향상된 혼합 사상기법을 이용한 효율적인 대블록 플래시 메모리 변환계층 설계 및 구현

(Design and Implementation of an Efficient FTL for Large Block Flash Memory using Improved Hybrid Mapping)

박 동 주 † 곽 경 훈 ††
(Dong-Joo Park) (Kyounghoon Kwak)

요 약 플래시 메모리는 크기가 작고, 적은 전력을 사용하며 충격에 강하기 때문에 MP3 플레이어, 핸드폰, 디지털 카메라와 같은 휴대용 기기에서 저장장치로 널리 사용되고 있다. 플래시 메모리의 많은 장점 때문에 개인용 컴퓨터 및 노트북에서 사용되는 저장장치인 하드디스크를 플래시 메모리로 대체하고자 하는 연구도 진행되고 있다. 플래시 메모리는 덮어쓰기가 허용되지 않으며 읽기/쓰기의 기본 단위와 삭제의 기본 단위가 다르기 때문에 FTL(Flash Translation Layer)라는 플래시 변환 계층을 사용한다. 최근에는 기존의 플래시 메모리와 다른 물리구조와 특성을 갖는 대블록 플래시 메모리가 등장하여 기존의 FTL을 그대로 사용하게 되면 플래시 메모리를 효율적으로 사용할 수 없다. 본 논문에서는 기존의 FTL 중 가장 좋은 성능을 내는 FAST(Fully Associative Sector Translation)을 기반으로 데이타블록 내에서 페이지단위 사상을 적용하여 대블록 플래시 메모리의 특성에 맞는 FTL 기법을 제안한다.

키워드 : 플래시 메모리, 플래시 변환 계층, 대블록 플래시 메모리

Abstract Flash memory is widely used as a storage medium of mobile devices such as MP3 players, cellular phones and digital cameras due to its tiny size, low power consumption and shock resistant characteristics. Currently, there are many studies to replace HDD with flash memory because of its numerous strong points. To use flash memory as a storage medium, FTL(Flash Translation Layer) is required since flash memory has erase-before-write constraints and sizes of read/write unit and erase unit are different from each other. Recently, new type of flash memory called "large block flash memory" is introduced. The large block flash memory has different physical structure and characteristics from previous flash memory. So existing FTLs are not efficiently operated on large block flash memory. In this paper, we propose an efficient FTL for large block flash memory based on FAST(Fully Associative Sector Translation) scheme and page-level mapping on data blocks.

Key words : flash memory, flash translation layer, large block flash memory

1. 서 론

플래시 메모리는 전원 공급이 중단되어도 저장된 정보가 사라지지 않는 비휘발성 특징이 있다. 또한 플래시 메모리는 하드디스크(HDD)에 비해 크기가 작고 가벼우며, 전력을 적게 사용하고 충격에 강한 특징이 있다. 이러한 장점 때문에 MP3 플레이어, 핸드폰, 디지털 카메라와 같은 휴대용 기기의 저장장치로 널리 사용되고 있다. 최근에는 플래시 메모리의 뛰어난 성능 때문에 개인용 컴퓨터(PC)와 노트북에서 사용되는 저장장치인 하드디스크를 플래시 메모리로 대체하고자 하는 연구도 진행되고 있다.

하드디스크는 읽기와 쓰기연산의 비용이 동일하지만

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음

† 정 회 원 : 숭실대학교 컴퓨터학부 교수
djpark@ssu.ac.kr
†† 정 회 원 : 숭실대학교 컴퓨터학부
kwak_kh@naver.com

논문접수 : 2008년 4월 30일
심사완료 : 2008년 10월 30일

정보과학회논문지 : 컴퓨팅의 실제 및 레터 제15권 제1호(2009.1)

플래시 메모리는 읽기와 쓰기연산의 비용이 서로 다르다. 또한 플래시 메모리는 하드디스크와 달리 덮어쓰기 연산이 허용되지 않기 때문에 이미 사용한 공간에 새로운 데이타를 저장하려면 먼저 삭제연산을 수행한 뒤에 새로운 데이타를 저장해야 한다. 플래시 메모리의 읽기/쓰기 연산의 비대칭적 비용과 큰 비용을 필요로 하는 삭제연산 때문에 플래시 메모리를 하드디스크와 같은 저장장치로 사용하기 위해서 플래시 변환 계층(FTL, Flash Translation layer)라는 시스템 소프트웨어를 사용한다.

FTL은 파일 시스템과 플래시 메모리 사이에 위치하여 파일 시스템에서 사용하는 논리주소와 플래시 메모리에서 사용하는 물리주소를 사상해 주고, 덮어 쓰기연산이 발생하여도 사용자가 삭제연산을 알 수 없게끔 감추어주는 역할을 한다. FTL을 사용하면 기존에 사용하던 파일 시스템이나 소프트웨어를 변경하지 않고도 하드디스크 등의 저장장치를 사용하는 환경에서 플래시 메모리를 저장장치로 사용할 수 있다.

플래시 메모리의 용량은 "황의 법칙"에 따라 매년 2배씩 증가하고 있다. 최근에는 플래시 메모리를 대용량화 하면서 기존의 플래시 메모리와 물리적 구조와 특성이 다른 플래시 메모리가 개발되고 있다. 이에 따라 플래시 메모리에서 보다 효율적으로 동작하면서 새로운 플래시 메모리의 물리구조와 특성에 맞는 FTL 개발이 필요하다.

플래시 메모리를 저장장치로 사용하기 위한 기존 연구 [1]에서는 FAST 기법을 제안하였지만, 이는 기존의 소블록 플래시 메모리에 적합한 알고리듬을 사용하고 있어 물리적 구조와 특성이 다른 대블록 플래시 메모리에는 적합하지 않다. 대블록 플래시 메모리의 특성을 고려한 FTL을 개발하기 위한 연구로는 [2]가 있으며 본 논문에서는 대블록 플래시 메모리에서 효율적으로 동작하는 FTL인 Page Level Advanced Mapping(이하 PLAM)기법을 제안하고 그 우수성을 규명한다.

본 논문의 구성은 다음과 같다. 2장에서는 플래시 메모리의 물리 구조와 특성에 대해서 기술하고, 3장에서는 기존에 연구된 FTL의 기법에 대해서 논한다. 4장에서는 대블록 플래시 메모리에서 효율적으로 동작하는 FTL을 설계 및 구현하기 위해 제안한 PLAM기법을 제안한다. 5장에서는 PLAM기법의 성능을 기존의 FTL기법의 성능과 비교하여 PLAM기법의 우수성을 보인다. 마지막으로 6장에서는 결론 및 향후 계획에 대해 기술한다.

2. 플래시 메모리

2.1 플래시 메모리의 구조

플래시 메모리를 대용량화 하면서 기존의 플래시 메모리와 다른 특성을 갖는 플래시 메모리가 개발되었다. 새로운 플래시 메모리는 기존의 플래시 메모리에 비해 더 큰 단위의 페이지와 블록을 갖기 때문에 "대블록 플래시 메모리"라고 한다. 이에 대비하여 기존의 플래시 메모리를 "소블록 플래시 메모리"라고 한다.

2.2 소블록 플래시 메모리

플래시 메모리는 읽기/쓰기 연산의 기본 단위인 "페이지(Page)"와, 삭제의 기본 단위인 "블록(Block)"으로 구성된다. 그림 1은 소블록 플래시 메모리의 물리구조를 보여준다[3]. 페이지는 512바이트의 섹터(Sector)와 16바이트의 여유공간(Spare Area)로 구성된다. 섹터는 사용자 데이타를 저장하며, 여유공간은 섹터의 논리주소나 오류 정정코드(ECC, Error Correcting Code) 등을 기록하는데 사용한다[4]. 블록은 32개의 페이지로 구성된다.

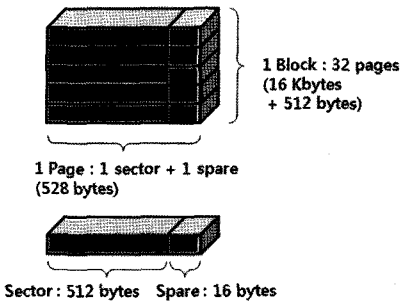


그림 1 소블록 플래시 메모리의 물리 구조

한 번 사용한 페이지에 대해서는 쓰기연산이 허용되지 않으며 이전에 기록한 데이타를 삭제한 뒤에 쓰기연산을 수행할 수 있다. 플래시 메모리는 블록이 삭제연산의 기본 단위이기 때문에 페이지에 기록한 데이타를 삭제하려면 그 페이지가 속한 블록을 삭제해야 한다. 따라서 한 개의 페이지를 삭제하기 위해서는 32개의 페이지를 삭제해야 한다. 덮어쓰기 연산이 발생할 때마다 삭제연산이 발생하는 단점을 보완하기 위해서 Flash Translation Layer(FTL)이라고 하는 주소변환 계층을 사용한다[5].

2.3 대블록 플래시 메모리

플래시 메모리를 대용량화하기 위해 기존의 플래시 메모리와 다른 물리적 구조와 특성을 갖는 대블록 플래시 메모리가 개발되었다[6]. 대블록 플래시 메모리도 "페이지"와 "블록" 두 개의 기본 단위를 사용한다. 그림 2는 대블록 플래시 메모리의 물리구조를 보여준다. 대블록 플래시 메모리의 페이지는 4개의 섹터와 4개의 여유공간으로 구성된다. 대블록 플래시는 페이지단위뿐만 아니라, 섹터단위의 읽기/쓰기 연산도 지원한다. 따라서

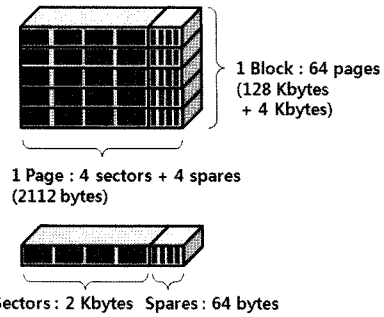
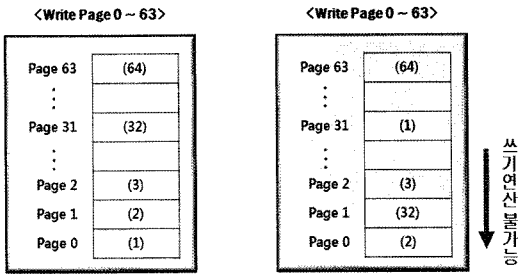


그림 2 대블록 플래시 메모리의 물리 구조



(a) 순차 페이지 쓰기 (b) 임의 순서 페이지 쓰기

그림 3 순차쓰기 제약 사항

페이지를 섹터단위로 사용하는 경우, 한 페이지에 최대 4번의 쓰기연산 수행이 가능하다. 대블록 플래시 메모리는 한 개의 블록이 64개의 페이지로 구성된다. 따라서 한 번의 삭제연산을 하려면 64개의 페이지를 삭제해야 한다. 대블록 플래시 메모리는 한 블록에 속하는 페이지의 개수가 많기 때문에 기존의 플래시 메모리보다 더 효율적인 FTL이 필요하다.

2.4 플래시 메모리의 특성

최근 휴대용 기기의 저장장치로 각광을 받고 있는 플래시 메모리는 PC의 저장장치로 널리 사용하고 있는 하드디스크와 다른 물리적 특성을 갖는다. 하드디스크는 모터를 사용하여 플래터를 고속으로 회전시켜 데이타를 읽는 방식을 사용하는 기계장치이다. 따라서 전력 소모가 많고 무거우며, 발열이 많고 충격을 받으면 손상을 입을 수 있다. 반면, 플래시 메모리는 전자장치이기 때문에 하드디스크에 비해 발열이 적고, 전력 소모가 적으며, 무게가 적게 나간다. 또한 플래시 메모리는 하드디스크보다 더 큰 충격에도 견딜 수 있기 때문에 휴대용 기기의 저장장치로 사용하기에 적합하다.

플래시 메모리는 데이타를 기록하기 위해 블록 소거와 같은 연산이 추가로 필요하지만, 표 1에서와 같이 임의접근을 위한 회전지연시간과 탐색시간이 필요한 하드디스크에 비해 대체로 빠르게 동작한다. 최근에는 이러한 플래시 메모리의 장점을 활용하기 위해 개인용 컴퓨터나 노트북에서 사용하는 HDD를 플래시 메모리로 대체하려는 연구가 진행되고 있다.

플래시 메모리는 지정된 횟수(일반적으로 10만 번)만큼 블록을 삭제할 수 있다. 블록을 지정된 횟수만큼 삭제한 후에는 저장한 데이타의 안정성이 보장되지 않는다. 표 1에 의하면 삭제연산은 쓰기연산에 비해 비용이 크기 때문에, FTL은 삭제연산이 필요한 덮어쓰기 연산을 빈 공간에 대한 쓰기연산으로 변환하기 위한 알고리듬을 사용한다. 알고리듬에 따라 같은 작업에 대한 읽기/쓰기/삭제 연산 횟수도 다르게 나타나기 때문에, 효율적인 알고리듬을 사용한 FTL을 개발하면 플래시 메모리의 더 좋은 성능과 긴 수명을 보장할 수 있다.

대블록 플래시 메모리는 블록내의 페이지를 순차적으로 사용해야 하는 제약사항이 추가되었다. 대블록 플래시 메모리의 블록은 0번부터 63번 페이지로 구성되는데, 그림 3(a)와 같이 페이지를 0번부터 순차적으로 쓰는 경우 블록내의 모든 페이지를 사용할 수 있다. 그러나 그림 3(b)와 같이 31번 페이지를 먼저 사용하면 32~63번 페이지만 사용 가능하며, 0~30번 페이지는 사용할 수 없다. 순차쓰기 제약사항이 고려되지 않은 기존의 FTL은 대블록 플래시 메모리에서 효율적으로 동작하지 않는다. 순차쓰기 제약사항은 대블록 플래시 메모리를 위한 FTL을 설계할 때 반드시 고려해야 할 중요한 요소이다.

3. 관련 연구

3.1 FTL

FTL은 플래시 메모리를 하드디스크와 같은 저장장치

표 1 접근속도 비교 (HDD vs Small Block Flash vs Large Block Flash)[7]

저장장치	임의 접근 소요 시간		
	읽기	쓰기	삭제
HDD	12.7 ms (2 KB)	13.7 ms (2 KB)	N/A
Small Block NAND Flash	15 us (512 B)	200 us (512 B)	2 ms (16 KB)
Large Block NAND Flash	25 us (2 KB)	200 us (2K B)	1.5 ms (128 KB)

HDD : 시게이트 바라쿠다 7200.7 ST380011A
Small Block NAND Flash : Samsung 512MB SLC NAND Flash K9F1208R0C
Large Block NAND Flash : Samsung 2GB SLC NAND Flash K9KAG08U1M

로 사용하기 위해 필요한 시스템 소프트웨어이다. 플래시 메모리의 사용이 급증하면서 다양한 FTL의 개발이 이루어졌다.

초기에 개발된 FTL은 그림 4와 같은 섹터단위 사상을 사용했다[8]. 섹터단위 사상은 파일시스템에서 사용하는 주소인 논리섹터번호(LSN, Logical Sector Number)를 플래시 메모리의 물리섹터번호(PSN, Physical Sector Number)에 사상한다. 섹터단위 사상은 입력한 데이타를 플래시 메모리 내의 모든 빈 섹터 중 하나를 선택하여 사상할 수 있기 때문에 유연성이 높지만 모든 섹터의 사상정보를 저장하기 위해 큰 공간을 할당해야 한다.

섹터단위 사상의 단점을 보완하기 위해서 그림 5와 같은 블록단위 사상 방식이 고안됐다[9-11]. 블록단위 사상은 블록마다 32개의 섹터가 존재한다는 것을 이용하여 논리주소를 논리블록번호(LBN, Logical Block Number)와 논리섹터 오프셋(LSO, Logical Sector Offset)으로 분리하여 논리블록번호를 물리블록번호로 사상한다. 블록단위 사상을 하기 위해서 LSN을 한 블록 당 페이지 개수로 나누어서 LBN을 구하고 그 나머지를 LSO로 한다. 정확한 섹터의 위치는 사상정보를 통해 얻은 블록의 LSO번째 섹터다. 블록단위 사상은 각 블록에 대한 사상정보만 저장하기 때문에 사상정보를 저장하기 위한 공간이 섹터단위 사상에 비해 매우 작다. 하지만 블록단위 사상은 블록내의 LSO 위치에만 데이타를 저장하기 때문에 같은 위치에 쓰기연산이 반복되는 경우 블록 내의 페이지를 모두 활용하지 못하고 빈 블록으로 합병연산을 수행한다. 합병연산은 불필요한 읽기/쓰기/삭제 연산을 발생시키므로 전체적인 성능을 떨어트린다.

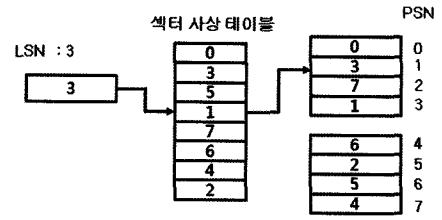


그림 4 섹터단위 사상기법

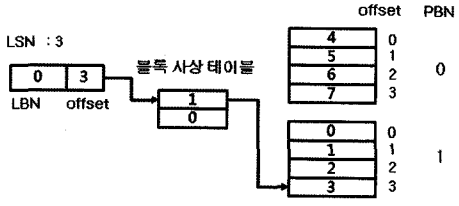


그림 5 블록단위 사상기법

3.2 FAST

기존의 FTL은 섹터단위 사상이나 블록단위 사상만 사용하였는데, [12]에서는 섹터단위 사상과 블록단위 사상을 혼합한 방식(Hybrid Mapping Scheme)인 블록단위 연관기법(BAST, Block Associative Sector Translation)을 제안했다. BAST는 플래시 메모리의 블록을 데이타를 저장하기 위한 데이타블록과 덮어쓰기 연산이 발생할 때 사용하는 로그블록으로 나누어 사용한다. BAST는 데이타블록에 대해서는 블록단위 사상을 하고, 로그블록에 대해서는 섹터단위 사상을 한다. BAST는 덮어쓰기 연산이 발생하면, 빈 블록을 로그블록을 할당하여 쓰기연산을 수행한다. 이때 한 개의 데이타블록이 한 개의 로그블록을 할당받아 사용한다. 로그블록의 개수는 한정되어 있기 때문에 로그블록이 모두 사용 중일 때 새 로그블록이 필요한 경우나, 데이타블록에 할당한 로그블록을 모두 사용한 상태에서 덮어쓰기 연산이 발생한 경우 합병연산이 발생한다. 합병연산은 로그블록 내에 있는 최신의 데이타와 데이타블록 내에 있는 덮어쓰기가 발생하지 않은 데이타들을 새 블록에 기록하는 연산이다. BAST는 한 개의 데이타블록이 한 개의 로그블록을 사용하기 때문에 여러 데이타블록에 대해 덮어쓰기 연산이 발생하면 로그블록 활용률이 낮아지고 잦은 합병연산이 발생한다.

BAST의 로그블록 활용률이 떨어지는 단점을 보완하기 위해 모든 데이타블록이 로그블록을 공유하는 완전연관기법(FAST, Fully Associative Sector Translation)이 제안되었다[1]. FAST는 그림 6과 같이 기존의 로그블록을 순차쓰기 로그블록과 임의쓰기 로그블록으로 나누어 사용한다. 블록내의 0번 페이지부터 순차적으로 덮어 쓰기연산이 발생하는 경우 순차쓰기 로그블록에 쓰기연산을 수행한다. 순차쓰기 로그블록을 모두 사용하면 사상 정보만 변경하는 교환연산을 수행한다. 블록내의 0번 페이지가 아닌 페이지부터 덮어쓰기 연산이 발생하는 경우 임의쓰기 로그블록에 저장한다. FAST는 모든 데이타블록이 로그블록을 공유하기 때문에 BAST에 비해 로그블록의 활용률이 높다. 따라서 같은 작업부하에 대해 BAST보다 합병연산이 비교적 적게 발생하고 불필요한 읽기/쓰기/삭제연산이 줄어든다.

FAST는 블록의 첫 번째 페이지에 반복적으로 덮어쓰기연산이 발생하면, 순차쓰기 로그블록의 활용률이 낮아지고 합병연산이 증가한다. 또한 임의쓰기 로그블록은 합병연산을 수행할 때, 로그블록과 연관된 데이타블록에 대해 모두 합병연산을 수행해야 하기 때문에 한 번 합병연산을 수행하는데 필요한 비용이 커지는 단점이 있다. FAST는 데이타블록 내의 섹터를 순차적으로 사용하지 않기 때문에 대블록 플래시 메모리에 그대로 적용

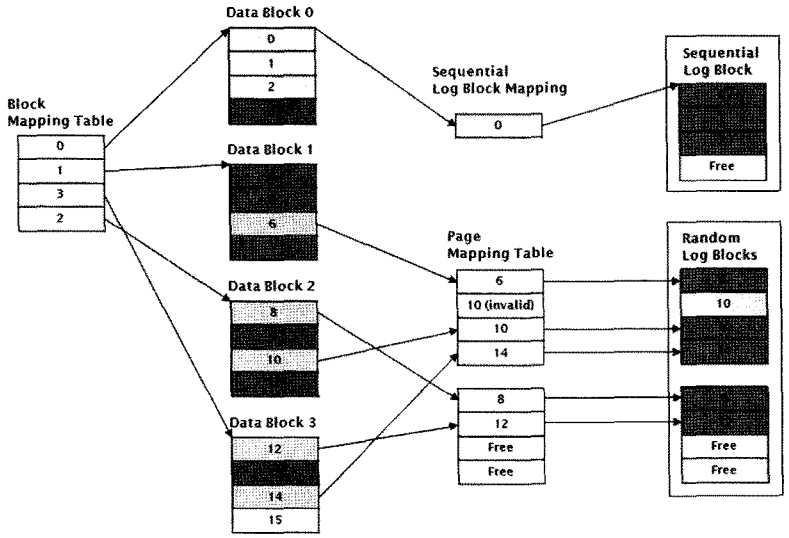


그림 6 FAST의 구조

하여 사용할 수 없다.

3.3 LSTAFF*

[13]에서 블록에 저장한 데이타의 형태에 따라 블록의 상태를 변경하는 STAFF(State Transition Applied Fast Flash Translation Layer)를 제안했다. LSTAFF (Large STAFF)는 소블록 플래시 메모리를 대상으로 설계된 STAFF를 대블록 플래시 메모리의 특성에 맞도록 확장한 FTL이다[14]. LSTAFF에서 사용하는 블록의 상태는 표 2와 같다.

LSTAFF는 블록 내에 저장한 페이지의 상태에 따라 그림 7과 같이 블록의 상태를 전이한다. 블록을 사용하기 전이나 블록이 삭제된 상태이면 "F"상태가 된다. "F"상태의 블록은 새 블록에 쓰기연산이 필요한 경우에 할당하여 사용한다. 쓰기연산이 발생하면, 블록사상 테이블을 참조하여 블록이 "M"상태이면 논리섹터 오프셋의 위치에 기록가능한지 확인하여 기록한다. 만약 해당 위치에 데이타가 기록되어 있으면, 블록의 상태를 "N"으로 변환하고 빈 페이지에 쓰기연산을 수행한다. "N" 상태인 경우에는 빈 페이지를 찾아 기록한다. "S"상태인 경우에는 더 이상 빈 페이지가 없으므로 새 블록을 할당해 쓰기연산을 수행한다.

표 2 LSTAFF의 블록상태

상태	설명
F	블록이 삭제되었거나 아직 사용되지 않은 상태
O	블록이 더 이상 유효하지 않은 상태
M	논리섹터 오프셋과 물리섹터 오프셋이 같은 상태
S	M 상태에서 모든 페이지를 사용한 상태
N	논리섹터 오프셋과 물리섹터 오프셋이 다른 상태

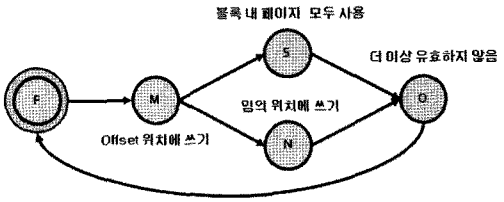


그림 7 LSTAFF 블록 상태 변환

LSTAFF에서는 "S"상태에서 쓰기연산이 발생하면, 논리블록에 새로 할당된 물리블록과 이전의 물리블록이 함께 사상된다. 이 방법은 합병연산의 횟수를 줄여주지만 논리블록 한 개에 2개의 물리블록을 할당하기 때문에 실제 사용가능한 논리블록 개수는 물리블록 개수의 절반으로 줄어든다. 또한 LSTAFF는 블록단위 사상기법과 달리 논리섹터 오프셋과 물리섹터 오프셋이 일치하는 위치에 쓰기연산을 수행해야하는 제약이 없다. 하지만 페이지단위의 사상정보를 별도로 저장하지 않기 때문에 "N" 상태에서 읽기연산을 수행할 때, 원하는 데이타를 찾기 위해서 많은 페이지를 읽어야 한다.

[2]에서는 LSTAFF를 보다 효율적으로 대블록 플래시 메모리에서 동작시키기 위해서 이를 개선한 LSTAFF*를 제안하였다. LSTAFF*는 "M", "N"모드를 세분화 하여 임의순서 쓰기연산에 대해서 높은 유연성을 가지도록 하였다. 또한 순차쓰기 제약사항을 극복하기 위해서 오프셋 값에 상관없이 블록 내에서 빈 공간을 검색하여 가장 앞쪽부터 페이지를 사용하도록 하였다. 하지만 LSTAFF*역시 LSTAFF와 마찬가지로 페이지 단위의 사상정보를 별도로 저장하지 않기 때문에 블록

이 "N"상태일 때 원하는 데이타를 찾기 위해서 많은 읽기 연산이 필요하다. 또한 LSTAFF*는 1개의 논리블록당 2개의 물리블록을 할당하여 사용하므로 실제 사용가능한 논리블록의 개수는 물리블록 개수의 절반에 불과하다.

4. PLAM의 설계 및 구현

대블록 플래시 메모리를 위한 FTL의 개발을 위해 FAST 알고리듬을 기반으로 PLAM을 설계하였다. 그림 8은 본 논문에서 제안하는 PLAM의 전체 구조를 보여준다. 대블록 플래시 메모리의 순차쓰기 제약사항을 만족시키기 위해 데이타블록에 대해서도 페이지단위 사상을 사용하였다. FAST의 블록단위 사상 테이블을 변경하여 블록상태 정보를 저장하였고, 페이지 내의 여유영역에 페이지단위 사상정보를 저장하는 방법을 사용하였다. 여유영역에 저장한 페이지단위 사상정보는 페이지단위 사상 테이블로 변환하여 사용한다. 또한 FAST의 알고리듬이 대블록 플래시 메모리에서 보다 효율적으로 동작하도록 페이지단위의 읽기버퍼와 쓰기버퍼를 추가하였고, 로그블록을 효율적으로 사용하기 위해서 순차쓰기 로그블록 관리자와 합병대상 블록선정 알고리듬을 추가하였다.

4.1 페이지단위 사상 기법

본 논문에서 제안하는 PLAM기법은 데이타블록에 대해 블록단위 사상을 사용한다. 하지만 대블록 플래시 메모리는 기존의 소블록 플래시 메모리와 블록내의 페이지 개수나 페이지 내의 섹터 개수가 다르기 때문에 소블록 플래시 메모리를 위한 FTL에서 사용하였던 사상정보를 구하는 공식을 변경해야 한다. 대블록 플래시 메모리에 맞게 사상정보를 구하기 위해서 다음과 같이 논리페이지번호(LPN, Logical Page Number), 논리섹터오프셋(LSO, Logical Sector Offset), 논리블록번호(LBN), 논리페이지 오프셋(LPO, Logical Page Offset)을 구한다.

$$LPN = LSN / NSP$$
$$LSO = LSN \ \% \ NSP$$
$$LBN = LPN / NPB$$
$$LPO = LPN \ \% \ NPB$$

소블록 플래시 메모리는 LSN을 블록 당 페이지 개수(NPB, Number of Pages per Block)으로 나누어 LSO를 구했지만, 대블록 플래시 메모리는 페이지를 구성하는 섹터의 개수가 4이므로 먼저 LSN을 페이지 당 섹터개수(NSP, Number of Sectors per Page)로 나누어서 LPN을 구한 뒤 LPN을 NPB로 나누어서 LBN과 LPO를 구한다. PLAM기법에서 LPO는 블록내의 페이지 오프셋을 뜻한다. LBN과 LPO는 각각 블록단위 사상기법의 LBN과 LSO와 같은 방식으로 사용한다. LSO는 페이지 내의 4개의 섹터 중 쓰고자 하는 데이타가 위치할 섹터의 번호를 나타낸다.

순차쓰기 제약사항을 극복하기 위해 PLAM기법은 블록단위 사상을 사용하되, 블록 내에서는 페이지단위 사상을 사용한다. 페이지단위 사상을 사용하면 기존의 섹터단위 사상을 사용할 때와 마찬가지로 사상정보를 저

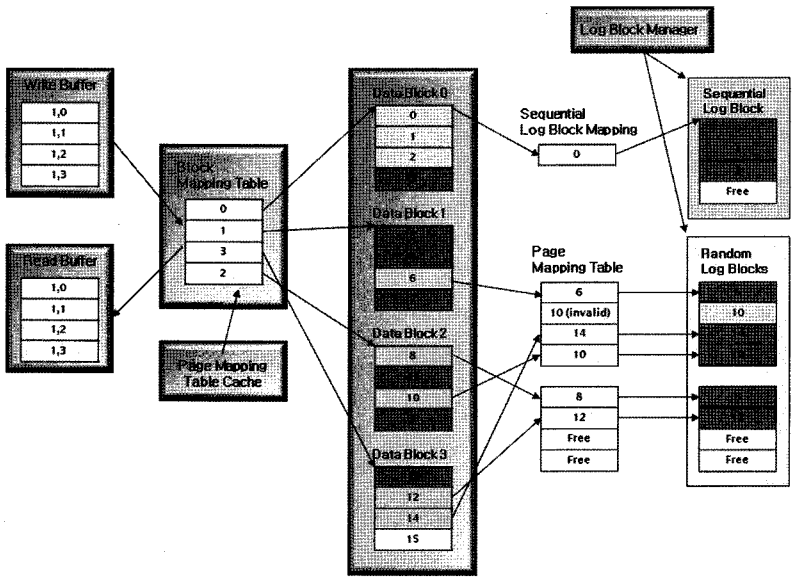


그림 8 PLAM의 구조

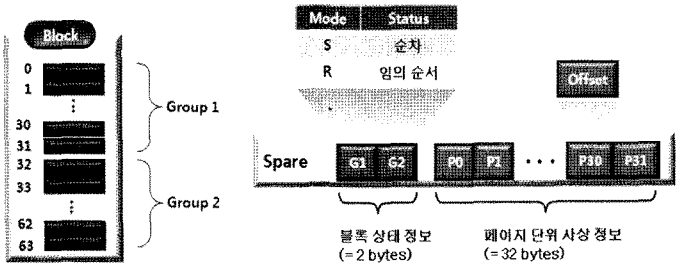


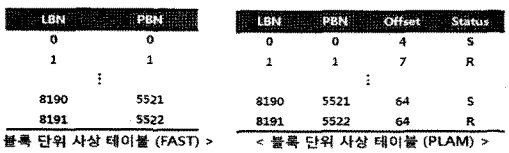
그림 9 페이지단위 사상 정보

장하기 위해 많은 공간이 필요하다. 페이지단위 사상정보를 별도의 공간을 할당하지 않고 저장하기 위해서 그림 9와 같이 페이지 내의 여유공간을 활용한다.

여유공간에는 LSN이나 오류 정정코드(ECC)가 기록되기 때문에 여유공간의 64바이트를 모두 페이지단위 사상정보를 기록하는데 사용할 수 없다. 따라서 한 개의 블록을 2개의 그룹으로 나누어 페이지단위 사상정보를 기록한다. 블록의 페이지를 0번부터 순차적으로 사용하는 경우에는 블록의 상태를 "S"(순차, Sequential)상태로 한다. 만약 비순차적인 쓰기연산이 발생하는 경우 블록의 상태를 "R"(임의, Random)상태로 변경한다. 또한, 페이지단위 사상정보에 저장한 LSN에 해당하는 LPO를 기록한다.

블록내의 페이지를 순차적으로 사용하기 위해 그림 10과 같이 마지막으로 사용한 블록내의 페이지 오프셋보다 하나 큰 값을 블록단위 사상 테이블에 저장한다. 쓰기연산이 발생하면 블록단위 사상 테이블을 확인하여 테이블에 저장된 오프셋 위치에 쓰기연산을 수행한다. 블록단위 사상 테이블에 기록된 오프셋이 64인 블록에 대해서 쓰기연산이 발생하면, 로그블록에 쓰기연산을 수행한다. 이 방법을 사용하면 블록내의 페이지를 항상 순차적으로 사용할 수 있고, 블록내의 모든 페이지를 사용할 수 있다.

블록내의 페이지 저장상태가 순차적인 경우 LPO을 통해 데이타가 저장된 페이지에 접근할 수 있으므로 별도의 페이지단위 사상정보가 필요하지 않다. 그림 10과 같이 블록단위 사상테이블에 블록의 상태를 저장하여 블록 내의 페이지가 순차적인 경우 LPO를 통해 바로 접근하도록 하였다. 만약 블록이 "R"상태이면 블록단위



LBN	PBN
0	0
1	1
⋮	
8190	5521
8191	5522

< 블록 단위 사상 테이블 (FAST) >

LBN	PBN	Offset	Status
0	0	4	S
1	1	7	R
⋮			
8190	5521	64	S
8191	5522	64	R

< 블록 단위 사상 테이블 (PLAM) >

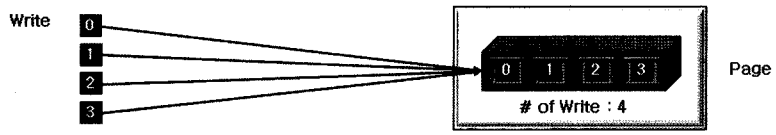
그림 10 블록단위 사상 테이블

사상테이블에 저장된 오프셋보다 하나 적은 페이지를 읽어 가장 최근에 저장한 페이지단위 사상정보를 구하여 원하는 데이타가 위치한 오프셋을 알아낸 뒤 다시 읽기연산을 수행한다. 그룹2의 페이지단위 사상정보를 읽어서 LPO를 검색하여 원하는 LPO를 찾지 못하면 그룹1의 페이지단위 사상정보를 읽어서 LPO를 구한다. 이때 그룹2의 페이지단위 사상정보에서 그룹1의 페이지단위 사상정보를 확인하여 그룹1의 상태가 "S"이면 그룹1의 사상정보를 읽지 않고, LPO에 해당하는 페이지에 대해 읽기연산을 수행할 수 있다. 앞서 읽은 블록의 그룹1에 해당하는 페이지단위 사상정보와 그룹2에 해당하는 페이지단위 사상정보는 각각 1개씩 버퍼에 저장되며 같은 블록에 대한 연산을 수행할 때 반복해서 사상정보를 얻어오는 것을 방지한다.

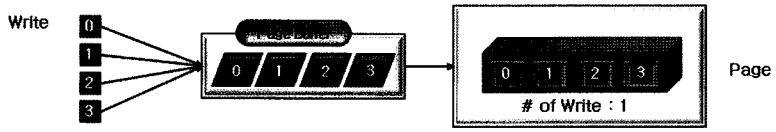
4.2 페이지단위 버퍼 기법

FAST는 쓰기연산이 발생하면 바로 쓰기연산을 수행한다. FAST를 대블록 플래시 메모리에 적용하면 그림 11(a)와 같이 한 개의 페이지에 대해 4번의 쓰기연산을 한다.

본 논문에서 제안하는 PLAM기법은 그림 11(b)와 같이 순차쓰기 연산을 4번 처리하는데 한 번의 쓰기연산을 수행하도록 하기 위해서 쓰기버퍼를 사용하였다. 쓰기버퍼를 사용하면 섹터 데이타 쓰기연산이 발생할 때 같은 페이지에 속하는 데이타를 버퍼에 저장하였다가 다른 페이지에 속하는 데이타를 쓰려고 할 때 페이지에 쓰기연산을 수행한다. 쓰기연산을 수행하기 전에 쓰기버퍼에 빈 섹터가 존재하면, 합병연산이 발생할 때 페이지의 빈 섹터를 채우기 위해 반복적으로 이전에 기록된 페이지를 탐색하는 것을 방지하기 위해서 바로 이전에 기록한 페이지를 읽어서 빈 섹터를 채우는 작업을 수행한다. 이 방법을 사용하면 순차쓰기 연산에 대해 기존의 방법보다 최대 4배 빠르게 수행할 수 있으며, 합병연산도 간단하게 처리할 수 있다. 쓰기버퍼를 사용하면 쓰기연산 수행 중에 전원차단 등의 문제가 발생하여 쓰기버퍼의 내용이 플래시 메모리에 저장되지 못하고 데이타가 유실되는 단점이 있다. 하드디스크가 기존에 데이타

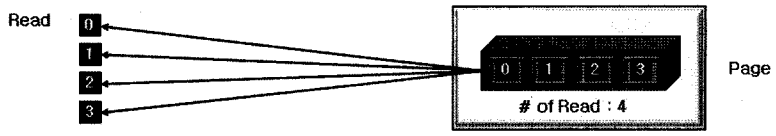


(a) 쓰기버퍼 미사용

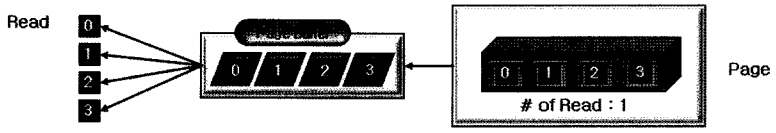


(b) 쓰기버퍼의 활용

그림 11 쓰기버퍼의 활용



(a) 읽기버퍼 미사용



(b) 읽기버퍼의 활용

그림 12 읽기버퍼의 활용

가 기록된 부분에 쓰기연산을 수행하는 in-place update를 사용하는 반면, 플래시 메모리는 기존에 데이타가 기록된 부분이 아닌 다른 곳에 쓰기연산을 수행하는 out-of-place update를 사용하기 때문에 쓰기연산을 수행하는 도중 오류가 발생하여도 기존에 저장한 데이타를 사용하도록 복구하는 것이 가능하다. 데이타를 복구하기 위해서 여유영역에 저장된 LSN과 ECC 정보를 사용한다.

대블록 플래시 메모리에서 페이지단위로 읽기연산을 수행하면 한 번 읽기연산으로 4개의 섹터를 읽을 수 있다. 그림 12(b)와 같이 읽기연산으로 읽은 4개의 섹터를 읽기버퍼에 저장하면 다시 같은 LSN의 데이타를 읽는 경우나, 같은 LBN과 LPO에 속하는 데이타를 읽는 경우 바로 읽기버퍼에서 섹터데이타를 읽을 수 있다. 만약 읽기연산으로 가져온 데이타를 저장하지 않으면 그림 12(a)와 같이 여러 번의 읽기연산이 필요하다. 읽기버퍼를 최신의 상태로 유지하기 위해서 쓰기버퍼의 데이타를 플래시 메모리에 저장한 후 읽기버퍼의 LPN과 쓰기연산이 발생한 LPN이 같으면 읽기버퍼의 내용을 쓰기버퍼의 내용으로 갱신한다.

4.3 개선된 로그블록 관리 기법

덮어쓰기 연산을 처리하기 위해 FAST는 순차쓰기 로그블록과 임의쓰기 로그블록을 사용한다. 순차쓰기 형식이 깨지거나 다른 논리블록에서 순차쓰기가 발생한 경우 순차쓰기 로그블록에 대해 합병연산을 수행한다. 순차쓰기 로그블록의 활용률이 낮을 때 합병연산이 발생하면 데이타블록에서 유효한 페이지 데이타를 가져오기 위해 추가적인 연산이 발생한다.

FAST는 0번 LSO에 쓰기연산이 발생하면 순차쓰기 연산으로 인식하는 반면, PLAM은 0번 LPO에 쓰기연산이 발생하면 순차쓰기 연산으로 인식한다. 대블록 플래시 메모리는 한 개의 블록에 소블록 플래시 메모리보다 많은 64개의 페이지가 있기 때문에 순차쓰기 로그블록의 활용률이 낮은 경우에 불필요한 읽기/쓰기 연산이 많이 발생하게 된다. 이를 방지하기 위해 쓰기버퍼에 0번 LPO에 해당하는 데이타가 있는 경우 다음 번 쓰기연산이 발생한 LBN, LPO와 쓰기버퍼에 저장된 데이타의 LBN, LPO를 비교하여 순차쓰기 연산이 아닌 경우 임의쓰기 로그블록에 쓰기연산을 수행한다. 이 방법을 사용하면 반복적으로 0번 LPO에 임의쓰기 연산이 발생하는 경우에 순차쓰기 로그블록을 한 페이지만 사용하고 반복적으로 합병연산이 발생하는 현상을 방지할 수 있다.

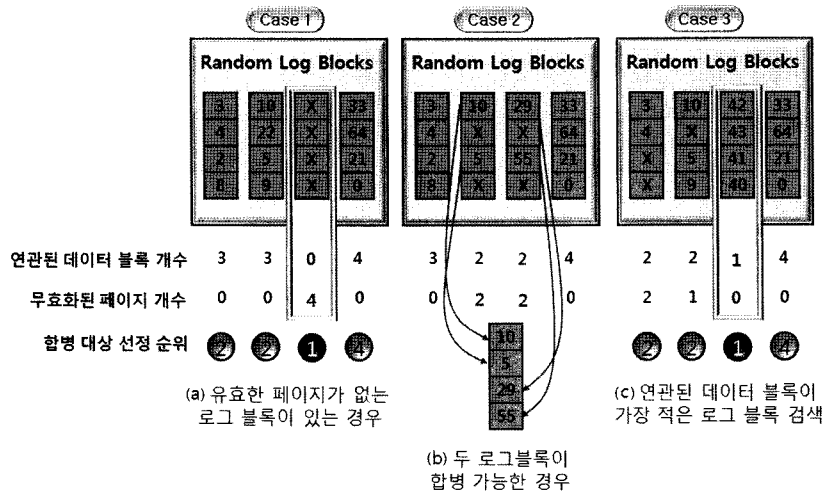


그림 13 합병 대상 로그블록 선정

FAST는 임의쓰기 로그블록에 대해서 합병 대상 로그블록을 선정하기 위해 Round Robin 알고리듬을 사용하는데, 이 알고리듬은 주어진 조건을 고려하여 합병 비용을 줄이도록 시도하지 않는다. 합병에 필요한 비용을 최소화하기 위해서 합병 대상 로그블록을 선정하는 효율적인 알고리듬이 필요하다. 로그블록 합병연산의 경우 해당 로그블록과 연관된 데이타블록의 개수만큼 합병연산이 발생한다. 로그블록에 있는 페이지와 같은 논리 주소를 갖는 페이지가 다른 덮어쓰기 연산으로 인해 다른 로그블록에 쓰인 경우 로그블록에 있는 페이지는 무효화된다. 합병연산 횟수가 가장 적게 발생하는 합병대상 로그블록을 선정하기 위해서 각 로그블록에서 무효화된 페이지의 개수와 각 로그블록과 연관된 데이타블록의 개수를 수집한다.

그림 13(a)와 같이 로그블록 내의 모든 페이지가 무효화된 로그블록을 선택해 삭제연산을 수행한다. 그림 13(a)와 같은 경우가 없으면, 그림 13(b)와 같이 무효화된 페이지가 가장 많은 두 개의 로그블록을 선택하여 두 로그블록 내의 유효한 페이지의 개수를 더한다. 두 로그블록 내의 유효한 페이지 개수의 합이 64(블록 내의 페이지 개수)보다 같거나 적으면 새 블록을 할당하여 한 개의 로그블록으로 합병한다. 이 방법을 사용하면 한 번의 합병연산으로 한 개의 빈 로그블록을 생성할 수 있으며, 유효한 페이지 개수가 64보다 같거나 적으므로 합병한 블록 내에도 사용하지 않은 페이지가 생성될 수 있다. 마지막으로 앞선 두 경우에 속하지 않는 경우 그림 13(c)와 같이 로그블록에 연관된 데이타블록의 개수가 가장 적은 로그블록을 선택하여 합병연산을 최소화 한다.

5. 성능 평가

이번 장에서는 실험을 통해 본 논문에서 제안한 PLAM기법을 FAST, LSTAFF*와 비교하여 PLAM기법의 우수함을 보인다. 이 중 FAST는 소블록 플래시 메모리에서 실험한 결과를 사용하였다. 본 실험에서는 개선된 로그블록 관리를 사용하지 않은 PLAM과 개선된 로그블록 관리자를 사용한 PLAM+LM (Log Block Management)로 나누어 로그블록 관리자의 효과를 알아본다. 성능 평가를 위해서 [1]에서 사용한 시뮬레이터와 동일한 시뮬레이터를 사용했으며, 소블록 플래시 메모리인 [3]과 대블록 플래시 메모리인 [6]의 데이타를 이용하였다. 사용한 샘플 작업부하는 표 3과 같으며, 작업부하 A~D는 [1,12]에서 사용한 것과 같다. 작업부하의 쓰기연산은 플래시 메모리 섹터에 대한 쓰기연산이다. FAST와 PLAM, PLAM+LM의 임의쓰기 로그블록의 개수를 8, 16, 32, 64개로 설정하여 각각의 성능을 측정하였다.

그림 14는 패턴 A 작업부하를 수행했을 때 발생한 총 쓰기연산 횟수를 나타낸다. 패턴 A에서는 대부분 순차쓰기 연산이 발생했다. PLAM은 쓰기버퍼를 사용하여 4번의 순차쓰기 연산을 한 번의 쓰기연산으로 변환했기 때문에 FAST보다 빠른 성능을 보여준다. 연산이

표 3 샘플 작업부하

패턴	출처	쓰기연산 수
A	리눅스	398,000
B	심비안	404,900
C	디지털 카메라 A	3,144,800
D	디지털 카메라 B	6,957,600

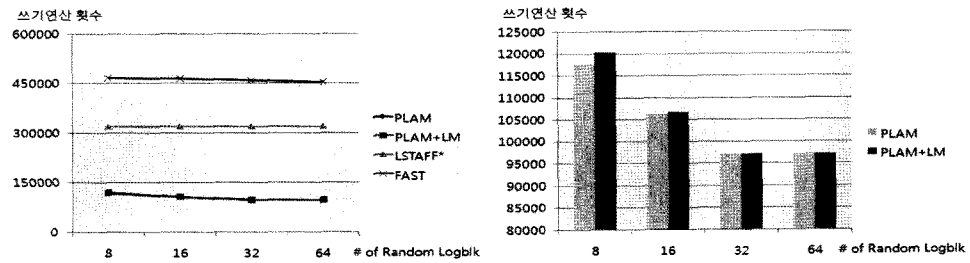


그림 14 패턴 A 쓰기연산 횟수

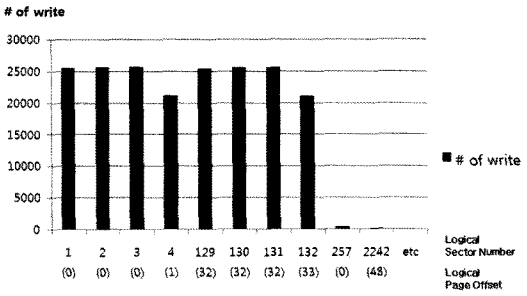


그림 15 패턴 B 분석

대부분 순차쓰기 연산이기 때문에 PLAM과 PLAM+LM의 성능이 거의 동일하다. 로그블록의 개수가 많아질수록 성능이 더 좋아지는 것을 알 수 있다.

그림 16은 패턴 B 작업부하를 수행했을 때 발생한 총 쓰기연산 횟수를 나타낸다. PLAM의 수행시간이 FAST보다 많이 걸리는 것을 볼 수 있다. 패턴 B는 그림 15에서와 같이 LPO 0번에 해당하는 쓰기연산이 많이 발생한다. 이는 소블록 플래시 메모리에서는 LSO 0번에 해당하지 않아 순차쓰기 연산으로 인식되지 않는 연산이었다. 따라서 대블록 플래시 메모리에서는 순차쓰기 로그블록의 0번 페이지만 반복적으로 사용하기 때문에 순차쓰기 로그블록 활용률이 떨어져 전체적인 성능이 감소한다. 로그블록 관리자를 포함한 PLAM+LM에서는 0번 LPO에 대해서 덮어쓰기 연산이 발생하더라도 다음 쓰기연산이 순차적이지 않으면 임의쓰기 로그블록에 쓰기연산을 수행한다. PLAM에서는 임의쓰기 로그블록에 같은 LBN을 갖는 페이지가 존재하면 순차쓰기 로그블록에 쓰기연산을 수행하는 대신 임의쓰기 로그블록에 쓰기연산을 수행한다.

그림 17은 패턴 C 작업부하를 수행했을 때 발생한 총 쓰기연산 횟수를 나타낸다. 패턴 C는 패턴 A와같이 순차쓰기 연산이 많기 때문에 패턴 A를 수행한 결과와 비슷한 결과가 나타난다. 임의쓰기 로그블록의 개수가 늘어나면 다른 덮어쓰기 연산에 의해서 무효화되는 페이지가 많이 생기므로 합병연산 시 추가적으로 발생하는 읽기/쓰기/삭제 연산이 적게 발생한다. 특히, 임의쓰기 로그블록이 64개일 때 PLAM+LM은 PLAM에 비해서 7.4% 성능 향상이 있다.

그림 18은 패턴 D 작업부하를 수행했을 때 발생한 총 쓰기연산 횟수를 나타낸다. 패턴 D는 대부분의 순차쓰기 패턴과 일부의 임의쓰기 패턴으로 이루어져있다. PLAM은 이러한 패턴에서도 좋은 성능을 나타내는 것을 볼 수 있으며, 임의쓰기용 로그블록의 개수가 증가할수록 성능이 좋아지는 것을 볼 수 있다. 임의쓰기 로그블록이 32개일 때 PLAM+LM은 PLAM에 비해서 3.1%의 성능향상이 있다.

그림 19~22는 알고리즘의 효율성을 비교하기 위해서 샘플작업 부하에서 발생한 쓰기연산과 실제 발생한 총 쓰기연산의 배율을 나타냈다. 대블록 플래시 메모리의

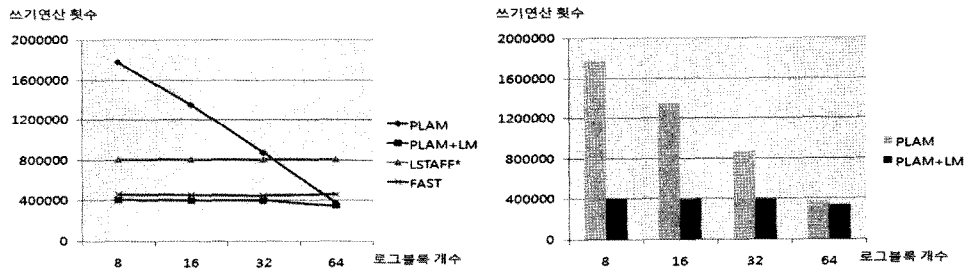


그림 16 패턴 B 쓰기연산 횟수

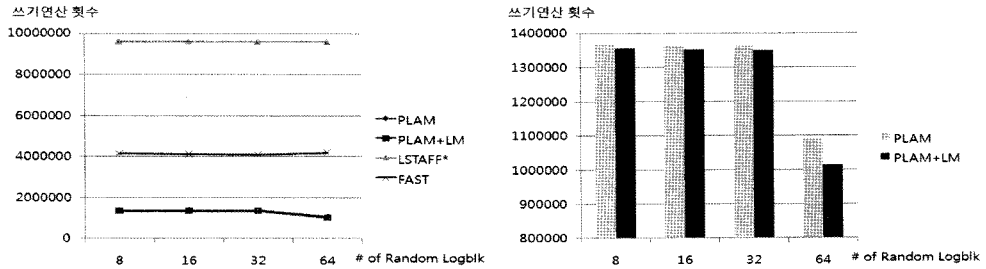


그림 17 패턴 C 쓰기연산 횟수

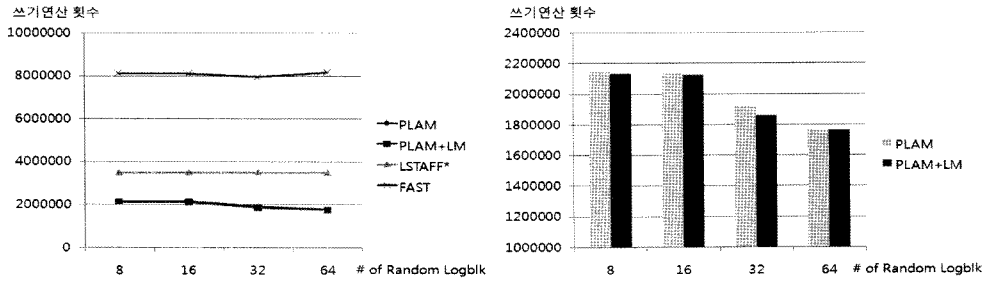


그림 18 패턴 D 쓰기연산 횟수

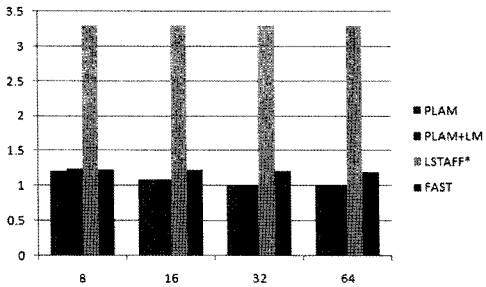


그림 19 패턴 A 쓰기연산에 대한 배율

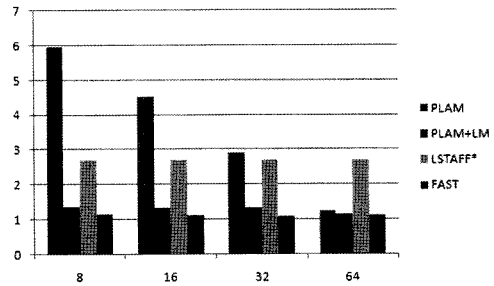


그림 20 패턴 B 쓰기연산에 대한 배율

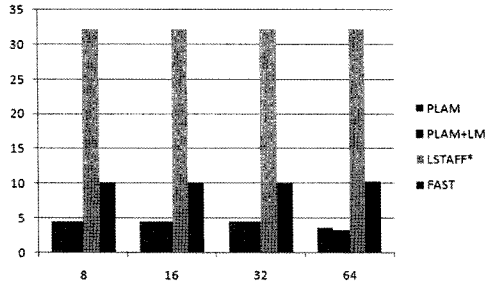


그림 21 패턴 C 쓰기연산에 대한 배율

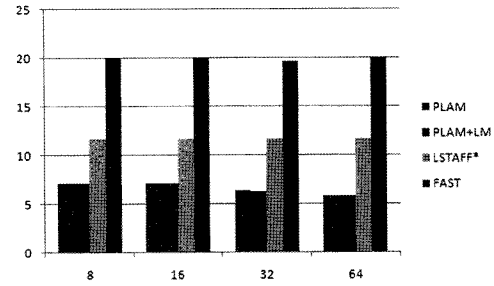


그림 22 패턴 D 쓰기연산에 대한 배율

'TL에서는 버퍼의 효과를 감안하여 샘플작업 부하에서 발생한 쓰기연산에 대한 비율이 아닌, 버퍼를 통해 발생한 쓰기연산 대비 총 쓰기연산으로 배율을 구하였다. 쓰기연산이 비교적 적고 순차쓰기가 많은 패턴 A에서는 PLAM과 FAST의 배율이 비슷하게 나타난다. 패턴 B는 대부분이 임의쓰기 연산으로 이루어져 있다. PLAM이 0번 페이지에 대해 집중적으로 임의쓰기 연산이 발생하였을 때 성능이 크게 저하되는 것과 달리 PLAM+LM은 로그블록을 효율적으로 관리하기 때문에 로그블록이 8개인 경우에도 효율적으로 동작하는 것을 알 수

있다. 패턴 B에서 FAST에 비해서 PLAM+LM의 쓰기 연산 배율이 더 낮지 않은 것은 대블록 플래시 메모리의 블록은 소블록 플래시 메모리의 블록보다 더 많은 페이지가 존재하므로 합병 연산 발생할 때 더 많은 쓰기연산이 발생하였기 때문이다. 쓰기연산이 많은 패턴 C와 패턴 D에서 FAST는 패턴에서 발생한 쓰기연산에 비해 실제 발생한 쓰기 연산이 급격하게 증가했지만, PLAM은 거의 일정한 배율로 쓰기연산이 발생하는 것을 알 수 있다.

실험결과와 같이, PLAM은 쓰기버퍼를 사용하여서 FAST에 비해 쓰기연산 발생횟수를 크게 줄였다. PLAM+LM는 로그블록을 효율적으로 사용하기 위한 로그블록 관리자를 추가하여 0번 LPO에 해당하는 LSN이 반복적으로 들어오는 경우에도 좋은 성능을 발휘한다. PLAM은 소블록 플래시 메모리에서 가장 성능이 좋은 FTL중의 하나인 FAST의 기본 알고리듬을 기반으로 하고 있기 때문에 LSTAFF*보다 좋은 성능을 보인다.

5. 결론 및 향후 계획

플래시 메모리는 작은 크기와 적은 전력 소모, 충격에 강한 특징으로 휴대용 기기에서 저장장치로 널리 사용되고 있다. 유비쿼터스 시대를 맞아 새로운 휴대용 기기들이 개발되고 있으며, 휴대용 기기의 사용자층이 넓어지면서 소비도 크게 증가하고 있다. 특히 개인용 컴퓨터나 노트북에서 사용하는 저장장치인 하드디스크를 대체하여 플래시 메모리를 저장장치로 사용하고자 하는 연구도 널리 진행되고 있어 앞으로 플래시 메모리의 사용량이 더욱 가파르게 증가할 것이다. 플래시 메모리가 대용량화 되고 플래시 메모리를 저장장치로 사용하는 기기가 다양해지면서 플래시 메모리를 효율적으로 사용하기 위한 보다 좋은 성능의 FTL의 개발이 필요하다. 특히 기존의 플래시 메모리와 물리적 구조와 특성이 다른 대블록 플래시 메모리가 개발되면서 이에 맞는 새로운 FTL의 개발도 필요하게 되었다.

본 논문에서 제안한 PLAM은 대블록 플래시 메모리의 물리적 구조와 특성에 맞게 설계하였으며 기존의 FTL에 비해 좋은 성능을 나타내는 것을 보였다. 앞으로는 램에 저장할 페이지 테이블의 수와 읽기/쓰기버퍼의 수를 변화시키면서 이들이 FTL의 성능에 미치는 영향을 확인하고 최적의 값을 찾는 연구를 수행할 계획이다. 또한 로그블록의 활용률을 높이기 위해 기존의 로그블록과 다른 형태로 로그블록을 활용하는 방안도 연구할 계획이다.

참 고 문 헌

[1] 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, Vol.6, No.3, Article 18, 2007.

[2] T.-S. Chung, D.-J. Park, M. Lee, Y. Ryu, J. Kim, "LSTAFF*: An Efficient Flash Translation Layer for Large Block Flash Memory," Submitted for publication, 2007.

[3] Samsung Electronics, "64M × 8 Bit SLC Small Block NAND Flash Memory," http:www.samsung.com/, 2007.

[4] Samsung Electronics, "NAND Flash Spare Area Assignment Standard," http://www.samsung.com/, 2005.

[5] Intel Corporation, "Understanding the Flash Translation Layer(FTL) Specification," http://www.intel.com/, 1998.

[6] Samsung Electronics, "2G × 8 Bit SLC Large Block NAND Flash Memory," http://www.samsung.com/, 2007.

[7] S.-W. Lee and B. Moon, "Design of Flash-based DBMS: An In-Page Logging Approach," ACM SOGMOD International Conference on Management of Data, Beijing, China, June 2007.

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

[9] T. Shinohara, "Flash Memory Card with Block Memory Address Arrangement," United States Patent, No. 5,905,993, 1999.

[10] A. Ban and R. Hasharon, "Flash File System Optimized for Page-mode Flash Technologies," United States Patent, No. 5,937,425, 1999.

[11] P. Estakhri and B. Iman, "Moving Sequential Sectors within a Block of Information in a Flash Memory Mass Storage Architecture," United States Patent, No. 5,930,815, 1999.

[12] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for Compact Flash System," IEEE Transactions on Consumer Electronics, Vol.48, No.2, pp. 366-375, 2002.

[13] T.-S. Chung, S. Park, M.J. Jung and B.-S. Kim, "STAFF: State Transition Applied Fast Flash Translation Layer," International Conference on Architecture of Computing Systems, pp. 199-212, 2004.

[14] T.-S. Chung, D.-J. Park, Y. Ryu and S. Hong, "LSTAFF: System Software for Large Block Flash Memory," Asia Simulation conference, pp. 704-712, 2004.

박 동 주

1995년 서울대학교 컴퓨터공학과(학사) 1997년 서울대학교 컴퓨터공학과(석사) 2001년 서울대학교 전기전자컴퓨터공학부(박사). 2001년~2003년 삼성전자 책임연구원. 2004년~2005년 숭실대학교 컴퓨터공학부 전임강사. 2006년~현재 숭실대학교 컴퓨터공학부 조교수. 관심분야는 플래시 메모리, 임베디드 데이타베이스, 멀티미디어 데이타베이스 등

곽 경 훈

2006년 숭실대학교 컴퓨터학과 졸업(학사). 2008년 숭실대학교 대학원 컴퓨터학과 졸업(공학석사). 2008년~현재 (주)알티베이스 연구원. 관심분야는 플래시 메모리, 임베디드 데이타베이스