섹터 매핑 기법을 적용한 효율적인 FTL 알고리듬 설계

준회원 윤 태 현*, 정회원 김 광 수**, 황 선 영*

Design of an Efficient FTL Algorithm for Flash Memory Accesses Using Sector-level Mapping

Tae-Hyun Yoon* *Associate Member*, Kwang-Soo Kim**, Sun-Young Hwang* *Regular Members*

요 약

본 논문은 플래쉬 메모리 접근시 erase 횟수를 줄이기 위하여 섹터 매핑 기법을 적용한 FTL (Flash Translation Layer) 알고리듬을 제안한다. 블록 매핑 기법을 적용한 기존의 알고리듬에 비하여 제안한 알고리듬은 섹터 단위의 매핑 테이블을 활용하여 데이터를 억세스하는 섹터 매핑 기법을 사용하여 erase 횟수를 줄임으로써 전체적인 메모리 억세스 시간을 줄이고 플래쉬 메모리의 수명을 연장시킬 수 있다. 제안한 알고리듬에서는 write를 위한 빈 공간이 없을 때 erase 횟수가 가장 적은 블록을 victim 블록으로 선택함으로써 wear-leveling을 구현하였다. 제안한 알고리듬을 검증하기 위하여 MP3 재생기, 동영상 재생기, 웹 브라우저, 문서 편집기의 어플리케이션에 대해 실험을 수행하였다. 제안한 알고리듬을 사용하였을 때 기존의 BAST, FAST 알고리듬과 비교하여 72.4%, 61.9%의 erase 횟수가 감소하였다.

Key Words : Flash Memory, FTL, Sector-level Mapping, Embedded System, File System

ABSTRACT

This paper proposes a novel FTL (Flash Translation Layer) algorithm based on sector-level mapping to reduce the number of total erase operations in flash memory accesses. The proposed algorithm can reduce the number of erase operations by utilizing the sector-level mapping table when writing data at flash memory. Sector-level mapping technique reduces flash memory access time and extendsthe life time of the flash memory. In the algorithm, wear-leveling is implemented by selecting victim blocks having the minimal number of erase operations, when empty spaces for write are not available. To evaluate the performance of the proposed FTL algorithm, experiments were performed on several applications, such as MP3 players, MPEG players, web browsers and document editors. The proposed algorithm reduces the number of erase operations by 72.4% and 61.9%, when compared with well-known BAST and FAST algorithms, respectively.

Ⅰ. 서 론

플래쉬 메모리는 저전력 소모, 비휘발성, 적은 면적으로 인한 높은 휴대성 등의 장점 때문에 MP3 player, PMP, PDA와 같은 휴대용 임베디드 시스템에서 저장매체로 널리 사용되고 있다. 최근에는 플래쉬 메모리 용량의 증가와 가격의 하락으로 일반 컴퓨터 시스템의 저장매체로도 각광받고 있다. 특히 전력 소모에 민감한 노트북에서 하드 디스크를 NAND 플래쉬 메모리에 기반한 반도체 디스크 (Solid State Disk)로 대체함으로써 많은 성능 향상을 얻고 있다[1],[2]. 하지만 플래쉬 메모리는 몇 가지

※ 본 논문은 2009년도 「서울시 산학연 협력사업」의 「나노IP/SoC설계기술혁신사업단」의 지원으로 이루어졌습니다.
* 서강대학교 전자공학과 CAD & ES 연구실 (hwang@sogang.ac.kr), ** 서강미래기술원
논문번호 : KICS2009-08-375, 접수일자 : 2009년 8월 27일, 논문최종접수일자 : 2009년 11월 11일

제약 사항 때문에 하드 디스크를 대체하는 데에 어려움이 있다.

NAND 플래쉬 메모리는 여러 개의 블록으로 구성되어 있다. 각각의 블록은 32개 이상의 섹터들로 구성되어 있다. 섹터는 read/write의 기본 단위이고, 블록은 erase의 기본 단위이다. 플래쉬 메모리의 첫 번째 문제는read, write, erase 동작에 소요되는 시간이 다르다. 예를 들면, 삼성 K9WBG08U1M NAND 플래쉬 메모리[3]의 경우 read 동작에 걸리는 시간은 25us, write 동작에 걸리는 시간은 200us, erase 동작에 걸리는 시간은 2ms이다. 이처럼 erase 동작에 걸리는 시간이 가 장 길어서 erase 동작의 횟수를 줄이는 것이 전체 소요 시간을 줄이기 위해서 가장 중요하다. 두번째 문제는 erase-before-write 문제이다. 플래쉬 메모리는 특정 섹터의 데이터가 수정되었을 때 일반적인 하드 디스크처럼 덮어쓸 수가 없다. 앞서 언급하였듯이 read와 write의 단위는 섹터이고 erase의 단위는 블록이므로 하나의 섹터에 데이터를 쓰기 위하여 블록을 erase하는 과정을 거쳐야만 한다. 플래쉬 메모리는 블록당 10만~100만 번의 erase 횟수 제한이 있으며 그 이후에는 결함이 생길 수 있다. 이와 같은 문제점들을 해결하기 위하여 FTL (Flash Translation Layer)의 역할이 중요하다.

FTL은 파일 시스템과 플래쉬 메모리 사이에 위치하며, FTL은 파일 시스템에서 처리하는 논리적인 주소를 플래쉬 메모리의 물리적인 주소로 변환하는 역할을 한다[4]. 블록당 erase 횟수에 제한이 있는 플래쉬 메모리의 특성에 따라 wear-leveling을 고려해야 하며, 전원 오류의 상황에서 데이터의 안정성을 보장해 주어야 한다. FTL의 가장 중요한 역할은 파일 시스템의 write 동작 요구에 대하여 플래쉬 메모리의 빈 공간을 찾아 write를 해 주고, 그에 따라 발생할 수 있는 erase 횟수를 줄이는 것이다. 동일한write 요구에도 각각의 FTL 알고리듬에 따라 read, write, erase 동작의 횟수가 달라지므로FTL은 플래쉬 메모리의 성능을 좌우한다.

FTL의 어드레스 매핑 기법에는 섹터 매핑 기법, 블록 매핑 기법, 하이브리드 매핑 기법이 있다[5]. 섹터 매핑 기법에서는 각각의 논리적 섹터 번호(LSN: Logical Sector Number)를 물리적 섹터 번호 (PSN: Physical Sector Number)로 매핑한다. 섹터 매핑 기법은 플래쉬 메모리 공간을 100% 활용하고, 다른 매핑 기법에 비하여 erase 동작 횟수가 가장 적어 속도면에서 성능이 뛰어나다는 장점이 있지만, 매핑 테이블의 크기가 커져서 많은 양의 SRAM을 필요로 한다는 단점이 있다. 블록 매핑 기법에서는 논리적 블록 번호(LBN: Logical Block Number)와 offset을 물리적 블록 번호(PBN: Physical Block Number)와 offset으로 매핑한다. 블록 매핑 기법은 매핑 테이블의 크기가 작아 비교적 적은 양의 SRAM을 필요로 하는 장점이 있으나, 파일 시스템이 같은 LSN에 대한 write 동작을 여러 번 요구할 경우 그 때마다 merge 동작을 해주어야 하는 단점이 있다. 하이브리드 매핑 기법은 블록 매핑 기법을 기반으로 하고 섹터 매핑 기법을 부분적으로 적용하는 방식으로 현재 대부분의 FTL들이 이 방식을 적용하고 있다.

섹터 매핑 기법은 많은 양의 SRAM을 필요로 하여 가격 면에서 단점을 보이지만, 일반컴퓨터 시스템의 반도체 디스크(SSD)를 기준으로 할 경우 이는 큰 문제가 되지 않는다. 기존의 섹터 매핑 기법은 m개의 섹터를 갖는 플래쉬 메모리에 대해m열의 매핑 테이블을 필요로 하고 merge 동작이 일어날 때 victim 블록 선정 기준이 효율적이지 않다. 본 논문은 동적 매핑 기법을 적용하여 매핑 테이블의 크기를 줄이고 각 블록당 invalid한 섹터의 개수(ISN : Invalid Sector Number)와 erase 횟수(ECN : Erase Count Number)에 대한 테이블을 두어, erase 및 copy의 횟수를 줄이고 wear-leveling을 고려한 새로운 섹터 매핑 기법을 제안한다.

본 논문은 다음과 같이 구성된다. 2절에서는 기존의 제안된 FTL의 특징을 설명하고, 3절에서는 본 논문에서 제안된 FTL의 구조와 알고리듬을 설명한다. 4절에서는 각각의 어플리케이션마다 erase 횟수, 전체 소요 시간을 측정한 실험 결과를 보인다. 마지막으로 5절에서는 결론을 제시한다.

Ⅱ. 관련 연구

FTL은 플래쉬 메모리의 성능에 중요한 요소임에 따라, 그 동안 FTL에 관한 많은 연구가 진행되어 왔다. BAST[2]는 write 동작을 위한 일종의 캐쉬 역할을 하는 로그 블록을 두는 형태의 FTL이다. BAST는 데이터 블록과 로그 블록 사이에 블록간의 연관성(Block-associativity)을 가진다. 즉, 하나의 로그 블록은 하나의 데이터 블록에 해당하는 데이터만 저장할 수 있다. BAST는 순차적인 write 동작 요구 발생시에 switch merge가 발생하여 좋은 성능을 갖지만, 임의의 write 동작 요구 발생시에 낮은

성능을 가진다. BAST의 단점을 보완하기 위하여 FAST[6]가 제안되었다. FAST는 로그 블록을 순차적인 write를 위한 로그 블록과 임의의 write를 위한 로그 블록으로 나눈다. 임의의 write를 위한 로그 블록은 섹터 간의 연관성(Full-associativity)을 가짐으로써 로그 블록 공간의 효율성을 높였다. FAST는 BAST에 비해 erase 횟수를 줄이고 성능이 향상되었지만, 섹터 매핑 기법을 사용할 때보다 erase 횟수가 많아 성능이 떨어지고, 메모리 공간 활용을 최대화하지 못한다.

EAST[7]는 state transition table과 reallocation block을 활용하여 메모리 공간의 활용도를 높였다. EAST는 FAST에 비해 임의의 write에 대한 성능은 높으나, 순차적인 write를 위한 로그 블록의 부재로 순차적인 write의 경우 FAST보다 성능이 떨어진다. LAST[8]는 temporal locality와 spatial locality를 고려하여 garbage collection 오버헤드를 줄이려고 노력하였다. LAST도 로그 버퍼를 기반으로 하며, 임의의 write를 위한 로그 버퍼를 hot partition과 cold partition으로 나누었다. LAST는 locality를 고려하여 기존의 FTL에 비해 성능이 향상되었으나 locality detector의 오버헤드가 크고 섹터 매핑 기법을 적용하였을 때보다 공간 활용도 및 속도가 떨어진다.

본 논문에서는 섹터 매핑 기법을 적용하여 기존의 블록 매핑 기법에 기반한 FTL보다 플래쉬 메모리 공간의 활용도를 높이고 erase 횟수를 줄여 전체 소요 시간을 줄이는 방법을 제안한다.

Ⅲ. 제안한 FTL 알고리듬

본 절에서는 제안한 FTL 알고리듬에 관해 기술한다. 3.1절에서는 제안한 매핑 기법에 대해 기술하고, 3.2절에서는 어드레스 매핑을 위한 섹터 매핑 테이블과 merge 동작 시 효율적인 victim 블록선정을 위한 블록 정보 테이블에 대해 기술한다. 3.3절에서는 제안한 FTL의read, write, merge 동작에 대한 알고리듬을 기술한다.

3.1 제안한 매핑 기법

제안한 FTL 알고리듬은 섹터 매핑 기법을 사용한다. 기존의 섹터 매핑 테이블은 초기에 모든 LSN에 대한 PSN의 맵을 가지고 있어, m개의 LSN이 존재하는 시스템에서 이와 같은 매핑 기법을 사용할 경우 m열의 매핑 테이블을 필요로 한다[5]. 이에 비해 제안한 FTL은 파일 시스템으로부터 write 요구가 들어온 LSN에게 빈 공간의 PSN을 할당하고 할당된 LSN에 대한 매핑 정보만 가진다. 동적 할당 기법을 사용하면 전체 LSN의 수와 비교하여 write 동작이 요구되는 LSN의 수는 적으므로 섹터 매핑 기법의 단점인 매핑 테이블 크기를 줄일 수 있다.노트북 PC에서 mp3 재생기, 동영상 재생기, 웹 브라우저, 문서 편집기의 어플리케이션에 대하여 실험한 결과 표 1과 같이 각 어플리케이션마다 전체 LSN의 수 중 30.3%, 69.6%, 61.3%, 64.2%의 LSN에 억세스하였다. 실험 결과가 제시하듯이 중복되는 LSN에 대한 write 요구가 많으므로 동적 할당 기법을 적용함에 따라 매핑 테이블이 차지하는 SRAM의 공간을 줄일 수 있다.

3.2 섹터 매핑 테이블과 블록 정보 테이블

제안한 FTL은 섹터 매핑 테이블과 블록 정보 테이블을 갖는다. 섹터 매핑 테이블은 논리적 주소와 물리적 주소간의 매핑을 위해 필요하며, 블록 정보 테이블은 merge 동작 시효율적인 victim 블록 선정을 통해 erase와 copy 동작의 수를 줄이기 위하여 존재한다.

블록 정보 테이블은 각각의 PBN에 대한 ISN (Invalid Sector Number)와 ECN (Erase Count Number)을 포함하며, ISN은 해당 블록에 저장되어 있는 데이터 중 쓸모 없는 섹터의 수이다. 플래쉬 메모리에 존재하는 LSN에 대한 write 요구 발생 시 기존의 데이터는 최신의 데이터가 아니기 때문에 쓸모 없어진다. 이런 데이터들은 merge 동작 시 erase 한 후 copy 동작을 할 필요가 없으므로 merge 동작 후 빈 공간을 확보할 수 있고 write 횟수를 줄일 수 있다. ECN은 해당 블록이 처음부터 현재까지 erase 된 횟수를 기록한다. Merge 동작 시 ECN이 가장 적은 블록을 victim 블록으로 선택함으로써 wear-leveling을 지원할 수 있다. 이처럼

표 1. Write 입력의 수와 write 된 LSN의 수 비교

	Write 동작의 수	전체 LSN의 수	Write된 LSN의 수
어플리케이션 #1 (MP3 재생기)	683,100	524,288	158,807 (30.3%)*
어플리케이션 #2 (동영상 재생기)	822,384	524,288	321,389 (61.3%)*
어플리케이션 #3 (웹 브라우저)	3,376,656	524,288	364,747 (69.6%)*
어플리케이션 #4 (문서 편집기)	1,262,400	524,288	336,488 (64.2%)*

*전체 LSN 중 write된 LSN의 비율을 나타낸다.

Merge 동작 시 ISN이 가장 많고 ECN이 가장 적은 블록을 victim 블록으로 선택하면 erase 및 copy의 횟수를 줄이고 wear-leveling을 고려할 수 있다. 그림 1의 예를 통해 섹터 매핑 테이블과 블록 정보 테이블에 대해 자세히 설명한다.

그림에서 블록 당 섹터의 수는 4이고 전체 블록의 수는 4라고 가정한다. 좌측의 시퀀스는 파일 시스템이 요구하는 write 동작의 패턴을 나타낸다. Write (LSN, Data) 는 논리적 섹터 LSN에 Data를 write함을 의미한다. 처음 write 요구가 들어오기 전에 섹터 매핑 테이블은 빈 상태이고 블록 정보 테이블의 Full, ISN, ECN은 모두 0으로초기화 되어야 한다. 첫 번째 write 동작이 호출되었을 때, FTL은 플래쉬 메모리의 빈 섹터를 찾아 write 한 후 섹터 매핑 테이블에 기록한다. 처음에는 모든 섹터가 비어있으므로 LSN 2는 PSN 0에 매핑된다. 단계 2와 단계 3도 같은 방법으로 진행된다. 단계 4의 경우 이미 LSN 2의 데이터가 플래쉬 메모리에 있으므로, PSN 0에 “Invalid (-1)”표시를 한 후 빈 섹터(PSN 3)를 할당한다. 섹터 매핑 테이블에 이를 업데이트하고 블록 정보 테이블의 ISN을 1로 업데이트한다. 한 개의 빈 블록이 남아있을 때까지 위와 같은 과정을 거친다. 그림 1의 예에서 단계 12까지 마치면 한 개의 빈 블록을 남겨두고 나머지 블록이 모두 채워지게 된다. 이 때의 섹터 매핑 테이블과 블록 정보 테이블은 그림 1의 상단과 같다. 단계 13에서 더 이상 빈 섹터가 존재하지 않기 때문에 merge 동작이 요구된다. Victim 블록을 선택하기 위해 FTL은 블록 정보 테이블을 참조한다. 모든 블록의 ECN이 같으므로 가장 많은 ISN을 갖는 PBN 0이 victim 블록으로 선택되고, valid 섹터인 PSN 3의 데이터를 빈 블록의 첫 번째 섹터 (PSN 12)에 copy한다. PBN 0를 erase하고 섹터 매핑 테이블과 블록 정보 테이블을 업데이트하면 그림 1의 하단과 같다.

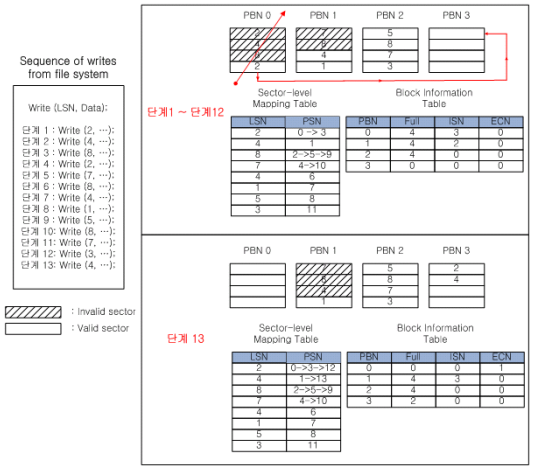


그림 1. 섹터 매핑 테이블과 블록 정보 테이블

제안한 FTL은 동적 할당 기법을 적용하여 섹터 매핑 테이블의 크기를 줄이고, 블록 정보 테이블을 이용하여 효율적인 victim 블록 선정을 통해 erase와 copy의 횟수를 줄일 수 있다. 기존의 블록 매핑 기법을 적용한 BAST, FAST에서 full merge의 경우 2번의 erase와 valid한 섹터 수만큼의 copy 동작이 필요했으나[2,6] 제안한 FTL의 merge 동작은 ISN이 가장 많은 블록을victim 블록으로 선택하여 한 번의 erase와 valid한 섹터 수만큼의 copy 동작을 요구한다. ISN이 블록 당 섹터의 수와 같은 블록을 victim 블록으로 선택될 경우 switch merge와 같이 한 번의 erase 동작만 필요하게 된다. 3.3절에서는 제안한 FTL의 read, write, merge 동작에 대한 알고리듬을 설명한다.

3.3 주요 동작의 알고리듬

3.3.1 Write 동작

본 절에서는 파일 시스템이 write 동작을 요구할 때, 제안한 FTL이 처리하는 알고리듬을 설명한다. 3.2절에서 언급하였듯이 첫 번째 write 동작이 수행되기 전에 플래쉬 메모리의 모든 섹터와 섹터 매핑 테이블은 비어 있고 블록 정보 테이블의 ISN, ECN은 모두 0으로 초기화 되어 있다.

제안한 FTL의 write 알고리듬의 pseudo code를 그림 2에 제시하였다. 파일 시스템이 write 동작을 요청하면 FTL은 빈 블록(free block)이 한 개 이상 존재하는지를 검사하고 하나의 빈 블록을 제외하고 빈 섹터가 존재하는 것을 확인한다. 하나의 빈 블록은 merge 동작 시 victim 블록의 valid한 데이터들을 복사하기 위해 존재하여야 한다. 위의 두 조건이 만족되면 빈 섹터의 PSN이 write 동작이 수행될 LSN에 순서대로 매핑된다. LSN이 섹터 매핑 테이블에 존재할 경우 이전에 매핑된 PSN은 invalid 상태로 표시한 후 블록 정보 테이블에서 해당 블록의 ISN을 업데이트한다. 이 과정을 마친 후 새로 매핑된 PSN에 데이터를 쓰고 섹터 매핑 테이블을 업데이트한다. 이전에 매핑된 PSN에 대한 정보는 필요 없으므로 그 자리에 새로 매핑된 PSN을 덮어 쓰며,

```
Procedure  Write_at_Sector (LSN, Data)
// Data를 LSN (Logical Sector Number)에 write한다
begin
   Nfree = CountFreeBlock();
   EmptyPSN = SearchEmptyPSN ();
   if (EmptyPSN exists && Nfree >=1) then
   // Free block이 존재하고 빈 섹터가 있을 때 Data를 write하고 SMT,BIT 경신
      Search LSN from SMT;
      if (LSN exists in SMT) then
         Mark the PSN mapped to LSN as invalid (-1);
         Call Update_BIT (PSN);
      end if
      Write Data to EmptyPSN;
      Call Update_SMT (LSN, EmptyPSN);
   end if
   else
   //빈 섹터가 없을 때 victim blocks를 merge한 후 write한다.
      Merge_Blocks ();
      Write_at_Sector (LSN, Data);
   end if
end;
```

그림 2. 제안한 FTL의 write 알고리듬

빈 섹터가 존재할 때까지 같은 과정을 반복하고 빈 섹터가 없을 경우 merge 동작을 수행한다.

3.3.2 Merge 동작

Merge 동작은 플래쉬 메모리에 더 이상 데이터를 쓸 수 있는 공간이 없을 때, 빈 공간을 확보하기 위해 필요하다. 로그 버퍼를 기반으로 하는 기존의 BAST, FAST의 경우 merge 동작은 크게 full merge, switch merge로 구분된다.[2],[6] Full merge의 경우 2번의 erase 동작이 필요하고 switch merge의 경우 1번의 erase 동작이 필요하다. 제안한 FTL은 merge 동작 시 1번의 erase 동작을 필요로 한다. 그림 3은 merge 동작의 pseudo code를 보인다. Merge 동작을 수행하기 위한 victim 블록의 선정 결과에 따라 merge 동작의 횟수와 copy 동작의 횟수가 달라지므로 효율적인 victim 블록 선정 알고리듬이 필요하다.

제안한 FTL은 victim 블록 선정 시 블록 정보 테이블의 ISN, ECN의 두 가지 정보를 참조한다. ISN은 특정 블록의 섹터 중 invalid한 데이터를 갖고 있는 섹터의 수이다. ISN이 큰 블록을 선택하면 merge 동작 수행 후 많은 공간을 확보할 수 있고 invalid 섹터는 copy 동작이 필요 없으므로 copy 동작의 수를 줄일 수 있다. ECN은 해당 블록이 현

```
Procedure  Merge_Blocks ()
// Victim block 2개를 선정하여 merge한다.
begin
   FreePBN = SelectFreeBlock ();
   VictimPBN = Select_Victim_Block ();
   for (each sector in VictimPBN)
   begin
   // victim 블록의 섹터 데이터가 vaild할 때만 free 블록에 copy한다
      if (data of PSN is valid) then
         CopyData (PSN, FreePBN )
         Call Update_SMT ();
      end if
   end for;
   EraseBlock (VictimPBN);
   Call Update_BIT ();
end;

Procedure  Select_Victim_Block ()
begin
   for (each PBN in Block Information Table)
   begin
      VictimPBN  =SearchMaxISN
      //ISN이 가장 큰 블록을 찾는다.
      if (more than 2 PBNs contain the maximum ISN) then
         VictimPBN = SearchMinECN;
         //ECN이 가장 작은 블록을 찾는다.
      end if
   end for;
   return VictimPBN;
end;
```

그림 3. 제안한 FTL의 merge 알고리듬

재까지 erase된 횟수로서, ECN이 작은 블록을 victim 블록으로 선정함으로써 wear-leveling을 고려하였다. 위의 두 가지 요소를 고려하여 victim 블록 선정 시 ISN이 가장 큰 블록을 검색하고, 최대의 ISN을 갖는 블록이 두 개 이상일 경우 ECN이 작은 블록을 선택하도록 설계하였다.

Victim 블록이 선정되면 merge 동작을 수행한다. Merge 동작이 수행되는 과정은 그림 4에 보인다. 각각의 섹터는 invalid 섹터를 표시하기 위한 flag를 갖고 있다. FTL은 victim 블록의 섹터마다 flag를 확인하고 valid한 섹터의 데이터를 빈 블록으로 copy한다. Copy된 데이터에 대해 섹터 매핑 테이블

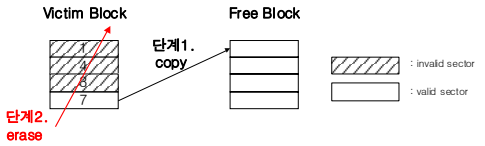


그림 4. 제안한 FTL의 merge 동작

을 업데이트하고 victim 블록을 erase 한 후 블록 정보 테이블을 업데이트한다. 이 때 erase된 victim 블록은 빈 블록 리스트에 추가된다.

3.3.3 Read 동작

블록 매핑 기법을 적용한 FTL에서는 read 동작 시 데이터 블록의 해당 위치를 읽었을 때 invalid 상태일 경우 로그 버퍼에서 해당 LSN에 대한 데이터를 다시 찾아야 한다. 본 연구에서 제안한 FTL은 섹터 매핑 기법을 적용하여 섹터 매핑 테이블에서 read 동작이 요구된 LSN에 매핑된 PSN을 바로 찾을 수 있으므로 read 동작에 필요한 시간을 단축할 수 있다. 섹터 매핑 테이블에는 최근에 write된 PSN에 대한 정보가 있으므로, 섹터 매핑 테이블에 있는 PSN을 그대로 읽어 들이면 된다. 제안한 FTL은 동적 할당 기법을 사용하므로 이전에 write 되지 않은 LSN은 섹터 매핑 테이블에 존재하지 않는다. 섹터 매핑 테이블에 존재하지 않는 LSN에 대한 read 요구가 들어올 경우 FTL은 파일 시스템에 no data 정보를 return한다.

Ⅳ. 실험 결과

제안한 FTL 알고리듬을 검증하기 위해 BAST[2], FAST[6]와 각각 비교하였다. BAST, FAST와 제안한 FTL 각각을 시뮬레이터로 구현하여 다양한 트레이스 파일들을 적용함으로써 성능을 평가하였다. 실험에 쓰일 플래쉬 메모리 모델은 삼성 K9WBG08U1M large block NAND 플래쉬 메모리[3]를 사용하였으며, 중요 파라미터들은 표 2에 제시하였다. 실험에 쓰인 disk access pattern은 windows-XP를 기반으로 한 노트북 PC에서 MP3 재생기, 동영상 재생기, 웹 브라우저, 문서 편집기의 어플리케이션을 대상으로 실험하여 추출하였다. 각각의 어플리케이션에 대한 트레이스 파일의 특징은 3 절의 표 1과 같다. 실험에 사용한 플래쉬 메모리는 1GB (=524,288 sectors)의 용량을 사용하였고, 어플리케이션마다 write 요구가 발생한 LSN은 전체 LSN의 수에 비해 각각30.3%, 42.3%, 69.6%, 64.2%이다. 어플리케이션 #1, #2는 순차적인 write 요구가 많았으며, 어플리케이션 #3, #4는 임의의 write 요구가 많았다.

각각의 어플리케이션에 대한 erase 횟수 비교는 표 3에 제시되었으며 그림 5는 실험 결과의 그래프를 보인다. 제안한 FTL을 BAST, FAST와 비교하

표 2. 실험에 쓰인 NAND 플래쉬 메모리의 중요 파라미터

NAND 플래쉬 메모리의 구성	Block size	128KB
	Sector size	2KB
	하나의 block 당 page의 수	64
각각의 operation의 access time	Read 연산	25 usec
	Write 연산	200 usec
	Erase 연산	2000 usec

표 3. 어플리케이션 별 erase 횟수 비교

	BAST 알고리듬	FAST 알고리듬	제안한 FTL 알고리듬
어플리케이션 #1 (MP3 재생기)	18,381	15,072	4,178 (-77.3%, -72.3%)*
어플리케이션 #2 (동영상 재생기)	29,729	19,914	9,314 (-68.7%, -53.2%)*
어플리케이션 #3 (웹 브라우저)	220,336	143,219	62,292 (-71.7%, -56.5%)*
어플리케이션 #4 (문서 편집기)	68,872	56,591	19,422 (-71.8%, -65.7%)*

*비교 대상과 제안한 FTL의 erase 횟수 증감을 나타낸다.

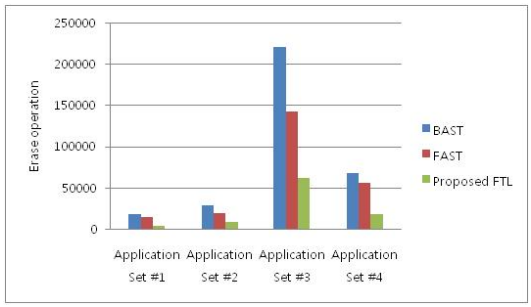


그림 5. 성능 비교

였을 때 평균 72.4%, 61.9%의 erase 횟수가 감소하였다. 어플리케이션 #1과 같이 특정 LSN에 대한 write 요구가 중복될 경우 77.3%, 72.3%로 가장 많은 성능 향상을 얻을 수 있었다. BAST[2]와 FAST[6]는 순차적인 write 요구가 많을 때 erase 횟수가 줄어드나 실험에 사용한 disk access pattern을 분석하였을 때 순차적인 write 요구는 많지 않았다. 제안한 FTL은 순차적인 write 요구에 상관 없이, 이미 write된 LSN에 대한 write 요구가 많을수록 erase 횟수가 줄어든다.

제안한 FTL은 섹터 매핑 기법을 적용하여 위와 같은 성능 향상을 얻었으나 블록 매핑 기법을 사용하였을 때보다 매핑 테이블의 크기가 커지는 단점

이 있다. 32GB의 NAND 플래쉬 메모리 기반의 반도체 디스크를 기준으로 매핑 테이블의 크기를 BAST[2], FAST[6]와 비교하였다. BAST와 FAST의 로그 버퍼의 크기는 1 GB로 가정하였다. 제안한 FTL은 동적 할당 기법을 사용하여 매핑 테이블의 크기가 일정한 것은 아니지만 모든 LSN의 데이터가 write 되었다고 가정하면 매핑 테이블의 크기는 8*1024*32*64*2B = 33 MB 가 된다. BAST[1]의 경우 매핑 테이블의 크기는 8*1024*32*2B = 524,288B 이고 FAST[6]는 8*1024*31*2B + 8*1024*64*2B = 1,556,480B의 크기를 갖는다. 매핑 테이블의 크기와 플래쉬 메모리의 용량(32 GB)의 합을 비교해 보면, 제안한 FTL을 사용할 경우 BAST, FAST에 비해 최대 0.96%, 0.93%의 면적이 증가된다. 앞서 언급하였듯이 제안한 FTL은 동적 할당 기법을 사용하여 매핑 테이블의 크기를 줄인다. 표 4는 본 논문에서 사용한 각각의 어플리케이션에 대한 면적 증감을 나타낸다. 실험 결과에서 보듯이 0.96%, 0.93% 보다 작은 면적 증감이 나타남을 알 수 있다. 제안한 FTL을 사용할 때 전체 LSN의 수 중 write된 LSN의 수가 작을수록 면적이 감소한다.

표 4. 어플리케이션 별 면적 비교

	BAST 알고리듬	FAST 알고리듬	제안한 FTL 알고리듬
어플리케이션 #1 (MP3 재생기)	32GB + 524,288B	32GB + 1,556,480B	32GB + 10,163,638B (+0.28%, +0.25%)*
어플리케이션 #2 (동영상 재생기)	32GB + 524,288B	32GB + 1,556,480B	32GB + 20,568,867B (+0.58%, +0.55%)*
어플리케이션 #3 (웹 브라우저)	32GB + 524,288B	32GB + 1,556,480B	32GB + 20,568,867B (+0.58%, +0.55%)*
어플리케이션 #4 (문서 편집기)	32GB + 524,288B	32GB + 1,556,480B	32GB + 20,568,867B (+0.58%, +0.55%)*

*비교 대상과 제안한 FTL의 erase 횟수 증감을 나타낸다.

V. 결 론

본 논문에서는 섹터 매핑 기법과 블록 정보 테이블을 적용한 FTL 알고리듬을 제안하였다. 기존의 블록 매핑 기법을 기반으로 하는 FTL은 매핑 테이블의 크기를 줄여 적은 양의 RAM을 필요로 하지만, erase 횟수 및 플래쉬 메모리 억세스를 위한 소요 시간은 섹터 매핑 기법을 사용하였을 때보다 증가하게 된다[5]. 제안한 FTL은 섹터 매핑 기법을 적용하고 merge 동작 발생 시 블록 정보 테이블의 ISN과 ECN을 참조하여 victim 블록을 선정함으로써 erase 횟수를 줄여 플래쉬 메모리 억세스에 필요한 전체 소요 시간을 줄이고 wear-leveling을 지원한다. 섹터 매핑 기법의 단점인 매핑 테이블의 크기를 줄이기 위하여 동적 할당 기법을 사용하였다. 기존의 BAST[2], FAST[6]와 비교하기 위해 mp3 재생기, 동영상 재생기, 웹 브라우저, 문서 편집기의 어플리케이션을 대상으로 실험하였다. 본 연구에서 제안하는 FTL은 기존의 BAST, FAST와 비교하여 각각 평균 72.37%, 61.93%의 erase 동작의 횟수를 줄였으며, 면적에서는 BAST, FAST에 비해 최대 0.96%, 0.93%의 작은 증가를 보였다. 이는 전체 플래쉬 메모리 시스템에 비추어 볼 때 미미한 수준의 증감이다. 제안한 FTL을 사용하여 erase 횟수를 줄임으로써 플래쉬 메모리 억세스에 필요한 시간을 줄일 수 있다.

본 연구에서 제안하는 FTL은 기존에 write된 LSN에 대한 write 요구가 많을수록 성능이 향상된다. 추후에 write 요구의 지역성을 고려하여 어드레스를 할당하는 기법을 연구할 필요가 있다.

참 고 문 헌

〔1〕 F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, and JA. A. Tauber, "Storage Alternatives for Mobile Computers," In Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI), 1994.

〔2〕 J. Kim, J. Kim, S. Noh, S. Min, and Y. Cho, "A space-efficient flash translation layer for compact flash systems," *IEEE Transactions on Consumer Electronics*, Vol.48, No.2, pp.366-375, May 2002.

〔3〕 Samsung Electronics, "2G x 8Bit / 4G x 8 Bit / 8G x 8 Bit NAND Flash memory (K9WBG08U1M) Data Sheets," 2007.

〔4〕 E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys (CSUR)*, Vol.37, No.2, pp.138-163, June 2005.

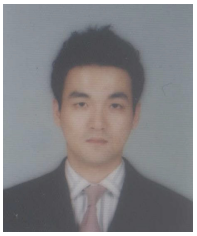
〔5〕 T. Chung, D. Park, S. Park, D. Lee, S. Lee, and H. Song, “System software for flash memory: A survey,” *IFIP Int. Conf. Embedded and Ubiquitous Computing, Lecture Note in Computer Science (LNCS)*, Vol.4096, pp.394-404, Springer-Verlag, 2006.

〔6〕 S. Lee, D. Park, T. Chung, W. Choi, D. Lee, S. Park, and H. Song, “A log buffer based flash translation layer using fully associative sector translation,” *ACM Transactions on Embedded Computing Systems*, Vol.6, No.3, July 2007.

〔7〕 S. Kwon and T. Chung, “An efficient and advanced space-management technique for flash memory using reallocation blocks,” *IEEE Transactions on Consumer Electronics*, Vol.54, No.2, pp.631-638, May 2008.

〔8〕 S. Lee, D. Shin, Y. Kim, and J. Kim, “LAST: Locality-aware sector translation for NAND flash memory-based storage systems,” *ACM SIGOPS Operating Systems Review*, Vol.42, No.6, pp.36-42, Feb. 2008.

윤 태 현 (Tae-Hyun Yoon) 준회원



2005년 2월 서강대학교 전자공학과

2008년 3월~현재 서강대학교 전자공학과 석사과정

<관심분야> Flash memory, Embedded System

김 광 수 (Kwang-Soo Kim) 정회원

1981년 서강대학교 전자공학과 학사

1983년 서강대학교 대학원 전자공학과 석사

1992년 서강대학교 대학원 전자공학과 박사

1983년~1997년 한국전자통신연구원 책임연구원

1998년~2005년 정보통신연구진흥원 책임연구원

2005년~2008년 대구경북과학기술원 책임연구원

2008년~현재 서강대학교 교수

<관심 분야> 지능형센서시스템, 센서 네트워크

황 선 영 (Sun-Young Hwang) 정회원

1976년 2월 서울대학교 전자공학과

1976년 2월 한국과학원 전기 및 전자공학과 공학석사

1986년 10월 미국 Stanford 대학교 전자공학 박사학위

1976년~1981년 삼성 반도체(주) 연구원, 팀장

1986년~1989년 Stanford 대학 Center for Integrated Systems 연구소 책임 연구원 및 Fairchild Semiconductor, Palo Alto Research Center 기술자문

1989년~1992년 삼성전자(주) 반도체 기술자문

2002년 4월~2004년 3월 서강대학교 정보통신대학원장

1989년 3월~현재 서강대학교 전자공학과 교수

<관심분야> SoC 설계 및 framework 구성, CAD 시스템, Embedded 시스템, DSP 시스템 설계 등