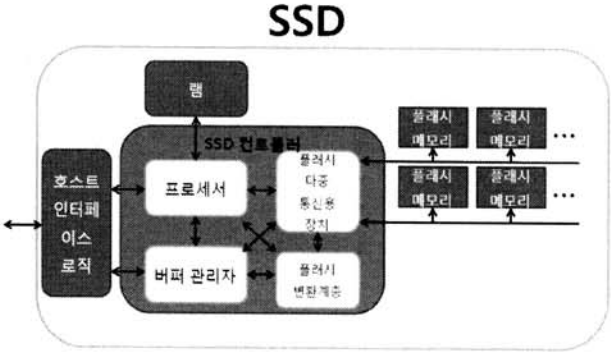
〈표 1〉 SLC와 MLC 비교[10]

	읽기 시간	쓰기 시간	지우기 시간	지우기 횟수
SLC	25μs	250μs	3500μs	100,000
MLC	70μs	1200μs	3500μs	10,000

하는 하드디스크에 존재하지 않는 여러 특징을 갖는다. 플래시 메모리에서는 데이터 쓰기, 읽기 단위와 지우기 단위가 서로 다르다. 즉, 데이터 쓰기와 읽기는 페이지 단위로 수행되는데 반해 데이터 지우기는 블록 단위로 수행된다. 그리고 플래시 메모리에는 각 블록 당 수행 가능한 지우기 횟수가 존재한다[11]. 일반적으로 각 블록 당 만 번에서 십만 번 정도의 지우기가 가능하며, 이를 초과하게 될 경우 해당 블록에 오류가 생겨 더 이상 사용할 수 없게 된다. 따라서 하나의 블록만 계속 사용하게 되면 해당 블록이 다른 블록들에 비해 먼저 소진될 수 있기 때문에 데이터 저장 시 예비 영역에 저장되어있는 지우기 횟수를 고려하여 각 블록 간의 지우기 횟수의 차이가 커지지 않도록 해야 하며, 이러한 기법을 웨어레벨링(wear-leveling)이라 한다. 또한 플래시 메모리는 내부 공간 업데이트(in-place update)가 불가능하다는 특징이 있다[5]. 이는 새로운 데이터를 저장하기 위해서 기존에 저장되어 있는 데이터를 지워야만 데이터 쓰기가 가능하다는 것이다. 하지만 쓰기의 단위와 지우기의 단위가 다르기 때문에 하나의 페이지 크기의 데이터를 업데이트하기 위해서 해당 페이지가 속하는 블록의 모든 페이지의 데이터를 지워야 새로운 데이터 쓰기가 가능하다. 그 외에 플래시 메모리는 <표 1>에서 보이는 바와 같이 읽기 속도에 비해 쓰기 속도가 매우 느리며, 지우기 시간도 매우 오래 걸린다.

SSD는 다수의 플래시 메모리에 동시에 쓰기를 수행함으로써 읽기 속도에 비해 느린 쓰기 속도를 갖는 플래시 메모리의 단점을 해결하고, 그 용량을 크게 하여 설계된 보조 저장 장치이다. SSD는 플래시 메모리 칩들의 데이터 공간을 효율적으로 관리하고 성능을 높이기 위해 플래시 변환 계층, CPU, 램(RAM, Random Access Memory) 등을 자체적으로 내장하고 있다. 또한 플래시 메모리의 장점과 함께 용량이 증가하고 가격이 빠르게 감소하고 있어 기존의 컴퓨터 시스템에서 가장 큰 병목 현상이던 보조 저장 장치의 데이터 처리 시간을 크게 줄일 수 있는 차세대 저장 장치로서 더욱 널리 사용될 것으로 기대할 수 있다[3].

SSD는 (그림 2)와 같은 일반적인 구조를 갖는다[12]. SSD



(그림 2) SSD의 일반적인 구조[12]

의 구성 요소는 크게 호스트 인터페이스 로직(host interface logic), SSD 컨트롤러(controller), 램, 그리고 다수의 플래시 메모리 칩으로 분류할 수 있다. SSD 컨트롤러는 SSD를 내장하는 시스템으로부터 호스트 인터페이스 로직을 통하여 데이터를 전송 받고, 전송 받은 데이터를 인터리빙(inter-leaving) 방식을 사용하여 다수의 플래시 메모리 칩에 거의 동시에 쓰기를 수행한다. SSD 컨트롤러는 데이터 채널(channel)을 통해서 하나의 플래시 메모리 칩에 데이터를 전송하고, 해당 플래시 메모리 칩에서 데이터 쓰기가 수행되는 동안 다른 플래시 메모리 칩에 데이터를 전송한다. 이처럼 인터리빙 방식을 사용하면 데이터 쓰기 시간보다 데이터 전송 시간이 짧은 점을 이용하여 동시에 다수의 플래시 메모리 칩에서 쓰기가 수행될 수 있다[12, 13].

하지만 SSD는 플래시 변환 계층에 의해 성능이 크게 달라질 수 있으며, 각각의 환경에 적합한 플래시 변환 계층이 다르기 때문에 모든 환경에서 일정 수준 이상의 성능을 보장할 수 있는 플래시 변환 계층이 필요하다.

2.2 플래시 변환 계층(FTL, Flash Translation Layer)

기존의 파일 시스템과 데이터베이스 시스템 등은 대부분이 하드디스크상에서 효율적으로 작동할 수 있도록 설계되었다. 따라서 플래시 메모리와 SSD에서 효율적으로 작동 가능한 파일 시스템과 데이터베이스 시스템이 필요하다. 플래시 변환 계층은 기존의 시스템들과 플래시 메모리 기반 저장장치 사이에 새로 설계된 펌웨어(firmware)로서, 플래시 메모리와 SSD가 기존의 하드디스크 기반의 시스템 상에서 효율적으로 작동할 수 있도록 도와준다. 플래시 변환 계층의 역할에는 크게 논리 주소 매핑(logical address mapping), 웨어레벨링, 가비지 컬렉션(garbage collection)이 있다.

논리 주소 매핑은 기존의 하드디스크 기반의 시스템이 플래시 메모리와 SSD상에서 효율적으로 작동 가능하도록 만드는 기법이다. 기존의 하드디스크에서 데이터가 업데이트 될 때 기존의 데이터가 저장되어 있던 공간에 덮어쓰기가 가능하지만 플래시 메모리는 덮어쓰기가 불가능하다. 그렇기 때문에 업데이트된 데이터를 저장하기 위해서는 기존에 저장되어 있던 데이터를 지워야만 새로운 데이터를 같은 공간에 기록할 수 있다. 하지만 앞에서 언급한 바와 같이 플래시 메모리는 쓰기와 지우기의 작업 단위가 다르고 지우기에 소요되는 시간이 길다. 그리고 데이터 기록 시 기존의 데이터를 지우는 작업을 수행한다면 그로 인한 시간에 의해 쓰기 성능이 저하된다. 이러한 문제점을 해결하기 위해 SSD 내부의 메모리에 매핑 테이블(mapping table)을 만들어 지우기 작업 없이 업데이트된 데이터를 저장할 수 있도록 하였다. 기존의 데이터에 해당하는 매핑 테이블의 포인터가 가리키는 플래시 메모리의 물리적 저장공간을 새로운 물리적 저장공간으로 바꾸고, 새로운 위치에 업데이트 데이터를 저장함에 따라 지우기 작업 없이 데이터를 저장할 수

있다. 논리 주소 매핑이란 이러한 매핑 테이블을 두어 SSD의 성능을 향상시키는 기술을 말하는 것이다.

웨어레벨링은 각 블록마다 제한된 지우기 횟수가 크게 벌이지지 않고 일정하게 소모하도록 하는 기법이다. 하드디스크와 달리 플래시 메모리는 각 블록 당 지우기 횟수가 제한되어 있기 때문에 특정 부분에 지우기 작업이 편중된다면 해당 부분의 지우기 횟수가 먼저 소모되어 전체 사용가능 용량이 줄어들 수 있고 그에 따라 성능이 저하될 수 있다. 따라서 데이터를 저장할 때 특정 부분의 제한된 지우기 횟수가 먼저 소모되지 않도록 지우기 횟수를 적절히 고려하여야 한다. 플래시 변환 계층은 이러한 웨어레벨링 기법이 효율적으로 수행될 수 있도록 설계되어야 한다.

플래시 메모리는 데이터 업데이트 요청 시 기존의 데이터를 그대로 두고 새로운 영역에 데이터를 저장하기 때문에 더 많은 데이터를 저장하기 위해서는 기존의 쓸모 없는 데이터를 지워야 한다. 이러한 쓰기 가능한 데이터 영역을 확보하는 작업을 가비지 컬렉션이라 한다. 가비지 컬렉션은 일반적으로 저장장치의 사용량이 일정이상 되었을 때, 그리고 프로세스 자원이 가비지 컬렉션을 수행할 충분한 여유가 있을 때 수행된다. 데이터 쓰기 작업 시 가비지 컬렉션이 발생하면 작업 시간이 지연될 수 있기 때문에 플래시 변환 계층은 가비지 컬렉션에 의한 성능 저하가 최소화 될 수 있도록 설계되어야 한다.

2.3 논리 주소 매핑

SSD 내부의 구조 및 플래시 변환 계층은 그 보안을 위해 SSD 제조 업체에서 핵심 기술을 공개하고 있지 않으며, 그에 따라 SSD 플래시 변환 계층에 관한 연구 또한 많이 이루어지지 않았다. 다만 여러 실험을 통해 SSD에서 사용되는 논리 주소 매핑 기법들에 대한 추측을 통하여 연구를 진행하고 있다. 초창기 SSD는 슈퍼 블록 단위로 데이터를 저장하는 슈퍼 블록 매핑 기법으로 예상 가능하며[13], 플래시 메모리를 일렬로 연결하여 사용하는 것처럼 데이터를 저장하는 블록 매핑 기법 또한 SSD에서 사용되는 것으로 예상할 수 있다. 그리고 여러 연구들을 통하여 오늘날의 많은 SSD들은 로그 기반 매핑 기법을 사용하는 것으로 예상되고 있다.

SSD 제조업체인 엠트론(MTRON)의 초창기 SSD 모델인 HYDRA 아키텍쳐(architecture)는 빠른 쓰기를 수행할 수 있도록 슈퍼 블록 매핑 기법을 사용하였다[3]. 슈퍼 블록 매핑 기법은 다수의 플래시 메모리 칩의 각 블록을 하나의 논리 블록으로 묶어 매핑 테이블에서 관리하는 기법이다. 이 기법은 인터리빙 방식을 사용함으로써 빠른 순차 쓰기가 가능하며, 논리 블록이 크기 때문에 매핑 테이블을 구성하는 데 메모리가 적게 필요하지만, 논리 블록이 커짐에 따라 작은 크기의 임의 쓰기에 의한 불필요한 복사가 많이 발생한다. 이를 해결하기 위하여 HYDRA 아키텍쳐는 버퍼링 기법을 이용하여, 작은 크기의 임의 쓰기를 버퍼에 임시로 저장했다가 한꺼번에 처리하도록 하였다[3]. 그러나 작은 크기의 임의 쓰기에 대해서 쓰기 성능을 보장하기 위해서는 많은 양의 메모리를 임의 쓰기를 지연시키기 위한 버퍼에 사용해야 한다는 단점이 있다. 또한 J. Kang은 슈퍼 블록 기반의 플래시 변환 계층을 제안하였다[1]. 이 연구는 기존의 블록 기반이었던 플래시 변환 계층을 슈퍼 블록 기반으로 새롭게 구성하여 제안하였다. 이 논문은 SSD가 아닌 플래시 메모리를 대상으로 하고 있으며, SSD 이전부터 슈퍼 블록 매핑 기법에 대한 연구가 되었다는 것을 보여준다. 이 기법은 인접한 논리 블록들을 하나의 슈퍼 블록으로 묶어 관리하도록 하였다. 플래시 메모리는 SSD 처럼 RAM이 존재하지 않기 때문에 이 기법에서는 쓰기 성능을 높이기 위해 플래시 메모리의 일부 블록을 쓰기 지연 버퍼와 같이 업데이트된 데이터를 저장하는 공간으로 사용하였다.

그에 비해 블록 매핑 기법은 플래시 메모리에서부터 사용되던 방법으로 각각의 블록을 하나의 논리 블록으로 다루는 방식이다. 이 기법은 슈퍼 블록 매핑 기법에 비해 논리 블록의 크기가 작기 때문에 같은 크기의 버퍼상에서 임의 쓰기에 대하여 저장공간을 효율적으로 사용 가능하다. A. BAN은 하나의 물리적 블록을 하나의 논리 블록으로 매핑하는 기법을 제안하였다[8]. 이 특허에서는 논리 주소 매핑 기법을 가상(virtual) 주소 매핑이라 하여 가상의 블록 주소를 플래시 메모리의 하나의 블록에 매핑하여 관리하도록 하였다. 그러나 플래시 메모리 및 SSD의 용량이 증가함에 따라 블록 매핑 기법은 슈퍼 블록 매핑 기법에 비해 상대적인 논리 블록의 크기가 작아지고, 그에 따라 논리 블록의 수가 많아지기 때문에 매핑 테이블을 구성하는데 더 많은 메모리가 필요하다는 단점이 존재한다.

로그 기반 매핑 기법은 플래시 메모리에서부터 널리 쓰이는 기법으로 SSD내에서 쓰일 것으로 추측되고 있으나 그 사용 여부를 정확히 알 수 없으며 실험을 통해 사용되는 것으로 추측되고 있다. 로그 기반 매핑 기법은 그 동안 끊임없이 연구 되었던 임의 쓰기에 의한 성능 저하를 줄이기 위한 하나의 기법으로, 매핑 테이블에 슈퍼 블록 또는 블록 기반으로 데이터를 저장하되 각 논리 블록에 페이지 단위로 데이터를 저장, 관리할 수 있는 로그 블록을 할당하여 업데이트 데이터를 로그 블록에 저장하도록 하였다[7]. 로그 기반 매핑 기법은 블록 크기 이하의 쓰기 요청에 의한 성능 저하를 줄이기 위한 기법으로 블록 또는 슈퍼 블록 매핑 기법과 페이지 매핑 기법을 복합적으로 사용한다. 최근의 많은 플래시 변환 계층에 관한 연구들이 로그 기반 매핑 기법을 응용하고 있다. J. Kim이 제안한 BAST는 플래시 메모리의 쓰기 성능을 높이기 위해서 로그 기반 매핑 기법을 사용하였다[2]. BAST에서는 블록을 데이터 블록과 로그 블록으로 구분하였으며, 데이터 블록은 매핑 테이블에서 블록 매핑 기법을 사용하여 관리하였다. 로그 블록은 매핑 테이

블 상에서 관리되지 않으며 작은 크기의 임의 쓰기에 의한 성능 저하를 줄이기 위해서 업데이트된 작은 크기의 새로운 데이터를 로그 블록에 저장하도록 하였다. 로그 블록에는 데이터를 페이지 단위로 저장하며 로그 블록이 가득 찼을 경우 데이터 블록과 병합, 교체 작업 등을 통하여 중복되는 데이터를 제거한다[2]. 데이터 블록에 남아 있는 업데이트되기 이전의 데이터를 삭제하고, 로그 블록에 저장되어 있던 업데이트된 데이터를 데이터 블록의 다른 데이터들과 병합하여 데이터 블록에 저장하여 로그 블록을 완전히 지우면서 새롭게 데이터를 저장할 수 있게 하였다. 하지만 로그 기반 매핑 기법은 읽기 작업 수행 시 로그 블록의 데이터를 함께 읽어야 하기 때문에 하나의 읽기 요청 당 두 번 이상의 읽기 작업이 수행되기 때문에 읽기 속도가 떨어지며, 하나의 논리 블록에 로그 블록을 함께 할당하기 때문에 SSD 용량의 전체를 데이터 영역으로 사용할 수 없다. 게다가 로그 블록이 가득 차면, 데이터 블록과 논리 블록을 병합하는 과정이 수반되기 때문에 쓰기 성능을 저하시킬 염려가 존재한다. 본 논문에서는 슈퍼 블록 매핑 기법 및 블록 매핑 기법이 갖는 장점을 살리면서 단점을 보완할 수 있고, 로그 기반 매핑 기법이 갖는 단점을 보완할 수 있도록 새로운 아이디어를 제안한다.

3. HAMM(Hybrid Address Mapping Method)

본 논문이 제안하는 HAMM은 기존의 슈퍼 블록 매핑 기법과 블록 매핑 기법의 단점을 보완하고 장점을 살릴 수 있는 논리 주소 매핑 기법이다. HAMM은 기본적으로 슈퍼 블록 매핑 기법과 블록 매핑 기법을 함께 지원한다. 매핑 테이블에서 정적인 슈퍼 블록 매핑과 동적인 블록 매핑을 함께 지원함으로써 적은 양의 메모리를 사용하면서 SSD의 효율적 관리와 성능 향상을 가능하게 한다.

HAMM의 목적은 크게 세 가지이다. 첫 번째 목적은 SSD내의 물리적 저장 공간을 효율적으로 사용하는 것이다. SSD는 논리 주소 매핑 기법에 따라서 저장 공간 사용의 효율성이 크게 달라질 수 있다. 임의 쓰기를 지연시키기 위한 버퍼의 크기가 동일할 때, 일반적으로 블록 매핑 기법이 슈퍼 블록 매핑 기법에 비해 저장 공간의 효율이 좋다[14]. 따라서 HAMM은 블록 매핑 기법과 비슷한 저장 공간의 효율성을 갖도록 하는 것이 그 첫 번째 목적이다. HAMM의 두 번째 목적은 버퍼와 매핑 테이블에 필요한 메모리의 양을 줄이는 것이다. 적은 양의 메모리를 사용하는 것은 SSD의 가격을 낮출 수 있는 중요한 요인 중 하나이다. 슈퍼 블록 매핑 기법은 임의 쓰기 요청에 대해 불필요한 복사를 수반하며, 저장 공간을 효율적으로 사용하지 못하기 때문에, 쓰기 요청된 데이터를 충분히 지연시키고, 그 데이터를 한꺼번에 쓰기 위한 버퍼를 위해 많은 양의 메모리를 필요로 한다. 또한 블록 매핑 기법은 논리 블록의 크기가 하나의 물리 블록의 크기와 같기 때문에, 매핑 테이블을 구성하고 저장하는데 있어 많은 양의 메모리를 사용한다. HAMM은 블록 매핑 기법과 비슷한 양의 버퍼를 사용하고, 매핑 테이블을 구성하는데 필요한 메모리의 양을 슈퍼 블록 매핑 기법 수준까지 줄이면서도 블록 매핑 기법과 비슷한 성능을 보일 수 있도록 한다. HAMM의 마지막 목적은 임의 쓰기가 요청된 데이터를 빠르게 처리할 수 있도록 하는 것이다. 2장에서 언급한 바와 같이 슈퍼 블록 매핑 기법은 임의 쓰기에 대해 매우 취약하다. 따라서 HAMM은 동적으로 블록 매핑이 가능하도록 하여 불필요한 복사 비용을 줄임으로써 임의 쓰기 속도를 높인다. 즉, 데이터가 버퍼로부터 출력될 때, 슈퍼 블록에 비해 블록 단위로 데이터를 출력하는 것이 효율적이라 판단되면 매핑 테이블에 블록 매핑을 위한 메모리를 동적으로 할당하고 데이터를 블록 단위로 저장한다. 이로 인해 슈퍼 블록 매핑 기법을 사용할 때 발생 가능한 불필요한 복사 비용을 줄일 수 있으며, 쓰기 성능을 더 높일 수 있다.

일반적으로 하드디스크는 데이터를 512B의 크기의 섹터로 처리하기 때문에 데이터 쓰기 요청 시, 데이터를 SSD에 저장하기 위해서는 페이지 주소를 계산해야 한다. 섹터의 주소를 *Saddr*, 페이지의 주소를 *Paddr*이라 할 때, 2KB 크기의 페이지 주소는 식 (1)과 같이 크기가 512B인 섹터의 주소를 4로 나누어 계산한다.

$$P_{addr} = \lfloor S_{addr} / 4 \rfloor \quad (1)$$

HAMM에서는 데이터를 슈퍼 블록, 블록 단위로 처리하기 때문에, 앞에서 계산한 페이지 주소를 바탕으로 해당 페이지가 속하는 논리 블록 주소, 논리 슈퍼 블록 주소를 구해야 한다. 논리 블록 주소를 *Baddr*, 논리 슈퍼 블록 주소를 *SBaddr*, SSD를 구성하는 플래시 메모리 칩의 수를 *Nflash*이라 하고 64개의 페이지가 한 개의 블록을 이룬다고 가정할 때, 논리 블록 주소, 논리 슈퍼 블록 주소는 각각 식 (2), (3)과 같이 계산한다.

$$B_{addr} = \lfloor P_{addr} / 64 \rfloor \quad (2)$$

$$SB_{addr} = \lfloor B_{addr} / N_{flash} \rfloor \quad (3)$$

HAMM에서는 쓰기 요청된 데이터를 버퍼 내에 저장할 때, 해당 데이터는 논리 블록 주소, 논 리 슈퍼 블록 주소를 포함하고 있다. 이는 버퍼로부터 데이터 출력 시, 동일한 논리 블록 주소 또는 동일한 논리 슈퍼 블록 주소를 갖는 데이터와 함께 출력할 수 있도록 하기 위함이며, 그에 따라 물리적 저장 공간을 효율적으로 사용할 수 있고 쓰기 성능을 높일 수 있다.

[알고리즘 1] HAMM 동작 과정

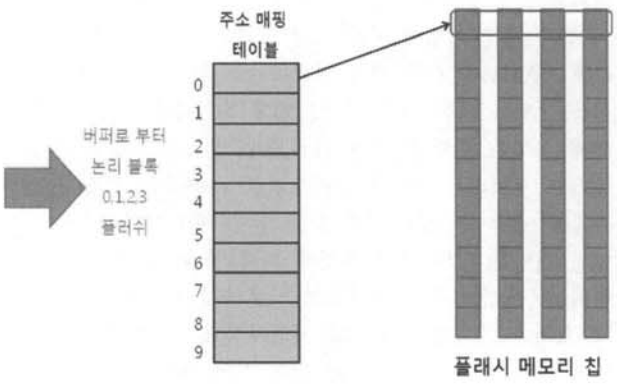
```
1 : Msb : # of data that max-super-block has
2 : MAXsb : # of the maximum data that a super block can have
3 : a : The threshold representing a unit of flush

4 : HAMM()
5 :   IF buffer is full THEN
6 :     FLUSH()
7 :     Save data requested for write operations into buffer
8 :   IF there is any change in max-super-block or max-block THEN
9 :     Recalculate new max-super-block, max-block

10 : FLUSH()
11 :   IF Msb >= MAXsb * a THEN
12 :     SUPERBLOCK_FLUSH()
13 :   ELSE
14 :     BLOCK_FLUSH()

15 : SUPERBLOCK_FLUSH()
16 :   Flush data of max-super-block from buffer into flash-disk
17 :   IF Flushed data previously have been saved by using block-mapping method THEN
18 :     Eliminate mapping pointers for block-mapping
19 :   Save flushed data by super-block-mapping

20 : BLOCK_FLUSH()
21 :     Flush data of max-block from buffer into flash-disk
22 :   IF Flushed data previously have been saved by using super-block-mapping method THEN
23 :     Maintain existing mapping pointer for super-block-mapping
24 :     Append a new pointer for block-mapping
25 :     Save flushed data by block-mapping
26 :   ELSE
27 :     IF it is possible for the flushed data to be transformed to super-block-mapping with data which have been saved by using block-mapping-method on same physical super block THEN
28 :       Eliminate mapping pointers for block-mapping
29 :       Append a new pointer for super-block-mapping
30 :       Save flushed data by super-block-mapping
31 :     ELSE
32 :       Save flushed data by block-mapping
```

버퍼에 데이터를 저장할 공간이 없을 때 버퍼로부터 데이터가 출력되며 이는 [알고리즘 1]의 5, 6번 줄에서 설명하고 있다. 버퍼로부터 데이터가 출력되는 경우는 슈퍼 블록, 블록 단위이며, HAMM에서 출력된 데이터를 플래시 메모리에 저장하는 경우는 크게 세 가지로 분류 가능하다. 첫 째는 데이터가 버퍼에서 슈퍼 블록 단위로 출력되며 정적인 매핑 테이블이 구성되는 경우이며, 이는 [알고리즘 1]의 SUPERBLOCK_FLUSH()에 해당된다. 동일한 논리 슈퍼 블록 주소를 갖는 데이터를 가장 많이 포함하고 있는 슈퍼 블록을 최대 슈퍼 블록이라 할 때, 최대 슈퍼 블록이 저장하고 있는 데이터의 수를 *Msb*, 슈퍼 블록이 저장할 수 있는 최대 데이터의 수를 *MAXsb*, 버퍼에서 데이터 출력 단위를 결정짓는 HAMM의 임계값을 a라고 할 때, 데이터가 슈퍼 블록 단위로 출력되는 경우는 식 (4)를 만족할 때이다.

$$Msb >= MAXsb * a \tag{4}$$

버퍼로부터 출력된 슈퍼 블록 크기의 데이터와 동일한 논리 슈퍼 블록 주소를 갖는 데이터가 이미 물리적 저장공간에 슈퍼 블록 단위로 저장되어 있거나 출력된 데이터가 기존에 저장되어있지 않던 새로운 데이터일 경우, 출력된 데이터의 논리 슈퍼 블록 주소에 해당하는 매핑 테이블의 위치에 포인터를 연결하고, 플래시 메모리 상에 적절한 위치를 선택하여 데이터를 저장하게 된다. 이 때, 기존에 블록 단위로 저장되어 있었다면 기존의 블록 매핑 포인터를 제거한다. 이러한 일련의 과정을 [알고리즘 1]의 16~19번 줄에서 설명하고 있다. (그림 3)은 4개의 플래시 메모리 칩으로 이루어진 SSD 내부에서 데이터가 슈퍼 블록 단위로 플래시 메모리에 저장되는 과정의 예를 보여준다. 버퍼로부터 논리 블록 주소 0, 1, 2, 3번에 해당하는 데이터가 출력된 경우, 출력된 모든 데이터들의 논리 슈퍼 블록 주소가 0번이기 때문에 슈퍼 블록 단위로 데이터가 저장되게 된다.

두 번째 경우는 버퍼에서 데이터가 블록 단위로 출력되며 정적인 매핑 테이블에 동적으로 메모리를 할당하여 블록 단위로 데이터를 저장하는 경우이다. HAMM에서는 슈퍼 블록 단위로 데이터를 저장하는 것보다 블록 단위로 데이터를 저장하는 것이 더 효율적이라고 판단될 때, 버퍼에서 데이

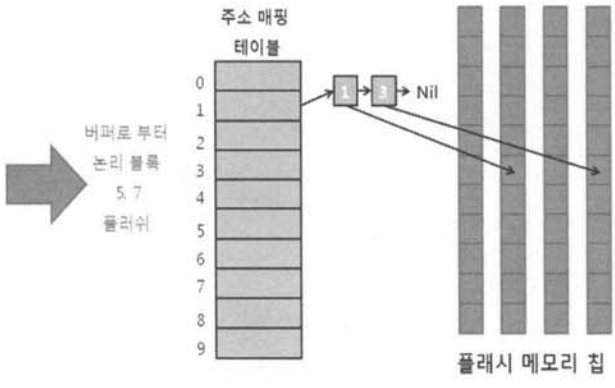


(그림 3) 슈퍼 블록 단위 데이터 저장

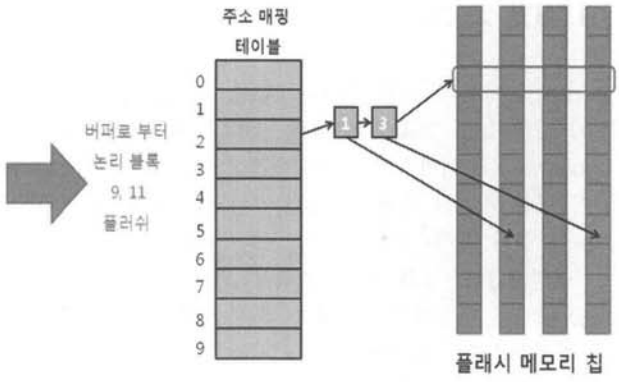
터를 블록 단위로 출력하게 되며, 이는 식 (5)를 만족할 때이다.

$$Msb < MAXsb * a \quad (5)$$

버퍼 내에 동일한 논리 블록 주소를 갖는 데이터를 가장 많이 포함하고 있는 블록을 최대 블록이라 할 때 버퍼로부터 최대 블록이 출력되게 되며, 매핑 테이블에 동적으로 포인터를 할당하여 블록 단위로 데이터를 저장한다. 이는 [알고리즘 1]의 BLOCK_FLUSH()에서 설명하고 있다. 블록 단위로 데이터가 저장되는 경우 동일한 논리 블록 주소를 갖는 데이터의 기존 저장 여부에 따라 저장 방식이 달라진다. 먼저 동일한 논리 블록 주소를 갖는 데이터가 기존에 존재하지 않는 경우, 동적으로 포인터를 할당하여 데이터를 저장하게 된다. 이 경우, 해당 데이터가 갖는 논리 슈퍼 블록 주소에 블록 매핑을 위해 메모리를 할당하고 포인터를 추가하여 데이터를 저장하며 알고리즘 1의 31, 32번 줄에서 설명하고 있다. (그림 4)는 새로운 데이터가 SSD에 저장되는 예를 보여준다. 버퍼로부터 동일한 논리 슈퍼 블록 주소 1번을 갖는 논리 블록 5, 7번에 해당하는 데이터가 출력된 경우, 정적인 매핑 테이블 1번에 논리 블록 5, 7번을 위한 포인터를 할당하고, 데이터를 블록 단위로 저장한다. 반면에 기존에 동일한 논리 블록 주소를 갖는 데이터가 슈퍼 블록 단위로 저장되어 있던 경우, 데이터의 일부분만 출력되었기 때문에 읽기 수행을 위해 기존의 매핑을 유지하면서 데이터를 블록 단위로 저장해야 한다. 이러한 과정은 알고리즘 1의 22, 23, 24, 25번 줄에서 설명하고 있으며, (그림 5)는 기존에 슈퍼 블록 단위로 저장되어 있던 데이터의 일부분이 업데이트된 경우를 설명하는 그림이다. 기존에 논리 슈퍼 블록 2번에 해당하는 데이터가 슈퍼 블록 단위 로 저장되어 있고, 버퍼로부터 논리 블록 9, 11번의 데이터가 출력된 경우, 해당 데이터의 논리 슈퍼 블록 주소는 2번이다. 매핑 테이블 상의 2번에는 기존에 슈퍼 블록 단위로 데이터가 저장되어 있었기 때문에 새로운 데이터를 블록 단위로 저장하기 위해 새로운 포인터를 할당하고 새로운 저장공간에 데이터



(그림 4) 블록 단위 데이터 저장(새로운 데이터)



(그림 5) 블록 단위 데이터 저장(데이터 일부분 업데이트)

를 저장한다. 기존에 저장되어 있던 논리 블록 8, 10번(논리 슈퍼 블록 2번의 0, 2번)데이터는 추후에 참조가 가능하도록 기존의 포인터를 유지한다. 이와 같이 데이터를 블록 단위로 저장하게 되면 버퍼에서 슈퍼 블록 단위로 데이터를 출력할 때에 비해 불필요한 복사의 양을 상당수 줄일 수 있다.

세 번째 경우는 앞에 설명한 두 가지 저장 방법의 특수한 경우이며, 기존에 블록 단위로 저장되어 있던 데이터가 슈퍼 블록 단위로 바뀌어 저장되는 경우이다. 앞에 설명한 첫 번째, 두 번째 저장 방법만을 이용하여 데이터를 저장할 경우 SSD를 어느 수준 이상 사용하였을 때, 모든 데이터가 블록 단위로 저장되는 경우가 발생 가능하다. 이 경우 결국엔 블록 매핑 기법과 같아지게 되므로 매핑 테이블의 크기가 커질 수 밖에 없다. 이러한 문제를 막기 위해서는 매핑 테이블을 최대한 슈퍼 블록 단위로 유지할 수 있도록 해야 한다. 즉, 순차적인 데이터가 일시적인 업데이트로 인하여 블록 단위로 저장되었다 하더라도 추후에 슈퍼 블록 단위로 다시 저장될 수 있도록 정책을 적절히 수립해야 한다. 먼저, 기존에 블록 단위로 저장되어 있던 데이터와 동일한 논리 블록 주소를 갖는 새로운 데이터가 슈퍼 블록 단위로 출력된다면, 기존의 블록 매핑에 할당된 포인터를 제거하고 새로운 물리적 저장 공간에 슈퍼 블록 단위로 데이터를 저장해야 한다. 또한 새로운 데이터가 블록 단위로 출력되더라도 슈퍼 블록 매핑으로 변환 가능하다면 슈퍼 블록 단위로 새롭게 저장하여야 한다. 즉, 동일한 논리 슈퍼 블록 내에 속하는 데이터의 일부만 블록 단위로 저장되어 있고, 기존에 저장되어 있지 않은 데이터 모두를 포함한 데이터가 출력된다면 슈퍼 블록 매핑으로 변환하여 저장 가능하다. 예를 들어, (그림 5)와 같이 데이터가 저장되어 있을 때, 논리 블록 8, 10번(논리 슈퍼 블록 2번의 0, 2번 블록)에 해당하는 데이터가 버퍼로부터 출력된다면, 기존의 저장되어 있던 논리 블록 9, 11번과 함께 슈퍼 블록을 이룰 수 있다. 따라서 기존에 블록 매핑을 위해 할당된 매핑 포인터를 제거하고, 슈퍼 블록 단위로 데이터를 저장한다. 이와 같이 블록 단위로 출력된 데이터가 슈퍼 블록 매핑으로 변환되는 과정을 알고리즘 1의 27, 28, 29, 30번 줄에서 설명하고 있다.

4. 실험 및 결과

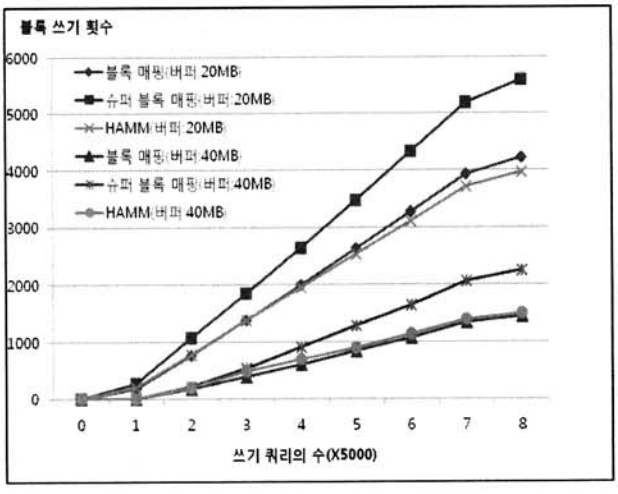
본 논문에서의 실험 환경은 AMD Athlon 64 X2 Dual 5200(2.71GHz), 2GB RAM의 시스템에 운영 체제로 Windows XP 서비스팩 3으로 설정하였다. 공개된 SSD 플래시 변환 계층이 존재하지 않기 때문에 실험의 비교 대상으로는 블록 매핑 기법과 슈퍼 블록 매핑 기법을 이론적으로 구현하였다. 또한 실제 플래시 변환 계층을 구현하고 SSD 내부에 장착하여 실험하는 것은 거의 불가능하기 때문에 자체적으로 시뮬레이터를 구현하여 실험하였다. 시뮬레이터는 SSD 생성기를 통하여 생성한 가상의 SSD의 정보를 읽어, 그의 특성에 맞게 쓰기 지연 버퍼, 매핑 테이블, 물리적 저장 공간 등을 설계하도록 하였다.

실험에 사용할 데이터는 OLTP기반의 TPC-C 벤치마크[15] 프로그램인 해머오라(Hammerora)를 MySQL 5.0에서 실행시켜 발생하는 I/O를 Diskmon[16] 프로그램을 사용하여 추출하였다. TPC-C 벤치마크는 가상의 온라인 상점을 가정하고 주문, 결제, 배달, 주문 상황, 재고 수준의 총 5종류의 트랜잭션(Transaction)을 임의로 발생시켜 OLTP 시스템의 성능을 측정할 수 있도록 제정한 모델이다[15]. TPC-C 벤치마크 프로그램에서 발생되는 5종류의 트랜잭션에 의해 데이터베이스에서 수행되는 "select * from WAREHOUSE where W_ID=150"와 같은 읽기 쿼리(Query)와 "update ORDER set O_OL_CNT=1 where O_ID=2500"와 같은 쓰기 쿼리를 디스크에 데이터가 입력 및 출력되는 것을 감시하는 프로그램인 Diskmon을 사용하여 실험에 사용할 데이터를 추출하였다. 데이터는 총 4번 추출하였으며 실행 시간과 가상 사용자의 수를 변경시켜가며 I/O를 추출하였다. 실험 결과 가상 사용자의 수와 실행 시간 모두 SSD 플래시 변환 계층의 성능 결과 비교에는 영향이 거의 없는 것으로 나타났다.

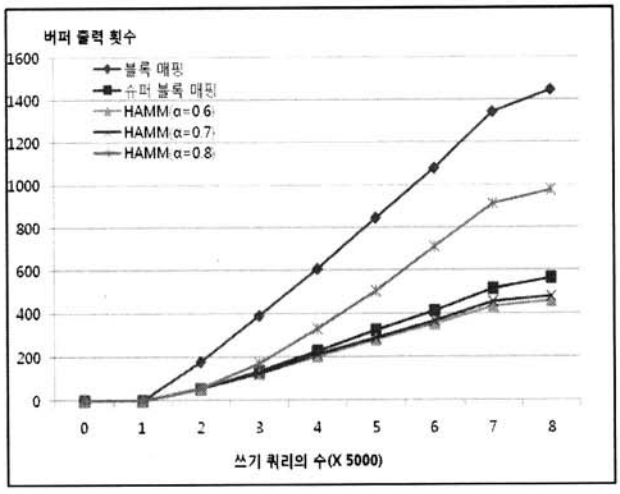
SSD 시뮬레이터를 통하여 하나의 블록이 2KB 크기의 페이지 64개로 이루어졌으며, 플래시 메모리 4개로 이루어진 2GB의 용량을 갖는 가상의 SSD를 생성하여 실험하였다. 쓰기에 관한 실험은 Diskmon을 이용하여 추출한 데이터 중 쓰기 쿼리만을 대상으로 하였으며, 약 4만 개의 쓰기 쿼리로 이루어진 데이터를 이용하여 실험하였다. 읽기 실험 역시 Diskmon을 이용하여 추출한 데이터 중 읽기, 쓰기 쿼리를 대상으로 실험하였으며, 약 13만 개의 쿼리로 이루어진 데이터를 이용하여 실험하였다.

(그림 6)은 버퍼의 크기에 따라서 데이터가 쓰여지는 블록의 수가 달라지는 것을 보여주는 그래프이다. 블록 매핑 기법, 슈퍼 블록 매핑 기법, HAMM(α = 0.8)을 대상으로 버퍼의 크기를 20MB, 40MB로 변화 시켜 실험을 수행하였다. 논리 주소 매핑 기법의 종류에 상관없이 그래프를 통하여 버퍼의 크기가 커질수록 데이터가 쓰여지는 블록의 수가 적어진다는 것을 확인할 수 있다. 또한 버퍼의 크기가 2배이지만 블록 쓰기 횟수가 1/2배를 초과하는 것을 볼 때, 데이터가 완전히 채워지지 않은 블록이 버퍼로부터 출력됨에 따라서 저장 공간이 더 많이 낭비된다는 것을 확인 가능하다. 또한 슈퍼 블록 매핑이 블록 매핑과 HAMM에 비해 저장 공간의 낭비가 심하며, 블록 매핑과 HAMM은 비슷한 성능을 보인다는 것을 확인할 수 있다.

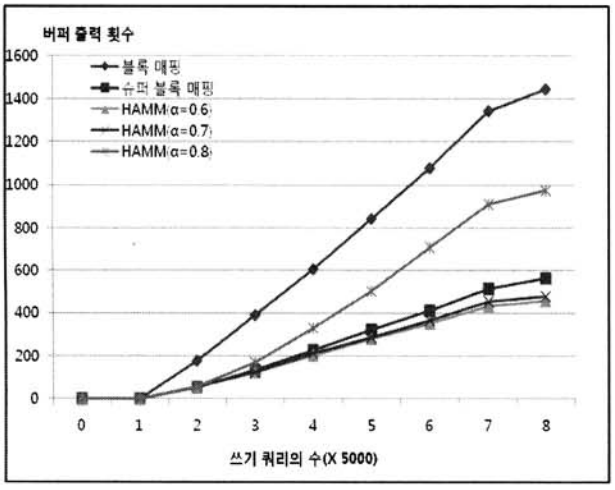
HAMM은 버퍼로부터 데이터 출력 단위를 결정하는 임계값에 따라서 그 성능의 차이를 보인다. (그림 7)은 HAMM의 임계값에 따른 성능의 변화를 실험한 결과를 보여주는 그래프이다. 버퍼의 크기가 40MB인 환경에서 임계값의 크기를 0.6, 0.7, 0.8로 바꾸어 가며 실험을 수행하였다. 실험 수행 결과 임계값의 크기가 커질수록 블록 쓰기 횟수가 줄어든다는 것을 알 수 있었으며, 그래프에서 확인 가능하다. 즉, 임계값이 0에 가까워질수록 슈퍼 블록 매핑 기법과 비슷해지고, 1에 가까워질수록 블록 매핑 기법에 가까워진다는 것을 알 수 있다. 또한 HAMM이 슈퍼 블록 매핑 기법



(그림 6) 버퍼의 크기에 따른 블록 사용량(α = 0.8)



(그림 7) HAMM의 임계값에 따른 블록 사용 쓰기 횟수 (버퍼 : 40MB)



(그림 8) 버퍼 출력 횟수(버퍼 : 40MB)

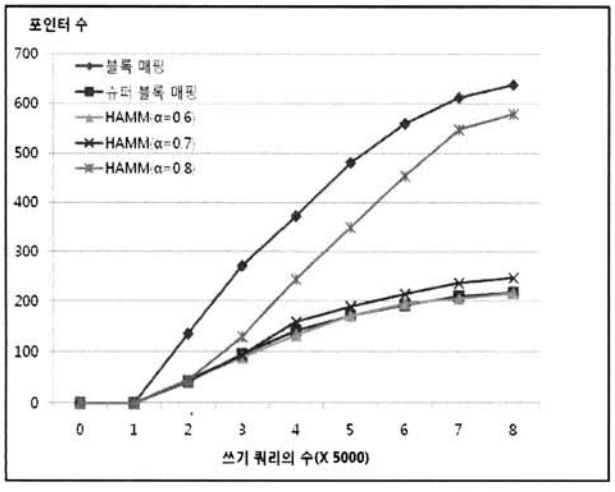
에 비해 블록 쓰기 횟수가 작은 점에서, 플래시 메모리 칩의 저장공간을 효율적으로 사용한다는 것을 알 수 있다.

(그림 8)은 각 논리 주소 매핑 기법에 따라 버퍼에서 데이터를 출력하는 횟수를 실험한 결과이다. 버퍼의 크기를 40MB로 설정하고 블록 매핑 기법, 슈퍼 블록 매핑 기법, HAMM(α = 0.8)을 비교하였을 때 블록 매핑 기법이 버퍼에서 데이터가 출력되는 횟수가 가장 많다는 것을 확인할 수 있다. 즉, 블록 매핑 기법은 슈퍼 블록 매핑 기법이나, HAMM에 비해 데이터 출력 단위가 작기 때문에 데이터를 더 작은 주기로 출력해야 한다. 이는 버퍼에서 데이터가 출력될 때 인터리빙 기법에 의해 출력되는 데이터의 양에 관계없이 거의 동일한 시간이 쓰기에 소요된다 가정할 때, 블록 매핑 기법에 비해 슈퍼 블록 매핑 기법, HAMM에 소요되는 시간이 훨씬 적다는 것을 알 수 있다. 다만, 이 실험에서 불필요한 데이터 복사, 가비지 컬렉션 등에 소요되는 시간은 결과에 포함되지 않는다.

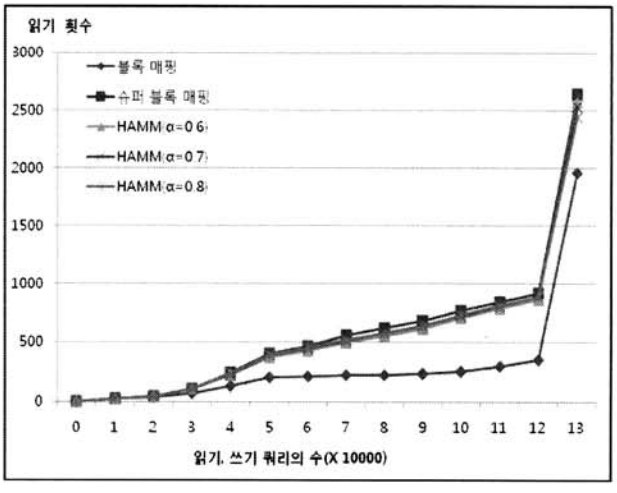
(그림 9)는 매핑 테이블 상에 논리 주소 매핑을 위한 포인터의 수에 따른 메모리 사용량을 비교하여 보여주는 실험 결과이다. 그래프에서 보여지는 바와 같이 블록 매핑 기법은 슈퍼 블록 매핑 기법에 약 3배 정도로 메모리 사용량이 많다. HAMM의 경우 앞의 실험에서 임계값이 커질수록 블록 쓰기 횟수가 적어진다는 것을 알 수 있었다. 하지만 임계값이 커질수록 블록 매핑 기법에 가까워지기 때문에, 매핑 테이블 구성 시 필요한 메모리의 양이 더 많다는 것을 실험을 통해 확인하였다. 따라서 적당한 임계값을 찾아 설정하는 것 또한 HAMM의 성능에 크게 영향을 미칠 수 있다고 판단된다. 최적의 임계값을 찾아 설정한다면 HAMM은 블록 매핑 기법에 비해 매핑 테이블의 크기를 크게 줄일 수 있다.

(그림 10)은 각 매핑 기법의 읽기 성능을 비교하여 보여주는 실험 결과이다. 그래프에서 보여지는 바와 같이 슈퍼 블록 매핑 기법이 읽기 횟수가 가장 많고, 그 뒤로 HAMM, 블록 매핑 기법 순으로 읽기 횟수가 많다. 그러나 13만여 개의 쿼리를 수행하였을 때 수행되는 읽기 횟수가 2000~2500회 정도인 것을 봤을 때 매핑 테이블을 통해 읽기가 수행되는 횟수가 매우 적다는 것을 알 수 있다. 또한 표 1에서 보여지는 바와 같이 읽기에 소요되는 시간이 쓰기에 소요되는 시간의 10분의 1 미만인 것을 볼 때, 읽기 성능은 각 매핑 기법에 크게 영향을 받지 않는 것으로 생각될 수 있다.

(그림 11)은 임계값의 변화에 따른 HAMM의 성능을 비교 실험한 그래프이다. 임계값을 0.05에서 0.95까지 0.05씩 증가시키며 실험을 수행하였으며, 임계값 0.4 이전에서는 거의 동일한 성능을 보이기 때문에 (그림 11)에서 생략하였다. 그래프를 통해서 임계값이 커질수록 읽기 횟수가 줄어드는 것을 확인할 수 있으며 또한 임계값이 0.7에서 0.8로 커지는 구간에서 블록에 쓰기가 수행되는 수가 급격히 줄어들며, 매핑 테이블 상의 포인터의 수가 급격히 증가하는 것을 볼 수 있다. 이는 임계값이 0.7에서 0.8로 증가할 때, 임계값에 따라 HAMM의 성능이 민감하게 변한다는 것을 의미한다.



(그림 9) 매핑 테이블 상의 포인터의 수(버퍼 : 40MB)

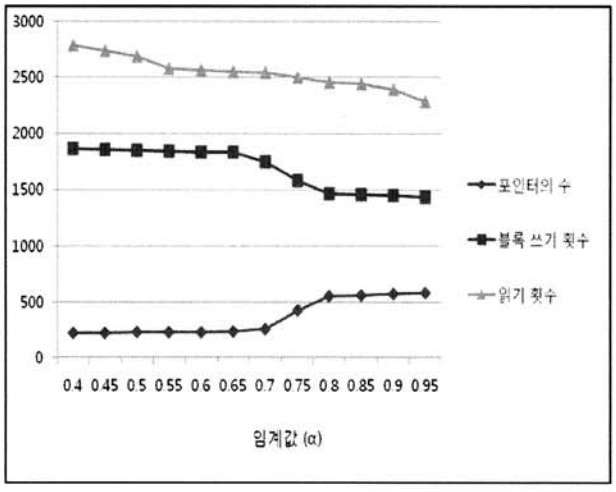


(그림 10) 논리 블록 읽기 횟수(버퍼 : 40MB)

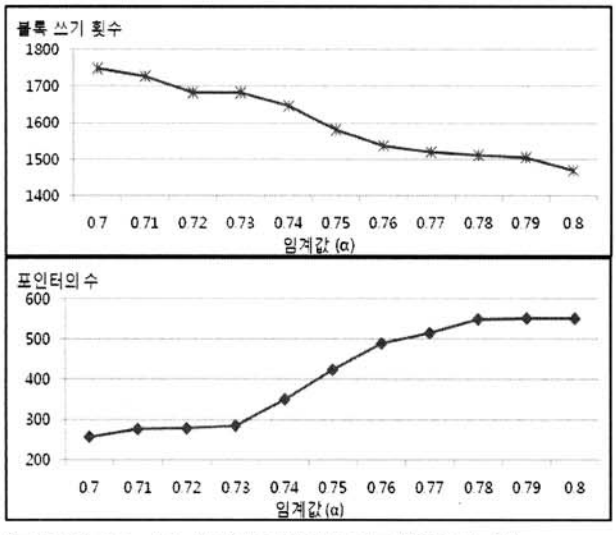
(그림 12)는 본 실험이 수행된 환경하에서 최적의 임계값을 찾기 위하여 (그림 11)에서 HAMM의 성능이 급격히 바뀌는 구간을 좀 더 자세하게 측정한 결과를 보여주는 그래프이다. HAMM의 쓰기 횟수 및 매핑 포인터의 수가 급격히 변하는 구간을 찾아 최적의 임계값을 구하기 위해 임계값을 0.7부터 0.8까지 0.01씩 증가시키며 실험을 수행하였다. 그래프에서 보여지는 바와 같이, 매핑 포인터의 수는 임계값 0.73부터 급격히 증가하는 것과 달리, 블록 쓰기 횟수는 상대적으로 적은 감소를 하는 것을 볼 수 있다. 특히 임계값이 0.73에서 0.78로 증가할 때, 매핑 포인터의 수가 약 2배 증가하기 때문에, 본 실험환경에서는 매핑 포인터의 수가 급격히 증가하기 이전의 임계값인 0.73이 최적의 임계값이라 생각할 수 있다.

HAMM의 성능의 정확한 측정을 위해 이전 실험까지 사용했던 벤치마크 프로그램인 해머오라 외에 또 다른 벤치마크 프로그램인 BenchmarkSQL을 사용하여 데이터를 추출하여 실험을 수행하였다[17]. BenchmarkSQL을 통하여 약 6만개의 쓰기 쿼리와 읽기 성능 실험을 위해 읽기, 쓰기 20만개의 쿼리를 추출하여 실험을 수행한 결과, 블록 쓰기 횟수, 버퍼 출력 횟수, 매핑 포인터의 수, 읽기 횟수 측면에서 HAMM의 성능은 해머오라를 사용했던 이전 실험들과 동일한 결과를 얻었다.

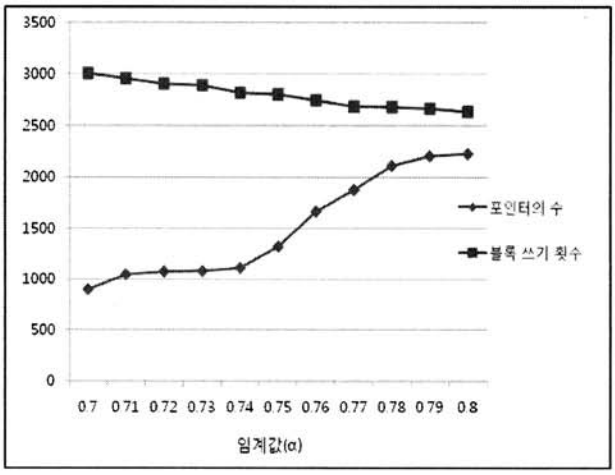
(그림 13)은 BenchmarkSQL을 사용한 실험 환경 하에서 최적의 임계값을 얻기 위해 수행한 실험을 보여주는 그래프이다. 이전 실험과 동일한 40MB 크기의 버퍼를 사용하였으며, 임계값을 0.7에서 0.8까지 0.01씩 증가시켜가며 실험을 수행하였다. 물리적인 블록에 데이터를 기록하는 블록 쓰기 횟수는 임계값이 증가할수록 약간씩 감소하였다. 그러나 그래프에서 확인 가능하듯이 매핑 테이블 상의 포인터의 수가 임계값 0.74 지점에서 급격하게 증가하는 것을 볼 때 BenchmarkSQL을 사용한 이번 실험 환경에서는 0.74의 임계값이 최적이라고 생각해 볼 수 있다.



(그림 11) 임계값에 따른 HAMM의 성능(버퍼 : 40MB)



(그림 12) 0.7~0.8 사이의 임계값에 따른 HAMM의 성능 (버퍼 : 40MB)



(그림 13) BenchmarkSQL을 사용한 실험 환경에서 0.7~0.8 사이의 임계값에 따른 HAMM의 성능(버퍼 : 40MB)

5. 결 론

플래시 메모리 기반 SSD는 하드디스크가 갖는 여러 단점을 해결할 수 있는 차세대 저장장치로 각광받고 있다. SSD 내부에는 기계적 동작이 없기 때문에 데이터 접근 및 처리 속도가 매우 빠르고, 전력 소모가 작으며, 외부 충격에도 고장이 적은 장점이 있다. 하지만 SSD는 하드디스크와는 동작 원리가 상이하기 때문에 기존의 하드디스크 기반 시스템 상에서 동작하는 것은 SSD의 성능 저하를 초래한다. 따라서 SSD와 기존의 시스템 사이에 플래시 변환 계층을 설계하여 SSD의 성능을 높일 수 있도록 하였다.

본 논문에서 제안하는 HAMM은 논리 주소 매핑이 슈퍼블록 단위뿐 아니라 블록 단위도 가능하여 각각의 단점을

보완하고 장점을 살릴 수 있는 새로운 논리 주소 매핑 기법이다. 또한 메모리를 동적으로 할당하여 메모리 사용의 효율을 높였으며, 작은 크기의 임의 쓰기에 강하고, 적은 양의 메모리로 고성능을 보일 수 있도록 설계하였다. 실험을 통하여 HAMM이 슈퍼 블록 매핑 기법에 비해 같은 크기의 쓰기 지연 버퍼 상에서 저장 공간을 더 효율적으로 사용하는 것을 알 수 있었으며, 블록 매핑 기법에 비해 매핑 테이블의 크기가 작아지는 것을 확인할 수 있었다.

본 논문에서는 플래시 변환 계층의 역할 중 논리 주소 매핑에 초점을 두었지만 추후의 연구에서는 웨어레벨링, 가비지 컬렉션 등 SSD의 성능에 영향을 미치는 여러 요인들을 고려하여 효율적인 SSD 시스템 설계에 관한 연구를 수행할 계획이다. 또한 버퍼로부터 출력되는 단위를 결정하는 임계값을 개별 시스템 환경에 맞게 자동 조절될 수 있도록 연구를 확장할 계획이다.

참 고 문 헌

[1] Jeong-Uk Kang, Heeseung Jo, Jin-Soo Kim and Joonwon Lee, "A Superblock-based Flash Translation Layer for NAND Flash Memory", Proceedings of the 6th ACM & IEEE International conference on Embedded software, pp.161-170, 2006.

[2] Jesung Kim, Jongmin Kim, Sam H. Noh, SangLyul Min and Yookun Cho, "A Space-Efficient Flash Translation Layer for Compactflash Systems", IEEE Transactions on Consumer Electronics, Vol.48, No.2, pp.366-375, 2002.

[3] Young-Hyun Bae, "Design of A High Performance Flash Memory-based Solid State Disk", KIISE, Vol.25, No.6, pp. 18-28, 2007.

[4] Jim Gray, Bob Fitzgerald, "Flash Disk Opportunity for Server Applications", ACM Queue, Vol.6, No.4, pp 18-23, 2008.

[5] Goetz Graefe, Hewlett-Packard Laboratories, "The Five-Minute Rule 20 Years Later: and How Flash Memory Changes the Rules", ACM QUEUE, Vol.6, Issue 4, pp.40-52, 2008.

[6] Sangwon Lee, Dongjoo Park, Teasun Chung, Dongho Lee, Sangwon Park and Hajoo Song, "A Log Buffer-Based Flash Translation Layer Using Fully Associative Sector Translation," ACM Transactions on Embedded Computer Systems, Vol.6, No.3, Article No. 18, 2007.

[7] Intel Corporation, "Understanding the Flash Translation Layer(FTL) Specification", Dec. 1998.

[8] Amir Ban, "FLASH FILE SYSTEM", USPTO.5404485, 1995.

[9] Aayush Gupta, Youngjae Kim and Bhuvan Urgaonkar, "DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings", Proceeding of the 14th international conference on Architectural support for programming languages and operating systems, pp.229-240, 2009.

[10] http://www.todaysppc.com/mbzine/bbs/zboard.php?id=TipsNManuals&page=1&sn1=&divpage=1&sn=off&ss=on&sc=on&select_arrange=headnum&desc=asc&no=44

[11] Eran Gal et al., "Algorithms and Data Structures for Flash Memories", Vol.37, No.2, pp.138-163, 2005.

[12] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy, "Design tradeoffs for SSD performance", In USENIX Annual Technical Conference, 2008.

[13] Jeong-Woo Lee, Dong-Ryul Ryu, "SEMICONDUCTOR SOLID STATE DISK CONTROLLER", USPTO. 20070106836, 2007.

[14] Jiwon Lee, Hongchan Roh, Sanghyun Park, "Dual Address Mapping Method for Efficient Wear-Leveling of SSDs", Proceedings of 1st International Conference on Emerging Databases(EDB2009), pp.29-33, August, 2009.

[15] W. W. Hsu, A. J. Smith, and H. C. Young, "I/O reference behavior of production database workloads and the TPC benchmarks - an analysis at the logical level" Technical Report CSD-99-1071, Computer Science Division, University of California, Berkeley, Nov. 1999.

[16] http://technet.microsoft.com/en-us/sysinternals/bb896646.aspx

[17] http://sourceforge.net/projects/benchmarksql/

이 지 원

e-mail : sppong@cs.yonsei.ac.kr

2008년 2월 연세대학교 컴퓨터과학과(학사)

2008년 3월~현 재 연세대학교 컴퓨터과학과 석사과정

관심분야 : 플래시메모리, SSD, 데이터베이스

노 홍 찬

e-mail : fallsmal@cs.yonsei.ac.kr

2006년 2월 연세대학교 컴퓨터과학부(학사)

2008년 2월 연세대학교 컴퓨터과학과(공학석사)

2008년 3월~현 재 연세대학교 컴퓨터과학과 박사과정

관심분야 : 플래시메모리 인덱스, SSD, 데이터마이닝

박 상 현

e-mail : sanghyun@cs.yonsei.ac.kr

1989년 2월 서울대학교 컴퓨터공학과(학사)

1991년 2월 서울대학교 컴퓨터공학과(공학석사)

2001년 2월 UCLA대학교 전산학과(공학박사)

1991년 3월~1996년 8월 대우통신 연구원

2001년 2월~2002년 6월 IBM T. J. Watson Research Center Post-Doctoral Fellow

2002년 8월~2003년 8월 포항공과대학교 컴퓨터공학과 조교수

2003년 9월~2006년 8월 연세대학교 컴퓨터과학과 조교수

2006년 9월~현 재 연세대학교 컴퓨터과학과 부교수

관심분야 : 데이터베이스, 데이터마이닝, 바이오인포매틱스, 적응적 저장장치 시스템