# 낸드 플래시 메모리 기반 SSD를 위한 작업부하 적응형 동적 페이지 매핑 기법

## (WADPM : Workload-Aware Dynamic Page-level Mapping Scheme for SSD based on NAND Flash Memory)

하 병 민 † 조 현 진 † 엄 영 익 ††
(Byungmin Ha) (Hyunjin Cho) (Young Ik Eom)

**요 약** 낸드 플래시 메모리를 이용한 SSD(Solid State Disk)는 하드 디스크를 대체할 매체로 주목 받고 있다. SSD는 성능을 최대화 하기 위해 다수의 낸드 플래시 메모리를 병렬적으로 구성한다. 하지만 SSD에 하이브리드 매핑 기법을 적용할 경우 SSD의 특징으로 인해 성능 감소가 발생 가능하다. 본 논문에서는 SSD를 위한 페이지 매핑 기반의 WADPM (Workload-Aware Dynamic Page-level Mapping Scheme) 기법을 제안한다. WADPM은 필요한 매핑 정보만 RAM에 상주하며 또한 매핑 정보의 적중률에 따라 매핑 정보의 크기를 동적으로 변경시킨다. 이로 인해 페이지 매핑 기법의 단점인 매핑 테이블의 크기가 큰 것을 예방한다. 실험을 통해 WADPM 기법은 하이브리드 매핑 기법에 비해 최대 3.5배의 성능 향상을 보이며, 매핑 테이블의 크기는 페이지 매핑 기법에 비해 최대 50%만 유지되는 것을 보인다.

키워드 : SSD, 낸드 플래시 메모리, FTL 매핑기법, 내부 단편화

*Abstract* The NAND flash memory based SSDs are considered to replace the existing HDDs. To maximize the I/O performance, SSD is composed of several NAND flash memories in parallel. However, to adopt the hybrid mapping scheme in SSD may cause degradation of the I/O performance. In this paper, we propose a new mapping scheme for the SSD called WADPM. WADPM loads only necessary mapping information into RAM and dynamically adjusts the size of mapping information in the RAM. So, WADPM avoids the shortcoming of page-level mapping scheme that requires too large mapping table. Performance evaluation using simulations shows that I/O performance of WADPM is 3.5 times better than the hybrid-mapping scheme and maximum size of mapping table of WADPM is about 50% in comparison with the page-level mapping scheme.

Key words : SSD, NAND flash memory, FTL mapping scheme, internal fragmentation

## 1. 서 론

낸드 플래시 메모리 기반 SSD(Solid State Disk)는 하드 디스크 드라이브(HDD)를 대체할 블록 장치로 주목 받고 있다[1]. SSD는 다수의 낸드 플래시 메모리를 병렬적으로 구성하며, 이에 따라 낸드 플래시 메모리가 가지는 저전력, 높은 랜덤 접근 성능 및 충격에 강한 장점들을 갖는다. 또한 지우기 전 기록할 수 없다는 점(erase-before-write)과 읽기, 쓰기 및 지우기 연산값이 다르다는 특징을 포함한다. SLC(Single Level Cell) 낸드 플래시 메모리의 경우 외부와의 대역폭은 25ns/byte (40MB/sec)이다[2]. 현재 SATA 3.0 인터페이스는 최대 6Gb/s의 속도를 가지며[3], 단일 낸드 플래시 메모리로는 해당 속도를 지원할 수 없다. 이를 위해 SSD는 내부에 다수의 낸드 플래시 메모리를 병렬적으로 구성

· 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음(NIPA-2010-(C1090-1021-0008))

† 학생회원 : 성균관대학교 전자전기컴퓨터공학과
chaosken@ece.skku.ac.kr
hjcho@ece.skku.ac.kr
†† 종신회원 : 성균관대학교 정보통신공학부 교수
yieom@ece.skku.ac.kr

논문접수 : 2010년 3월 9일
심사완료 : 2010년 5월 4일



정보과학회논문지: 시스템 및 이론 제37권 제4호(2010.8)

하며, 속도 향상을 위한 다양한 방법을 통해 병렬성(parallelism)을 최대화한다. 이 중 한가지가 논리 주소 스트라이핑(address striping)이며, 이를 통해 I/O 수행 시 최대한 많은 수의 낸드 플래시 메모리를 동작 시킨다. full-striping으로 논리 주소 영역을 지정하게 되면 SSD 내부를 구성하는 낸드 플래시 메모리 전체를 단일 논리 주소 영역으로 구성하고 이에 따라 데이터 기록 범위가 전 영역에 펼쳐지게 된다[4,5].

플래시 메모리는 기존 블록 장치와는 다른 특징을 가지며, 이에 따라 FTL(Flash Translation Layer)이라는 시스템 소프트웨어가 필요하다. FTL은 운영체제와 낸드 플래시 메모리 사이에 있는 시스템 소프트웨어로써 운영체제로부터 요청된 주소를 낸드 플래시 메모리내 물리 주소와 매핑하는 역할을 수행한다. 낸드 플래시 메모리를 위한 매핑 기법은 페이지 매핑(page-level mapping), 블록 매핑(block-level mapping), 하이브리드 매핑(hybrid mapping) 3가지로 구분할 수 있다[6]. 페이지 매핑 기법은 페이지 단위로 매핑 정보를 구성하며, 성능은 가장 좋지만 매핑 테이블의 크기가 커진다는 단점을 갖는다. 블록 매핑 기법은 블록 단위로 매핑 정보를 구성하며 페이지 매핑 기법에 비해 매핑 정보는 작지만 블록 내 페이지 갱신에 따른 오버헤드가 커진다는 단점이 있다. 하이브리드 매핑은 블록 매핑 기법과 페이지 매핑 기법을 혼합한 기법이며, 낸드 플래시 메모리 공간 내 일부를 데이터 갱신을 전담하는 로그 버퍼(log buffer)로 활용하게 된다. 따라서 하이브리드 매핑 기법을 사용하게 되면 플래시 메모리 공간은 데이터 블록과 데이터 블록 내 갱신데이터를 기록하는 로그 블록으로 구성되며 데이터 블록은 블록 매핑 기법으로, 로그 블록은 페이지 매핑 기법으로 매핑 정보를 관리한다. 이 때 로그 블록의 개수는 페이지 매핑 정보 크기에 따라 달라진다. 하이브리드 매핑 기법에서 처음 데이터 기록 요청이 발생하면 데이터 블록을 할당하여 기록하고 이 후 데이터 갱신이 발생하면 해당 데이터 블록에 로그 블록을 할당하여 갱신 데이터를 기록한다. 이 후 로그 버퍼 공간이 부족하게 되면 데이터 블록과 로그 블록을 병합하여 새로운 로그 버퍼 공간을 생성한다. 현재 가장 많이 활용되는 매핑 기법은 하이브리드 매핑 기법이며, 이에 따라 많은 연구가 진행되었다[7-10].

하지만 하이브리드 매핑 기법을 사용할 경우 해당 작업 부하의 패턴에 따라 성능이 감소할 수 있다. 만약 공간적 지역성(spatial locality)이 큰 작업 패턴을 갖는 데이터를 기록해야 하는 경우 작업 부하의 특성에 따라 갱신 요청이 많이 발생하게 된다. 로그 블록 공간은 한정되어 있기 때문에 갱신 데이터가 로그 블록 공간을 넘어설 수도 있다. 이 경우 빈번한 로그 블록의 병합 연산으로 인해 전체 성능이 감소할 수 있다.

또한 낸드 플래시 메모리는 첫 번째 페이지부터 순서대로 기록해야 하는 제약 조건을 가지고 있는데[2] 하이브리드 매핑 기법을 사용할 경우 내부 단편화(internal fragmentation)로 인해 블록 내 빈 페이지가 있음에도 불구하고 데이터를 기록할 수 없는 경우가 발생하기도 한다. 만약 랜덤 패턴을 가지는 데이터를 기록하게 되는 경우 내부 단편화의 문제가 커질 수 있다. 더구나 앞에서 언급한 full-striping 방식으로 SSD 내부 논리 주소 공간을 지정하게 된다면 내부 단편화 문제가 더욱 심각해질 수 있다. 이때 SSD 내부 낸드 플래시 메모리 공간에 단편화가 발생할 수록 저장 공간이 낭비되어 블록 병합 연산이 더 자주 발생할 수 있으며, 이에 따라 성능 저하가 발생하게 된다. 결국 공간적 지역성이 크거나 랜덤 패턴을 가진 데이터를 기록할 경우 SSD의 높은 성능을 위해서는 페이지 매핑을 사용하는 것이 적합하지만 앞에서 언급한 것처럼 페이지 매핑 기법은 매핑 정보가 커진다는 단점을 가지고 있다.

본 논문에서는 이러한 점을 고려하여 SSD에서 하이브리드 매핑 기법을 사용할 경우 발생 가능한 성능 저하 문제를 살펴보고 해당 문제를 해결하기 위한 페이지 매핑 기반의 WADPM(Workload-Aware Dynamic Page-level Mapping) 기법을 제안한다. WADPM 기법에서는 필요한 경우에만 매핑 정보를 RAM에 로딩하게 된다. 또한 I/O 수행 중 매핑 정보의 크기를 동적으로 변경할 수 있도록 하여 RAM에 상주하는 매핑 테이블의 크기를 작업 부하의 주소 접근 패턴에 따라 변경할 수 있도록 한다.

본 논문의 구성은 다음과 같다. 2장에서는 낸드 플래시 메모리 및 SSD 내부 구조를 설명하고 3장에서는 앞에서 언급한 내부 단편화 문제점에 대해 설명한다. 4장에서는 본 논문의 제안 기법인 WADPM에 대해 설명한다. 5장에서는 WADPM 기법을 검증하기 위한 실험 결과를 보이며 6장에서 본 논문의 결론을 맺는다.

## 2. 낸드 플래시 메모리 및 SSD 내부 구조

단일 낸드 플래시 메모리는 제조회사 및 종류에 따라 4GB 또는 8GB의 크기를 가진다. 그림 1에서 낸드 플래시 메모리의 구조를 보인다[2].

그림 1에서 보이는 것처럼 SLC 낸드 플래시 메모리의 경우 크기는 4GB이며 내부는 다이(die), 플레인(plane), 블록(block), 페이지(page)로 구성된다. 또한 내부의 병렬성을 위해 2개의 다이를 사용하며, 다이들 간에는 병렬적인 I/O를 수행할 수 있다. 또한 다이는 플레인들을 포함하며 플레인은 다수의 블록들로, 블록은 다수의 페이지들로 구성된다.

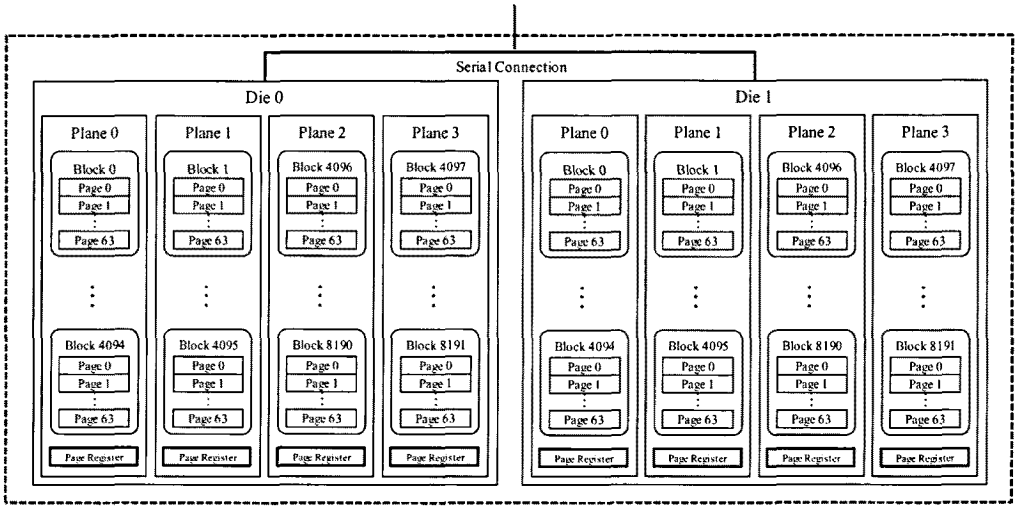



그림 1 플래시 메모리 내부 구조

I/O 수행시 운영체제는 해당 데이터를 논리 블록주소(LBA: Logical Block Address)를 통해 접근하게 된다. SSD 내부의 FTL은 현재 SSD 내부의 페이지 크기에 따라 다수의 LBA를 하나의 논리 페이지 번호로(LPN: Logical Page Number) 변환한다. 만약 LBA의 단위가 512 bytes이고 SSD의 페이지가 2KB라면 4개의 LBA를 하나의 페이지로 구성하게 된다. 또한 SSD 내부의 낸드 플래시 메모리들을 최대한 병렬적으로 동작시키기 위해 LPN을 SSD 전 영역에 스트라이핑(striping)한다. 그림 2에서 SSD 내부의 LPN 할당을 보인다.

그림 2에서 보이는 것처럼 SSD는 내부에 가지고 있는 모든 물리 플래시 메모리를 하나의 논리 플래시 메모리로 구성하며, 낸드 플래시 메모리내 다이와 플레인의 개수는 각각 2개라고 가정한다. 그림 2에서 데이터 기록 요청이 발생하여 LPN 0~7까지 8개의 페이지에 데이터를 기록해야 한다고 가정한다. 그림 2(a)처럼 LPN을 스트라이핑 하지 않는다면 8개의 페이지를 기록할 때 병렬성을 활용할 수 없게 된다. 즉, 0번 낸드 플래시 메모리의 0번 다이만 동작하여 기록 요청을 수행한다. 하지만 그림 2(b)처럼 낸드 플래시 메모리 전 영역에 걸쳐 LPN을 스트라이핑 한다면 낸드 플래시 메모리간 그리고 다이간 병렬성을 활용할 수 있게 된다. 이때는 4개의 다이가 동시에 동작하기 때문에 no-striping에 비해 이론적으로 기록 속도를 4배 증가시킬 수 있다[4].

SSD의 주소 정보를 변환하기 위해 페이지 매핑 기법을 사용할 경우 데이터 기록 요청이 발생하면 빈 페이지에 데이터를 기록한 후 해당 물리 페이지 번호(PPN: Physical Page Number)와의 매핑 정보를 유지한다. 만약 하이브리드 매핑 기법을 사용한다면 LPN과 PPN의 매핑 정보대신 데이터 블록은 블록 매핑 정보를, 로그 블록은 페이지 매핑 정보를 유지하게 된다. 이 때 LPN을 이용하여 논리 블록 번호(LBN: Logical Block

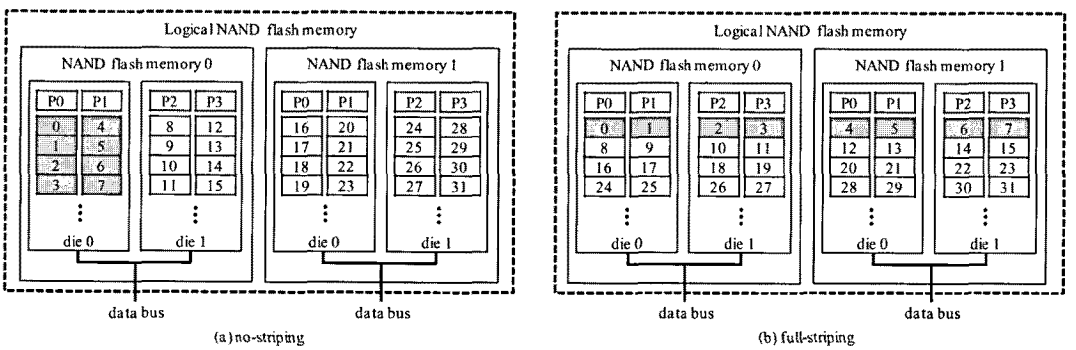



그림 2 SSD 내부 논리 주소 할당

Number)와 페이지 오프셋을 계산하고 처음 기록 요청이 발생된 데이터라면 데이터 블록에 기록한 후 LBN과 물리 블록 번호(PBN: Physical Block Number)의 매핑 정보를 유지한다. 이후 블록 내 데이터 갱신이 발생하면 이미 블록에 기록되어 있는 데이터를 무효화(invalidate)한 후 로그 블록을 할당하여 갱신 데이터를 기록한다. 그러나 앞에서 살펴본 바와 같이 SSD 내부의 LPN은 물리 페이지 위치에 상관없이 고정시켜 사용하기 때문에 LBA를 이용하여 LPN을 계산할 수 있으며 따라서 LBA와 LPN의 매핑 정보는 유지할 필요가 없다. 결국 LPN과 PPN 또는 LBN과 PBN과의 정보만 유지하면 된다.

## 3. 하이브리드 매핑 기법을 SSD에 적용할 경우의 문제점

앞에서 언급한 것처럼 SSD에 하이브리드 매핑 기법을 사용하게 되면 작업 부하의 특성에 따라 두 가지의 성능 저하 문제가 발생할 수 있다. 첫 번째는 공간적 지역성이 높은 작업 부하의 기록 요청으로 인해 데이터 갱신이 많이 발생하게 되는 경우이다. 이 경우에는 데이터 블록보다는 로그 블록에 데이터를 더 많이 기록하게 된다. 데이터 갱신이 발생하게 되면 빈 블록을 로그 블록으로 할당하여 데이터를 기록하게 된다. 이 때 로그 블록으로 할당할 수 있는 빈 블록이 많다고 해도 로그 블록의 개수가 한정되어 있기 때문에 전체 로그 블록의 개수를 넘어선 데이터 갱신이 발생하게 되면 병합(merge) 연산을 수행해야 한다. 병합 연산은 낸드 플래시 메모리의 세 가지 연산 중 연산 값이 가장 큰 블록 삭제 연산을 포함하게 되며, 이에 따라 전체 성능의 저하가 발생할 수 있다.

두 번째는 작업 부하가 랜덤 패턴으로 구성된 경우이다. 낸드 플래시 메모리는 자체적으로 데이터를 기록할 때 해당 블록의 첫 번째 페이지부터 순차적으로 기록해야 하는 제약 조건이 있다. 예를 들어 블록이 삭제된 직후 10번째 페이지에 데이터가 기록되었다면 블록을 다시 삭제하기 전까지는 0~9번째 페이지에 데이터를 기록할 수 없다. 만약 페이지 매핑 기법을 사용하게 되면 앞에서 언급한 문제는 발생하지 않는다. 하지만 블록 매핑 또는 하이브리드 매핑 기법을 사용하게 되면 기록해야 하는 페이지 오프셋이 빈 페이지라고 할지라도 기록할 수 없는 경우가 발생할 수 있으며 이에 따라 데이터 블록의 활용률(utilization)이 낮아질 수 있다. 더구나 SSD 내부 병렬성을 높이기 위해 논리 주소 영역을 full-striping 하게 될 경우 no-striping에 비해 이러한 문제가 더 커질 수 있다(그림 3).

그림 3처럼 p16~p23까지 8개 페이지에 대한 기록 요청이 발생했다고 가정한다. no striping의 경우 그림 3(a)와 같이 1번 플래시 메모리의 0번 다이에 있는 첫 번째 페이지부터 데이터가 기록된다. 하지만 full-striping의 경우 SSD를 구성하고 있는 전체 영역에 걸쳐 데이터가 기록되기 때문에 앞에서 언급한 낸드 플래시 메모리의 제약 사항에 따라 p0~p15가 빈 공간임에도 불구하고 8개의 블록이 지워지기 전까지 데이터를 기록할 수 없게 된다.

앞에서 언급한 두 가지 문제점을 확인하기 위해 SSD 시뮬레이터를 이용하여 성능 측정을 수행하였다. 해당 시뮬레이터는 SSDsim[4] 기반으로 구현 하였으며 4개의 작업 부하(I/O trace)를 이용하여 실험하였다. 표 1에서 실험에 사용한 4개 작업부하를 보인다.

표 1에서 보이는 것처럼 Bonnie++[11]와 IOzone[12]은 해당 벤치마크 프로그램을 이용하여 추출한 작업부하이며, 공간적 지역성이 큰 순차 패턴을 가지고 있다. Internet explorer는 윈도우XP 데스크톱 상에서 웹 서핑을 수행하면서 추출한 작업부하로 대부분 랜덤 패턴으로 구성되어 있다. PC application 작업 부하는 윈도우XP 데스크톱 상에서 웹 브라우징, 문서 작성, 게임

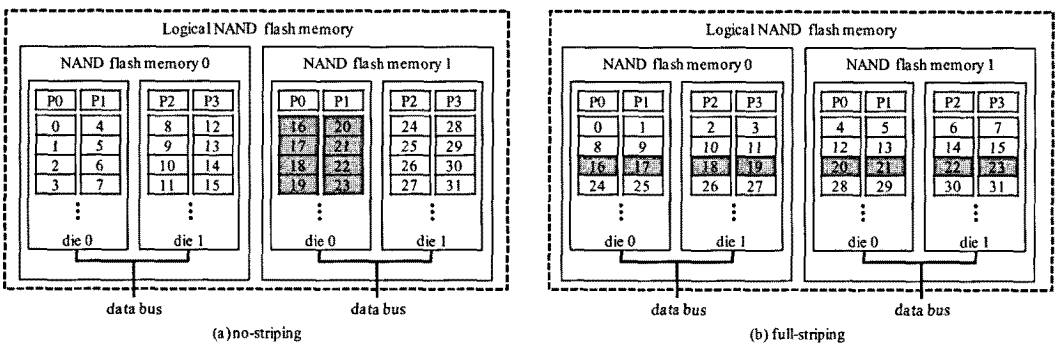



그림 3 full-striping을 사용할 경우 발생가능한 문제점

표 1 실험에 사용한 작업 부하

| 순번 | 이름 | 총 작업량 | 평균 요청 크기 | 주소 접근 패턴 |
|---|---|---|---|---|
| 1 | Bonnie++ | 884.38 MB | 33.06 KB | 순차 |
| 2 | IOzone | 3.42 GB | 58.68 KB | 순차 |
| 3 | Internet explorer | 356.77 MB | 14.21 KB | 랜덤 |
| 4 | PC application | 1.62 GB | 13.53 KB | 랜덤 + 순차 |

표 2 실험에 사용한 SSD 및 낸드 플래시 메모리 사양

| SSD 크기 | 4GB | 읽기 속도 | 25 us |
|---|---|---|---|
| 플레인 크기 | 256 MB | 쓰기 속도 | 200 us |
| 블록 크기 | 128 KB | 블록 지우기 속도 | 2 ms |
| 페이지 크기 | 2 KB | 플래시 메모리 버스 속도 | 40MB/sec |

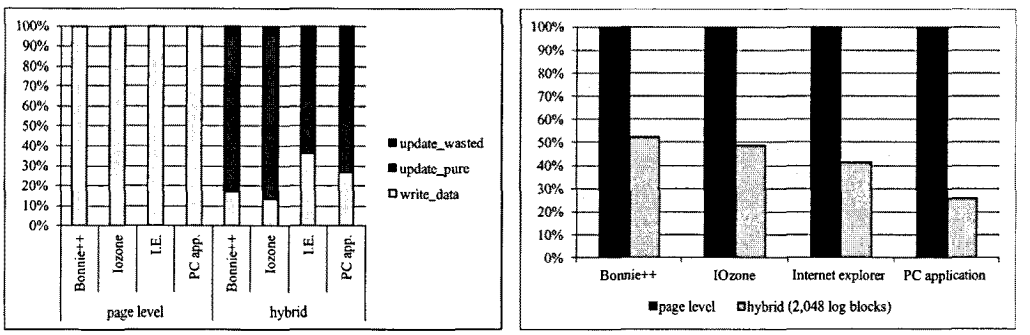
및 동영상 실행과 같은 실제 작업부하를 측정한 것이며, 랜덤과 순차 패턴을 동시에 포함하고 있다.

시뮬레이터 상에서 SSD의 크기는 작업 부하의 크기가 크지 않은 점을 고려하여 4GB로 설정하였으며, SSD내부에는 1GB 크기를 갖는 낸드 플래시 메모리 4개를 병렬적으로 구성하였다. 각 낸드 플래시 메모리는 내부에 두 개의 다이를 가지고 있으며, 각 다이는 다시 두 개의 플레인을 포함한다. 따라서 I/O를 수행할 때 4개의 낸드 플래시 메모리의 병렬성을 활용하여 최대 총 8개의 플레인을 동시에 동작시킬 수 있다. 표 2에서 실험에 사용한 낸드 플래시 메모리의 사양을 보인다.

성능 측정을 위해 SSD에는 페이지 매핑과 하이브리드 매핑 기법을 적용하여 비교 실험하였으며 하이브리드 매핑은 FAST[8] 기법을 적용하였다. 또한 하이브리드 매핑 기법에는 로그 블록의 개수를 2,048(256 MB)개로 설정하여 실험하였다.

그림 4에서는 페이지 매핑과 하이브리드 매핑의 비교를 보인다. 페이지 매핑의 경우 데이터 블록과 로그 블록의 구분이 없지만 하이브리드 매핑의 경우 데이터가 기록되는 위치는 데이터 블록과 로그 블록으로 구분된다. 데이터가 처음 데이터 블록에 기록(*write_data*)된 후 갱신이 발생하여 해당 데이터가 로그 블록에 기록될 때의 경우를 다시 두 가지로 구분하였다. 첫 번째는 데이터의 순수한 갱신으로 인한 기록(*update_pure*)이고 두 번째는 데이터 블록이 빈 페이지 임에도 불구하고 낸드 플래시 메모리의 제약사항으로 인해 로그 블록에 기록(*update_wasted*)되는 경우이다. 실험을 통해 전체 데이터를 기록할 때 수행하는 *write_data*, *update_pure*, *update_wasted*의 비율을 측정하였다.

그림 4(a)를 보면 페이지 매핑의 경우 로그 블록의 개념이 존재하지 않기 때문에 처음 기록되는 데이터와 갱신 데이터의 구분 없이 모든 기록은 write_data가 된다. 하지만 하이브리드 매핑을 사용할 때 갱신 데이터는 로그 블록에 기록해야 하는데 이 때 작업 부하의 주소 접근 패턴에 따라 update_pure와 update_wasted의 비율이 달라진다. Bonnie++와 IOzone 작업 부하의 경우 공간 지역성이 매우 크고 순차적인 패턴을 가지고 있기 때문에 데이터 블록에는 순서대로 기록하게 되며 이에 따라 순수한 데이터 갱신이 많이 발생한다. 하지만 Internet explorer와 PC app. 작업 부하의 경우 랜덤한 주소 접근 패턴을 많이 포함하고 있는데, 이에 따라 처음 데이터 블록에 데이터를 기록할 때 첫 번째 페이지 오프셋부터 데이터를 기록하지 못하고 중간에 빈 공간을 만들게 된다. 이에 따라 다음 데이터 기록 시 블록 내 빈 페이지가 존재함에도 불구하고 오프셋 위치에 따라 갱신 데이터로 간주되는 경우가 발생한다. 따라서 Bonnie++와 IOzone 작업 부하는 update_pure의 비율이 많고, Internet explore와 PC application 작업 부하




(a) 데이터 기록 비율 (b) throughput 비율

그림 4 페이지 매핑 기법과 하이브리드 매핑 기법 비교

에서는 update_wasted의 비율이 update_pure보다 조금 더 많다.

그림 4(b)에서 전체 I/O 처리량(throughput)을 보이며, 하이브리드 매핑을 사용한 처리량이 페이지 매핑을 사용한 처리량에 비해 얼마만큼의 비율을 갖는지 측정하였다. 하이브리드 매핑을 사용했을 때의 성능은 페이지 매핑을 사용한 경우와 비교하여 25%~50%의 성능만 유지되는 것을 알 수 있다. 따라서 페이지 매핑을 사용하는 것이 전체 성능 향상에 가장 좋다는 것을 알 수 있다.

하지만 앞에서 언급한 것처럼 페이지 매핑 기법은 매핑 정보가 커진다는 단점을 가지고 있다. 이 문제점을 해결하기 위해 WADPM 기법은 전체 매핑 정보를 필요한 경우만 RAM(Random Access Memory)에 로딩할 수 있도록 한다.

## 4. WADPM(Workload-Aware Dynamic Page-level Mapping) 기법

본 장에서는 본 논문의 제안 기법인 WADPM 기법에 대해 설명한다. WADPM은 페이지 매핑 기법 기반이며, 페이지 매핑 기법의 단점을 보완하기 위해 다음과 같이 세 가지 최적화 기법을 사용한다.

- 전체 매핑 정보 중 필요한 매핑 정보만 동적으로 RAM에 로딩한다.
- 플레인 단위로 매핑 테이블을 구성한다.
- 매핑 정보의 적중률에 따라 RAM에 유지되는 매핑 정보의 양을 조절한다.

WADPM 기법에서는 필요한 매핑 정보만 RAM에 유지하도록 한다. 만약 전체 매핑 정보를 메모리에 유지한다면 LPN에 해당하는 부분은 인덱스로 접근할 수 있으며 이에 따라 PPN이 차지하는 공간만 메모리를 사용하게 된다. 하지만 매핑 테이블의 일부를 RAM에 상주시키게 되면 PPN 뿐만 아니라 LPN의 정보 역시 RAM에 상주해야 한다. 결국 매핑 테이블의 한 엔트리의 크기가 두 배가 된다. WADPM 기법에서는 이 문제를 완화하기 위해 매핑 테이블의 주소 영역을 SSD 전체로 하지 않고 플레인 내로 한정한다. 이 경우 매핑 테이블의 엔트리 수는 플레인에 속한 페이지의 수가 되며, 플레인 내의 논리 페이지 순서(LPI: Logical Page Index)와 물리 페이지 순서(PPI: Physical Page Index)로 매핑 정보를 유지한다.

그림 5에서 기존의 페이지 매핑 테이블과 플레인 단위 매핑 테이블의 차이를 보인다.

SSD는 그림 5(a)와 같이 2개의 낸드 플래시 메모리로 구성되고, 낸드 플래시 메모리는 4개의 플레인으로 구성되며 각 플레인은 1개의 블록으로, 각 블록은 4개의 페이지를 가지고 있다고 가정한다. 또한 플레인 번호는 순서대로 넘버링 된다고 가정한다. 그림 5(b)에서 페이지 매핑 기법으로 매핑 테이블을 구성한 것을 보인다. SSD에 포함된 모든 페이지에 대한 정보를 유지하므로 테이블의 전체 엔트리 수는 전체 페이지 수가 된다. 그림 5(c)에서 플레인 단위로 매핑 테이블을 구성하므로 테이블당 엔트리 수는 플레인에 속한 페이지 수가 된다. 이때 SSD에 데이터를 기록하기 위한 데이터 주소 변환은 LBA→LPN→LPI→PPI→PPN의 순서대로 이루어지며, 표 3에서 주소 변환 수식을 위한 기호를 보인다.

표 3 주소 변환에 사용되는 기호

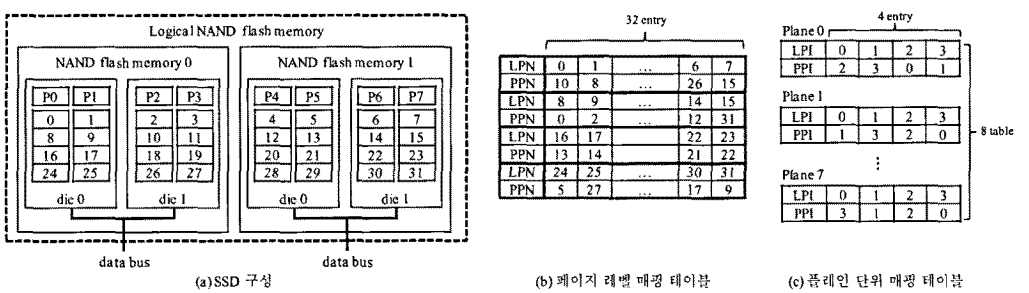
| 기호 | 설명 |
|---|---|
| $N_{sectors}$ | 페이지당 섹터의 개수 (the number of sectors per page) |
| $N_{planes}$ | SSD내 전체 플레인 개수 |
| $PN$ (LPN) | 임의의 LPN(Logical Page Number)이 속한 플레인 번호 |

표 3의 기호에 따라 주소의 변환 수식은 다음과 같다.

$$LPN = LBA \ / \ N_{sectors} \quad (1)$$

$$PN(LPN) = LPN \ \% \ N_{planes} \quad (2)$$

$$LPI = \lfloor LPN / N_{planes} \rfloor \quad (3)$$




(a) SSD 구성 (b) 페이지 레벨 매핑 테이블 (c) 플레인 단위 매핑 테이블

그림 5 페이지 매핑 테이블과 플레인 단위 매핑 테이블 비교

식 (1)에 따라 운영체제로부터 요청된 LBA가 페이지 주소로 변경된다. 이 값은 LBA를 페이지에 포함되는 섹터의 수로 나누어 줌으로서 간단히 구할 수 있다. 또한 SSD에서 full-striping을 사용할 경우 SSD에 속한 모든 플레인에 주소 영역이 나누어 진다. 따라서 LPN에 $N_{planes}$를 나누어 LPI를 구할 수 있고, LPN과 $N_{planes}$의 나머지 연산으로 LPI가 속한 플레인 번호를 구할 수 있다. 이후 LPI에 PPI를 매핑하여 주소 정보를 저장한다. 그림 6에서 LPI를 이용하여 PPI를 얻는 알고리즘을 보인다.

```
GetPhysicalPageIndex( LBA )
{
    LPN = LBA / Nsectors;
    PN(LPN) = LPN % Nplaens;
    LPI = ⌊LPN / Nplanes⌋;

    if( LPI is not found in the mapping table )
    {
        get free page from PN(LPN);
        update the mapping table;
    }

    PPI = get PPI information from the mapping table;

    return PPI;
}
```

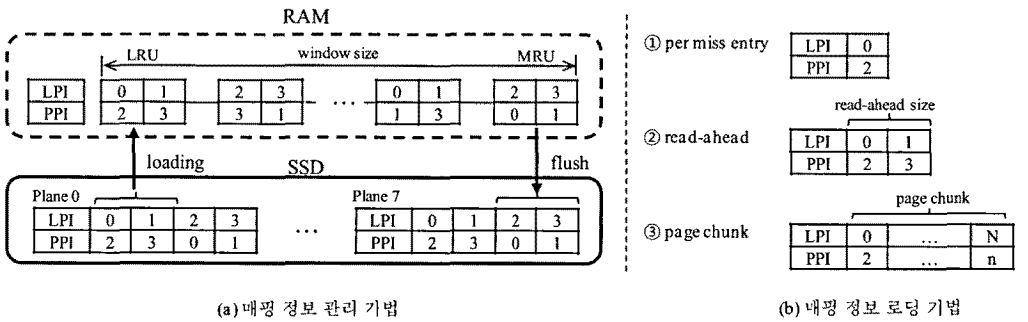
그림 6 LPI를 이용하여 PPI를 획득하는 의사코드

그림 6의 알고리즘은 LBA를 이용하여 LPI를 구한 후 LPI와 매핑된 PPI를 획득하는 과정이다. 만약 할당된 PPI가 없다면 LPI가 속한 플레인에서 빈 페이지를 하나 할당 받아 매핑한다.

이와 같이 WADPM 기법에서는 LPI와 PPI만을 이용하여 매핑 정보를 유지한다. 이렇게 하면 매핑 정보 엔트리당 유지해야 하는 비트수가 감소하기 때문에 전체 매핑 정보가 작아지게 된다. 또한 LPI와 매핑된 PPI의 페이지는 동일한 플레인에 위치하고 있다. 따라서 데이터 이동시 항상 같은 플레인의 페이지로 이동하게 되므로 플래시 메모리의 copy-back 동작을 항상 보장 할 수 있다. 낸드 플래시 메모리는 이미 기록된 페이지는 지우기 전에 다시 기록할 수 없는 단점이 있다. 때문에 주기적으로 가비지 컬렉션을 수행하여 사용 가능한 페이지를 만들어 주어야 한다. 가비지 컬렉션은 희생 블록을 선택하여, 희생블록의 유효한 페이지를 새 블록으로 옮기는 작업을 수행한다. 이후 희생 블록을 지워 사용 가능한 페이지를 만든다. 이때, 만약 일반적인 페이지 매핑 기법을 사용 한다면 유효한 페이지가 옮겨질 페이지가 동일한 플레인에 위치하지 않을 수 있다. 따라서 페이지 매핑 기법에서는 copy-back 연산을 사용하지 못할 수도 있다. 하지만 플레인 단위 매핑 기법을 사용한다면 데이터가 기록될 플레인이 고정된다. 데이터는 특정 플레인에 속한 페이지들 중 한곳에 기록되며, 데이터가 이동할 때도 같은 플레인의 페이지로 옮겨진다. 즉, 플레인 단위 매핑 기법에서는 유효한 페이지를 옮기는 작업이 모두 copy-back 연산을 이용하여 수행된다. 따라서 플레인 단위 매핑 기법을 사용하면 가비지 컬렉션 동작에 필요한 데이터 이동 비용을 최소화하여 플래시 메모리의 성능 증가에 도움이 된다.

또한 작업 부하의 주소 접근 패턴에 따라 I/O 요청 데이터의 지역성(locality)이 달라질 수 있으며, 이에 따라 RAM에 유지해야 할 매핑 정보가 달라질 수 있다. 이러한 점을 고려하여 WADPM 기법에서는 매핑 테이블 전체를 RAM에 유지하지 않고 필요한 부분만을 RAM에 유지하도록 한다. 또한 고정된 크기의 매핑 정보를 RAM에 유지하는 것이 아니라 매핑 정보의 적중률에 따라 RAM에 상주하는 매핑 테이블의 크기를 동적으로 변경할 수 있도록 한다. 그림 7에서 WADPM의 동적 로딩 기법을 보인다.

그림 7(a)와 같이 WADPM 기법에서는 페이지 매핑 정보가 SSD를 구성하는 낸드 플래시 메모리에 저장되어 있다고 가정하며 해당 매핑 테이블 중 일부만 RAM에 상주하게 된다. 따라서 한번에 얼마만큼의 매핑 정보를 RAM에 상주하도록 할 지와 어떠한 정책에 따라 매핑 정보를 RAM에 유지하도록 할 지에 따라 전체 성능




(a) 매핑 정보 관리 기법 (b) 매핑 정보 로딩 기법

그림 7 WADPM의 동적 로딩 기법

이 달라지게 된다. 플래시 메모리에 저장되어 있는 매핑 정보를 RAM으로 읽어 들이는 방법은 그림 7(b)와 같이 3가지로 구분할 수 있다. 첫 번째는 miss 엔트리만 읽어 들이는 방법(per miss entry), 두 번째는 miss 엔트리를 포함하여 몇 개의 엔트리를 더 읽어 들이는 방법(read-ahead), 세 번째는 miss 엔트리에 대한 주소 정보가 포함된 플래시 메모리 페이지 내에 저장된 모든 매핑 정보를 읽어 들이는 방법(page chunk)이다. 그림 7(b)는 플레인 0번의 0번 페이지에 대해 miss가 발생했을 때 RAM으로 매핑 정보를 읽어 들이는 세가지 방법의 차이를 보인다. WADPM 기법에서는 miss 엔트리에 대한 주소 정보가 포함된 플래시 메모리 페이지의 모든 매핑 정보를 읽어 들이는 방법(page chunk)을 사용한다. 이는 낸드 플래시 메모리의 특성상 읽기 동작이 페이지 단위로 이루어지기 때문이다. RAM으로 읽어진 매핑 정보는 LRU 기법에 따라 관리된다. 또한 설정에 따라 RAM에 상주하게 되는 매핑 정보의 크기(window size)를 고정할 수도 있고, SSD 동작 중 설정 값 이내에 동적으로 크기를 변경하도록 할 수 있다. 그림 8에서 매핑 정보의 크기를 조절하는 알고리즘을 보인다.

그림 8에서 RAM에 로딩된 매핑 정보의 적중률을 이용하여 매핑 테이블의 크기를 결정하는 알고리즘을 보인다. 알고리즘의 $\theta_{HR}$은 매핑 테이블 크기를 증가 혹은 감소시킬 때 기준으로 사용할 적중률이다. 또한 $\theta_{MT}$은 적중률이 높을 때 매핑 테이블 크기를 유지할 기간이다. 만약 매핑 정보 적중률이 $\theta_{HR}$ 이하이면 적중률을 높이기 위해 매핑 정보를 저장할 테이블의 크기를 증가시킨다. 또한 매핑 정보 적중률이 $\theta_{HR}$보다 높은 상태로 $\theta_{MT}$ 이상 유지되면 매핑 정보의 크기를 감소시킨다. 이때 현재 매핑 테이블에 로딩된 매핑 정보의 크기와 감소된 후의 매핑 테이블의 크기를 비교하여 현재 로딩된 매핑 정보의 크기가 더 크다면 로딩된 매핑 정보의 일부를 RAM에서 내보낸다.

```
AdjustWindowSize( void )
{
    θHR := Hit Ratio threshold;
    θMT := Maintenance threshold;

    if( currentHitRatio < θHR )
    {
        increase size of the mapping table;
    }
    else // currentHitRatio ≥ θHR
    {
        if( currentMaintenanceCount < θMT )
        {
            increase MaintenanceCount;
        }
        else  // currentMaintenanceCount ≥ θMT
        {
            if( currentMapSize > decreasedMapSize )
            {
                flush map data;
            }

            decrease map size;
            MaintenanceCount = 0;
        }
    }
}
```

그림 8 매핑 테이블 크기를 조절하는 의사코드

## 5. 실험결과

### 5.1 매핑 정보 읽기 방법 비교

그림 9에서 매핑 정보를 낸드 플래시 메모리에서 읽어 들이는 방법에 따른 매핑 정보의 적중률을 보인다.

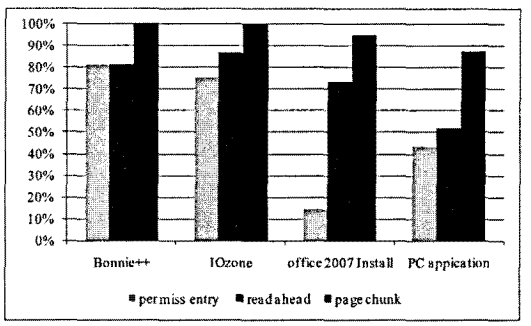



그림 9 매핑 정보 읽기 방법에 따른 적중률(hit ratio)

그림 9에서 페이지 단위로 매핑 정보를 읽어 RAM에 상주시키는 것이 가장 높은 적중률을 보이는 것을 알 수 있다. 이는 작업 부하에 따라 차이는 있지만 일반적으로 작업 부하가 집약성(locality)을 갖기 때문이다. 또한 플래시 메모리의 최소 읽기 단위가 페이지이므로 세 가지 방법의 비용은 모두 동일하다. 따라서 페이지 단위로 매핑 정보를 읽는 것이 가장 효율적임을 알 수 있다.

### 5.2 정적 매핑 테이블 크기에 따른 성능 비교

본 논문의 제안 기법인 WADPM의 성능을 검증하기 위해 RAM에 상주하는 매핑 테이블의 크기를 정적으로 변경하여 전체 성능을 측정하였다. I/O 수행 시 해당 데이터의 매핑 정보가 RAM에 존재하지 않는다면 SSD에서 읽어와야 하며 따라서 전체 성능이 감소하게 된다. 그림 10에서 매핑 테이블 크기에 따른 성능상의 차이를 보인다.

성능 측정을 위해 매핑 테이블의 크기는 페이지 매핑 테이블 크기에 비례하여 1%부터 20%까지 증가시켰으며 표 4에서 실제 매핑 테이블의 크기를 보인다.

하이브리드 매핑 기법은 앞에서 언급한 문제로 인해 가장 성능이 좋지 않다. Bonnie++와 IOzone 작업 부하의 경우 공간적 집약성으로 인해 작은 크기의 매핑 테이블만 유지해도 페이지 매핑 기법을 사용한 것과 비교하여 성능의 차이가 없는 것을 알 수 있다. Internet

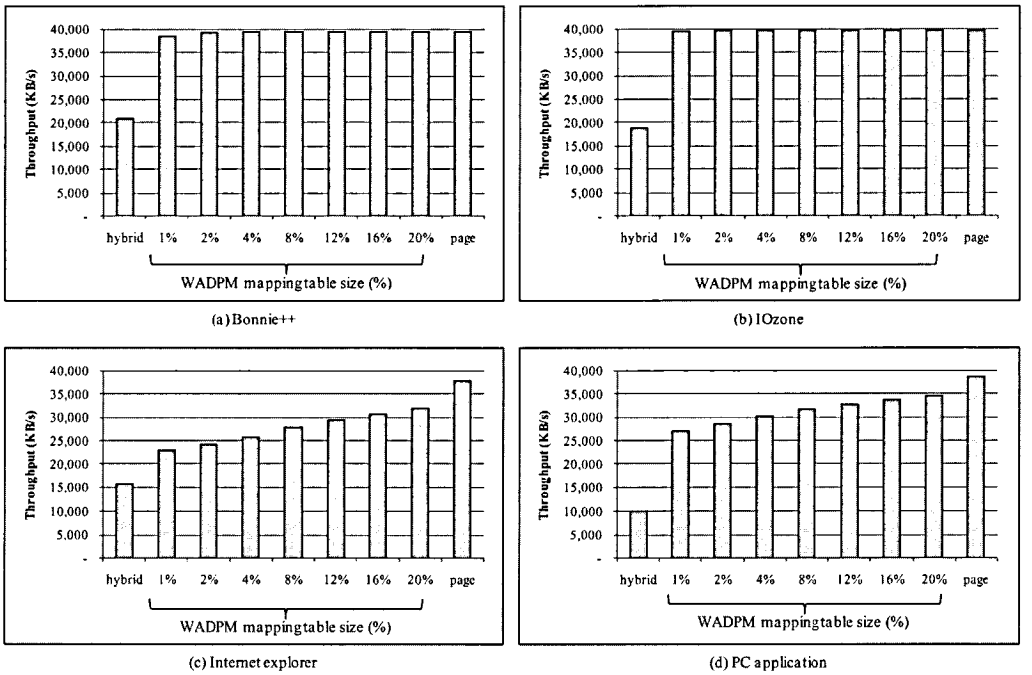



그림 10 매핑 테이블 크기에 따른 성능 비교

표 4 매핑 테이블 크기

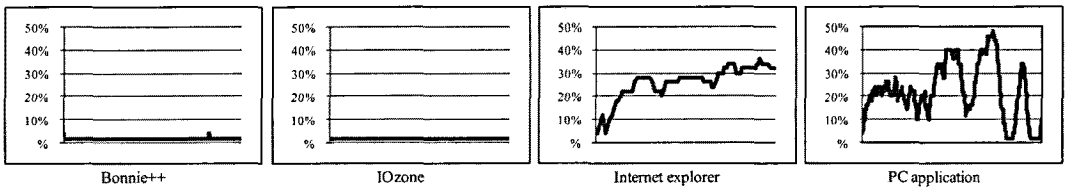
| hybrid mapping | WADPM | | | | | | | page mapping |
|---|---|---|---|---|---|---|---|---|
| | 1% | 2% | 4% | 8% | 12% | 16% | 20% | |
| 60KB | 54KB | 108 KB | 215 KB | 430 KB | 645 KB | 860 KB | 1,075 KB | 5.25 MB |

explorer와 PC application 작업 부하의 경우 매핑 테이블의 크기와 비례하여 전체 성능이 변화하는 것을 알 수 있다.

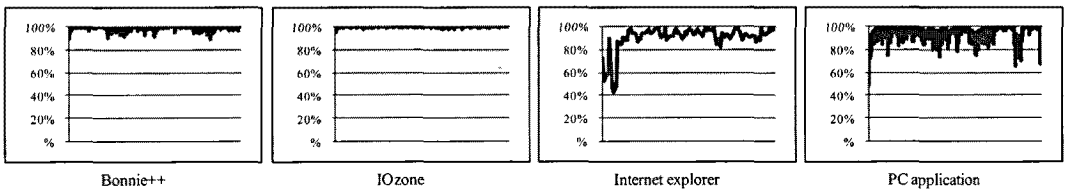
Internet explorer 작업 부하의 경우 대부분 랜덤 접근 데이터로 구성되어 있다. 하지만 작업 부하의 길이가 짧기 때문에 PC application 작업 부하에 비해 병합 연산이 적게 발생하며, 이에 따라 하이브리드 매핑 기법에서는 성능이 더 좋다. 하지만 작업 부하의 랜덤 패턴으로 인해 매핑 테이블 정보에 대한 빈번한 I/O가 발생하며 이에 따라 WADPM 기법에서는 전체적으로 성능이 낮아지는 것을 알 수가 있다. PC application 작업 부하의 경우 랜덤과 순차접근 데이터를 모두 포함하고 있으며, 작업 부하의 크기가 크기 때문에 병합 연산이 많이 발생하게 된다. 이로 인해 하이브리드 매핑을 사용할 경우 가장 좋지 않은 성능을 보인다. 실험 결과에 따라 지역성이 높은 데이터인 경우 페이지 매핑 테이블의 크기를 하이브리드 매핑 테이블 크기만큼 유지해도 전체 성능의 감소가 거의 없음을 알 수 있었다. 랜덤 패턴을 가진 작업 부하의 경우 매핑 테이블의 크기와 성능이 비례함을 알 수 있다. WADPM 기법에서 매핑 테이블의 크기를 페이지 매핑 기법의 20%로 유지할 경우 Internet explore 작업 부하의 경우 성능은 약 15%, PC application 작업 부하의 경우 약 10% 정도 성능이 감소하는 것을 보인다.

### 5.3 동적 매핑 테이블 크기에 따른 성능 비교

Bonnie++와 IOzone와 같이 공간적 지약성이 높은 작업 부하의 경우 전체 매핑 테이블에서 일부만 사용하기 때문에 RAM에 매핑 정보의 일부만 상주하면 된다. 반면 Internet explorer나 PC application와 같이 랜덤 접근 패턴과 주소 범위가 넓은 경우에는 RAM에 상주하는 매핑 테이블이 커져야 한다. 이에 따라 매핑 테이블의 크기는 매핑 정보의 적중률을 기준으로 동적으로 변경가능하도록 하였으며, 본 논문에서는 $\theta_{HR}$을 90%으로, $\theta_{MT}$를 5로 하였다. 즉, 매핑 정보의 적중률이 90% 이하로 저하되면 매핑 정보의 크기를 증가시키고, 적중률이 90% 이상으로 5회 이상 유지되면 매핑 정보의 크기를 감소시키게 된다(그림 11). $\theta_{HR}$과 $\theta_{MT}$의 상수 값은 실험을 통해 가장 좋은 성능을 나타내는 값으로 설정하였다.




(a) 매핑 테이블 크기의 변화




(b) 매핑 정보 적중률의 변화

그림 11 매핑 테이블 크기 변화 및 매핑 정보 적중율

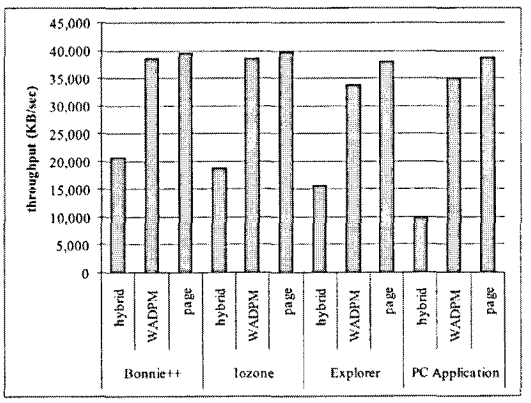



그림 12 매핑 방법에 따른 성능 비교

그림 11에서 보이는 것처럼 bonnie++와 iozone 작업 부하의 경우 전체 매핑 테이블의 2%만 유지되어도 매핑 정보의 적중률을 거의 100%가 된다. 하지만 internet explorer와 PC application 작업 부하의 경우 매핑 테이블의 크기는 internet explorer의 경우 35%, PC application 작업 부하의 경우 50%가지 증가한다.

그림 12에서 각 작업 부하의 매핑 방법에 따른 성능을 비교하였다.

페이지 매핑 기법의 테이블 크기와 비교했을 때 평균 매핑 테이블의 크기는 Bonnie++와 IOzone 작업 부하의 경우 2%가 되며, Internet explorer 작업부하는 26%, PC application 작업 부하는 22%가 된다. 이 때 성능을 비교해 보면 하이브리드 기법에 비해 최소 185%, 최대 350% 증가하였으며, 페이지 매핑 테이블에 비해 최소 2.6%, 최대 11%의 성능 감소를 보인다.

## 6. 결 론

SSD는 성능 향상을 위해 다수의 낸드 플래시 메모리를 병렬적으로 구성하며, 논리 주소 공간을 최대한 스트라이핑 한다. 또한 하이브리드 매핑 기법은 FTL의 매핑 기법으로 가장 많이 활용되는 매핑 기법이며, 이에 따라 활발한 연구가 진행되었다. 하지만 하이브리드 매핑 기법은 낸드 플래시 메모리 내부 단편화를 발생시키는 문제점을 가지고 있으며, 단일 낸드 플래시 메모리의 경우보다 SSD에서 더 큰 문제를 야기시킨다. 따라서 본 논문에서는 SSD 내부 환경에 적합한 페이지 매핑 기법을 제안하였다.

실제 SSD를 사용하게 되는 경우 다양한 작업 부하의 패턴이 혼합되어 SSD에게 I/O를 요청하게 될 것이다. 따라서 작업 부하의 패턴이 동적으로 변경될 경우 전체 매핑 테이블을 항상 RAM 상에 유지하는 것은 비 효율적이다. 매핑 정보를 동적으로 유지하는 WADPM 기법에서는 최대 11%의 성능감소를 통해 매핑 테이블의 크기를 페이지 매핑 기법에 비해 최소 1%부터 50%까지 가변적으로 유지할 수 있다. 이때 SSD의 성능은 하이브리드 매핑 기법 대비 최소 185%, 최대 350% 증가하였다.

## 참 고 문 헌

[1] T. Dinkelman, "SSDs A Shift in Data Storage," in *Proc. of Flash Memory Summit*, 2008.

[2] Samsung Corporation. K9XXG08XXM Flash Memory Specification. http://www.samsung.com.

[3] The serial ATA international organization, http://www.serialata.org.

[4] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse and R. Panigrahy, "Design

Tradeoffs for SSD Performance," in *Proc. of USENIX Technical Conference*, 2008.
[5] J-Y. Shin, Z-L. Xia, N-Y. Xu, R. Gao, X-F. Cai, S. Maeng and F-H. Hsu, "FTL Design Exploration in Reconfigurable High-Performance SSD for Server Applications," *Int'l Conf. on Supercomputing (ICS'09)*, 2009.
[6] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee and H.-J. Song, "System Software for Flash Memory: A Survey," in *proc. of Int'l Conf. of Embedded and Ubiquitous Computing (EUC'06)*, 2006.
[7] J. Kim, J. M. Kim, S. H. Noh, S. L. Min and Y. Cho, "A space efficient flash translation layer for compact flash systems," *IEEE Transactions on Consumer Electronics*, 2002.
[8] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Transactions on Embedded Computing Systems*, 2007.
[9] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho and J.-S. Kim, "A reconfigurable FTL (Flash Translation Layer) architecture for NAND flash based applications," *ACM Transactions on Embedded Computing Systems*, 2008.
[10] H.-J. Cho, D. Shin and Y. I. Eom, "KAST: K-Associative Sector Translation for NAND Flash Memory in Real-Time Systems," in *Proc. of Design, Automation and Test in Europe (DATE'09)*, 2009.
[11] Bonnie++ filesystem benchmark, http://www.coker.com.au/bonnie++.
[12] IOzone filesystem benchmark, http://www.iozone.org/.

하 병 민
2009년 성균관대학교 전자전기컴퓨터공학과(학사). 2009년~현재 성균관대학교 전자전기컴퓨터공학과 석사과정. 관심분야는 운영체제, 낸드 플레시 메모리 등

조 현 진
2004년 배재대학교 컴퓨터공학과(학사) 2004년~현재 성균관대학교 전자전기컴퓨터공학과 석박사 통합과정. 관심분야는 운영체제, 낸드 플래시 메모리 등

엄 영 익
1983년 서울대학교 계산통계학과(학사) 1985년 서울대학교 전산과학과(이학석사) 1991년 서울대학교 전산과학과(이학박사) 1993년~현재 성균관대학교 정보통신공학부 교수. 2007년~현재 성균관대학교 정보통신처 처장. 관심분야는 분산 컴퓨팅, 시스템 소프트웨어, 운영체제, 미들웨어, 시스템 보안 등