AFTL: Hot Data 검출기를 이용한 적응형 플래시 전환 계층

(AFTL: An Efficient Adaptive Flash Translation Layer using Hot Data Identifier for NAND Flash Memory)

윤 현 식 † 주 영 도 †† 이 동 호 †††
(Hyun-Sik Yun) (Young Do Joo) (Dong-Ho Lee)

요 약 최근 NAND 플래시 메모리는 빠른 접근속도, 저 전력 소모, 높은 내구성, 작은 부피, 가벼운 무게 등으로 차세대 대용량 저장 매체로 각광 받고 있다. 그러나 이런 플래시 메모리는 데이타를 기록하기 전에 기존의 데이타 영역이 지워져 있어야 한다는 제약이 있으며, 비대칭적인 읽기, 쓰기, 삭제 연산의 처리속도, 각 블록당 최대 소거 횟수 제한과 같은 특징들을 지닌다. 위와 같은 단점을 극복하고 NAND플래시 메모리를 효율적으로 사용하기 위하여, 다양한 플래시 전환 계층 제안되어 왔다. 그러나 기존의 플래시 전환 계층들은 Hot data라 불리는 빈번히 접근되는 데이타에 의해서 잦은 겹쳐쓰기 요구가 발생되며, 이는 급격한 성능 저하를 가져 온다. 본 논문에서는 Hot data 검출기를 이용하여, 매우 적은 양의 데이타인 Hot data를 검출한 후, 검출된 Hot data는 섹터사상 기법을 적용시키고, 나머지 데이타인 Cold data는 로그 기반 블록 사상 기법을 적용시키는 적응형 플래시 전환 계층(AFTL)을 제안한다. AFTL은 불필요한 삭제, 쓰기, 읽기 연산을 최소화시켰으며, 기존의 플래시 전환 계층과의 비교 측정을 통하여 성능의 우수성을 보인다.

키워드 : 플래시 메모리, 플래시 전환 계층, Hot data, 주소사상 기법

Abstract NAND Flash memory has been growing popular storage device for the last years because of its low power consumption, fast access speed, shock resistance and light weight properties. However, it has the distinct characteristics such as erase-before-write architecture, asymmetric read/write/erase speed, and the limitation on the number of erasure per block. Due to these limitations, various Flash Translation Layers (FTLs) have been proposed to effectively use NAND flash memory. The systems that adopted the conventional FTL may result in severe performance degradation by the hot data which are frequently requested data for overwrite in the same logical address. In this paper, we propose a novel FTL algorithm called Adaptive Flash Translation Layer (AFTL) which uses sector mapping method for hot data and log-based block mapping method for cold data. Our system removes the redundant write operations and the erase operations by the separating hot data from cold data. Moreover, the read performance is enhanced according to sector translation that tends to use a few read operations. A series of experiments was organized to inspect the performance of the proposed method, and they show very impressive results.

Key words : Flash memory, Flash Translation Layer, Hot data, Address mapping

· 본 연구는 정통부 및 정보통신연구진흥원의 정보통신 선도기반기술개발사업의 연구결과로 수행되었습니다.
· 이 논문은 2007 한국컴퓨터종합학술대회에서 'AFTL: Hot Data 검출기를 이용한 적응형 플래시 전환 계층'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 한양대학교 컴퓨터공학과
hsyoon@cse.hanyang.ac.kr
†† 정 회 원 : 강남대학교 컴퓨터미디어공학부 교수
ydjoo@kangnam.ac.kr
††† 정 회 원 : 한양대학교 컴퓨터공학과 교수
dhlee72@cse.hanyang.ac.kr

논문접수 : 2007년 9월 27일
심사완료 : 2007년 11월 22일

정보과학회논문지: 시스템 및 이론 제35권 제1호(2008.2)

1. 서 론

플래시 메모리는 기존의 저장매체의 훌륭한 대용물로 떠오르고 있다. 휴대용 전화기나 개인 정보 단말기(PDA)와 같은 소형 이동 기기에서의 사용뿐만 아니라, NAND플래시 메모리를 이용한 반도체 디스크(Solid State Disk)와 같은 매체는 노트북과 같은 전력소모에 민감한 장치에 뛰어난 성능향상을 가져올 것으로 기대되고 있다[1,2]. NAND 플래시 메모리는 외부 충격이나 전자기장 등으로 인해 발생할 수 있는 물리적인 손상에 강하여 높은 신뢰성을 보장하고, 부피와 소비전력이 매우 적다. 또한 비 휘발성 반도체 소자를 이용한 빠른 데이타 접근을 가능하게 한다. 빠르게 대용량화 되고 있는 NAND 플래시 메모리의 이러한 특징들로 인해서 향후 보다 많은 분야에서 사용될 것으로 기대된다.

플래시 메모리는 구성하는 반도체의 특성에 따라서 NOR형 플래시 메모리, NAND형 플래시 메모리, AND형 플래시 메모리, DINOR형 플래시 메모리 등으로 구분된다. 보편적으로 사용되고 있는 플래시 메모리는 NOR형과 NAND형으로서, NOR형 플래시 메모리는 일반적으로 프로그램 수행 용도로 사용되고(eXecute In Place), NAND 플래시 메모리는 데이타 저장용으로 사용된다.

NAND 플래시 메모리는 여러 개의 블록으로 이루어져 있고, 한 블록은 32개의 섹터로 구성되어있다. 하나의 섹터는 512Byte의 데이타 영역과 16Byte로 이루어진 예비(spare) 영역으로 이루어져 있다. 그리고 플래시 메모리는 블록당 10만~100만 번의 최대 소거 횟수 제한이 있으며, 그 이후에는 결함(bad)이 생길 수 있다[3]. 또한 플래시 메모리는 디스크와는 다르게 읽기와 쓰기, 삭제 연산간의 처리속도가(읽기: 35.9μs, 쓰기: 226μs, 삭제: 2ms) 비대칭적이다[2].

플래시 메모리를 사용하는데 있어서 가장 큰 제약은 데이타 기록을 위한 갱신이 불가능하여, 기록하기 전에 반드시 기존의 데이타 영역이 지워져 있어야 하는 특별한 구조(erase-before-write architecture)를 지닌다는 것이다. 따라서 디스크 기반으로 개발되어 있는 시스템을 그대로 플래시 메모리 상에 사용하는 것은 불가능하다. 이러한 문제점을 해결하기 위해서 플래시 전환 계층[2,4-6]에 관한 연구가 계속 되어 왔다.

플래시 전환 계층은 논리적인 주소를 물리적인 주소로 사상하는 기법을 제공하고, 전원 오류 등의 상황에서의 데이타 안정성을 보장하며, 최대 소거 횟수 제약을 극복하기 위해 균등한 쓰기를 보장해 주는 Wear-leveling 기능을 제공한다. 플래시 전환 계층에서 사용되는 주소사상 기법으로는 크게 섹터사상(sector mapping), 블록사상(block mapping), 하이브리드 사상(hybrid mapping) 기법으로 나눠 볼 수 있다[7]. 섹터사상은 가장 기본적인 형태의 사상기법으로서, 모든 논리주소와 모든 물리주소를 1:1로 사상하는 기법이다. 속도 면에서는 최고의 성능을 보이는 사상기법이지만, 사상테이블(mapping table) 이 너무 커지는 오버헤드를 가지기 때문에 순수한 섹터사상 기법을 실제의 시스템에 적용하기는 어렵다. 블록사상은 논리적 블록과 물리적 블록을 1:1로 사상하고, 섹터는 오프셋(offset)을 이용하여 찾아가는 방식으로 사상테이블의 크기를 줄일 수 있지만, 같은 주소에 겹쳐쓰기(overwrite) 연산이 일어났을 경우 매번 Copy-back 연산이 발생하기 때문에 성능 면에서 비효율적이다. 따라서 현재 대부분의 플래시 전환 계층은 블록사상 방식에 섹터사상 방식의 장점을 부분적으로 적용시키는 하이브리드 방식의 사상기법을 사용하고 있다.

플래시 전환 계층은 플래시 메모리의 성능을 좌우하는 중요한 열쇠가 된다. 동일한 쓰기요구(write request) 에도 각각의 플래시 전환 계층의 알고리즘에 따라서 일어 날 수 있는 쓰기, 읽기, 삭제 연산의 횟수가 상이하게 나타나기 때문이다. 따라서, 플래시 전환 계층을 설계할 때에는 비대칭적인 연산속도를 지니는 플래시 메모리의 특성을 반영하여, 읽기 연산은 일정 수준 이상 발생하더라도, 쓰기와 삭제 연산을 최대한 감소시킬 수 있는 방법을 고려해야 한다.

Hot data는 빈번하게 접근 되는(access) 데이타로서 그 크기가 매우 작은 데이타이다. 이 Hot data는 잦은 겹쳐쓰기를 유발시키기 때문에 별도의 특별한 처리가 없다면 플래시 메모리의 성능을 대폭 감소시키는 요인이 됨에도 불구하고, 기존의 플래시 전환 계층들은 Hot data에 대한 별다른 처리를 해주고 있지 않다. 따라서 본 논문에서는 Cold data와 Hot data를 구분하여 저장하고, 사상테이블의 크기 문제로 실제 시스템에서 적용하기 어려웠던 이상적인 형태의 주소사상 기법인 섹터사상 기법을 크기가 매우 작은 Hot data에 적용하고, Cold data는 블록사상기법을 적용하는 새로운 적응형 사상기법을 통하여 쓰기와 삭제 연산을 최소화한 효율적인 플래시 전환 계층을 설계하여 제안한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 기존에 존재하는 플래시 전환 계층과 Hot data 검출에 대해서 알아보고, 기존의 시스템의 문제점을 살펴본다. 제 3장에서는 제안하는 AFTL의 핵심 아이디어와 시스템의 구조에 대해서 자세히 기술하고, 제 4장에서 FAST (Fully Associative Sector Translation layer)[6]와의 성능을 비교 분석하고, 마지막으로, 제 5장에서 결론과 함께 향후 연구 방향에 대하여 기술한다.

2. 관련 연구

Takayuki Sinohara[4]는 하나의 블록을 데이타 저장을 위한 공간과 대체를 위한 공간(replacement area)으로 나누는 방식의 블록 사상기법을 제안하였다. 그러나 이 방법은 더 이상의 업데이트가 없을 경우 공간낭비를 발생하고, 반복되는 쓰기패턴에 쉽게 Copy-back 연산을 유발하게 된다.

Amir Ban[5]은 각각의 논리적 블록당 물리적 블록의 비를 1:2 (FMAX) 혹은 1:N (ANAND)으로 하는 블록 사상기법을 제안하였다. 이는 삭제연산의 횟수를 줄여줄 수는 있으나, 많은 공간낭비를 발생시킨다.

Jesung Kim[2]은 일종의 쓰기연산을 위한 캐시(cache) 역할을 하는 로그(log) 블록을 두는 형태의 플래시 전환 계층을 설계하였다. 하지만 이는 임의쓰기 요구 발생시에 매우 낮은 성능을 가진다.

Sang-Won Lee[6]는 임의쓰기요구를 위한 로그 블록과 순차적인(sequential) 쓰기요구를 위한 로그 블록을 구분하는 형태의 플래시 전환 계층을 제시한다. 이는 뛰어난 성능을 보여주고 있지만 Cold data와 Hot data를 같은 로그블록에 위치시킴으로써 불필요한 Copy-back 연산을 발생시켜 성능저하를 일으키고 플래시 메모리 수명 감소를 유발한다.

Hot data는 빈번하게 접근되는 작은 크기의 데이타로서 Chang, L.P.[8]에 따르면 데이타 접근의 지역성(locality)이 종종 많은 응용환경(application)에서 발견된다. 일반적인 환경에서의 데이타 입출력을 분석해 보면, 3분의 2 이상의 쓰기 연산이 8섹터 미만의 크기를 가지며, 이 쓰기 연산들은 전체 데이타 크기의 10% 수준이고, 단지 1% 수준의 주소공간(address space)을 사용할 뿐이다. 즉, 극히 일부분의 주소공간이 자주 참조되며, 작은 량의 데이타가 빈번하게 접근되고 쓰여진다.

이런 Hot data를 검출하기 위하여 Jen-Wei Hsieh[9]은 K개의 독립적인 해시함수를 사용하는 Multihash-Framework를 제안하였다. 그러나, Hot data를 효과적으로 검출하는 알고리즘이 개발되었음에도 불구하고, 이를 효과적으로 활용한 플래시 전환 계층은 존재하지 않는다. 본 논문에서는 기존의 플래시 전환 계층이 가지고 있던 문제점인 잦은 Copy-back연산의 발생과 겹쳐쓰기 요구 발생시의 성능 감소 현상을 Hot data에 대한 고려를 통해서 해결 하는 방법을 제안한다.

3. 시스템 설계 및 구현

3.1 주요 아이디어

제안하는 시스템의 전체적인 구조는 그림 1과 같다. AFTL은 크게 Hot data 검사기(checker)와 Hot data 검출기, 섹터사상과 블록사상으로 구성 되어있다. 본 시스템에서는 Hot data 검출기를 통하여 검출된 적은 양의 Hot data에는 가장 이상적인 형태의 사상 기법인 섹터 사상기법을 적용하고, 그 밖의 데이타에는 로그 기반의 블록 사상 기법을 적용하는 적응형 방법을 사용한다. Hot data 검사기에서는 입력된 값이 각각 기존에 Hot 섹터와 Hot 블록으로 판별되었는지 여부를 확인하기 위해서, Hot 섹터목록과, Hot 블록 목록과 비교하게 된다. 제안된 연구에의 Hot data 검출은 Multi-hash Framework의 개념에 기초하고 있다. 하지만, 기존의 Hot data 검출기들이 단순히 Hot 블록만 검출하던 것과는 달리 본 연구에서는 Hot 블록 내에도 지역성이 존재한다는 판단과 섹터사상에의 사용을 위해서 2단계 Hot data 검출기를 사용하여 Hot 섹터와 Hot 블록의 여부를 결정하게 된다. 블록사상 단계에서는 로그 기반 블록 사상 기법으로 FAST기법을 사용하였다. 섹터사상 단계에서 시스템은 이 단계로 넘어오는 모든 논리적 섹터주소(LSN)와 그에 대한 물리적 섹터주소(PSN)를 LSN-to-PSN 사상테이블에 유지시켜 준다. 섹터사상 단계의 입력은 오직 Hot data로 검출된 데이타들로서, 관련연구에서 언급했듯이 이 양은 매우 적기 때문에, 사상테이블의 크기 때문에 사용하기 어려웠던 섹터사상 기법을 효과적으로 적용할 수 있는 것이다.

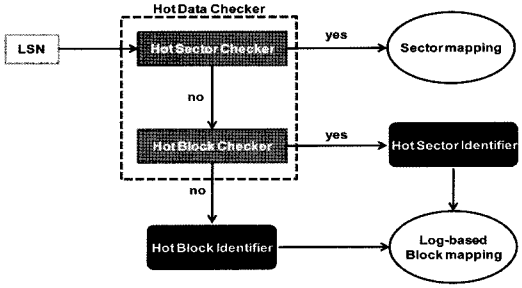


그림 1 AFTL의 프레임워크

3.2 알고리즘

본 장에서는 AFTL의 알고리즘에 대해서 서술한다.

알고리즘 1,2,3의 이해는 매우 직관적이다. 알고리즘 1은 AFTL이 어떤 방식으로 구현되는가를 보여준다. 만약, 입력된 LSN이 이미 *hot_sector_list*에 이미 존재한다면, 이상적인 성능을 지닌 섹터사상을 수행하게 된다. 만약에 *hot_sector_list*에 존재하지 않는다면, 논리적 블록주소(LBN)을 계산하여, *hot_block_list*와 비교하여, hot sector로 인식되지는 않았지만, 기존에 hot block으로 검출된 바가 있는지 여부를 확인하게 된다. 만약 hot block으로 검출된 데이타가 입력된 경우라면, 블록내의

hot sector를 검출하기 위해서 알고리즘 3 `hot_sector_identifier()`를 호출한다. 알고리즘 3은 reckon값을 반환하게 되는데, 이 reckon 값은 임계값을 넘어서는 카운터의 수를 의미한다. 이 후 반환된 reckon 값이 해시함수의 수인 k와 같다면, hot 섹터로 검출된 경우이기 때문에, *hot_sector_list*에 해당 lsn을 기록한 후 로그기반 블록사상을 수행한다. 해당 데이타는 추후에 재입력이 있을 시에는 알고리즘 1에 의해서 섹터사상 기법이 수행되게 된다. 한편, 입력된 데이타가 *hot_sector_list*와 *hot_block_list* 양쪽에 모두 존재하지 않는 경우에는 hot block 인지 여부를 확인하기 위해서 알고리즘2의 `hot_block_identifier()`를 호출하게 된다. 알고리즘 2도 reckon 값을 반환하게 되는데 동작은 알고리즘 3의 경우와 동일하며, 다만 여기서는 hot block 여부를 결정 짓기 때문에 reckon 값이 k와 같다면 *hot_block_list*에 LBN을 저장하고 로그기반 블록사상을 실시하게 된다. 알고리즘 2와 3은 해당 논리 주소에 해당하는 쓰기요구가 얼마나 많이 들어왔는지를 판별하기 위한 것으로서 다음과 같이 수행된다. 첫째, 해당 논리 주소의 값을 K개의 독립적인 해시함수에 의해서 해싱하고, 둘째, 해싱값에 따른 카운터를 증가시켜주고, 마지막으로 해당된 해시 값의 카운터가 임계값보다 크다면, reckon 값을 1씩 증가시켜준다.

Algorithm 1 `AFTL` (*lsn*, *data*)
Input: lsn, data
Output: none
1 *lbn* := *lsn* div SectorsPerBlock
2 *k* := the number of hash functions
3 **if** compare *lsn* to *hot_sector_list*
4 sector mapping
5 **else if** compare *lbn* to *hot_block_list*
6 call *hot_sector_identifer* (*lsn*)
7 **if** *reckon* is *k*
8 store *lsn* to *hot_sector_list*
9 log based block mapping
10 **else** call *hot_block_identifier* (*lsn*, *data*)
11 **if** *reckon* is *k*
12 store *lbn* to *hot_block_list*
13 log based block mapping
14 **end**

Algorithm 2 `hot_block_identifier` (*lsn*)
Input: lsn
Output: reckon
1 *lbn* := *lsn* div SectorsPerBlock
2 *k* := the number of hash functions
3 *threshold* := threshold value for hot data identification
4 **for** *i* =1 to *k*
5 *entry* = hashing[*i*] (*lbn*)
6 increase *hashtable*[*entry*]
7 **if** *hashtable*[*entry*] > *threshold*
8 increase *reckon* by 1
9 **end**

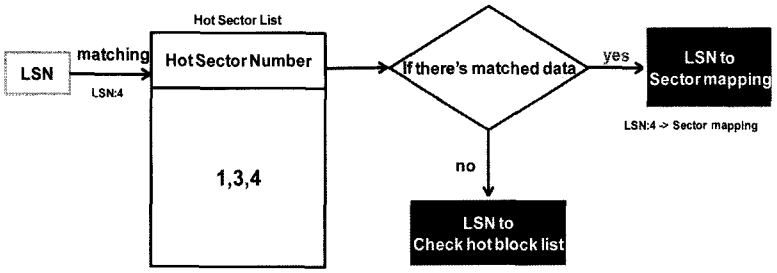
Algorithm 3 `hot_sector_identifier` (*lsn*, *data*)
Input: lsn
Output: reckon
1 *k* := the number of hash functions
2 *threshold* := threshold value for hot data identification
3 **for** *i* =1 to *k*
4 *entry* = hashing[*i*] (*lsn*)
5 increase *hashtable*[*entry*]
6 **if** *hashtable*[*entry*] > *threshold*
7 increase *reckon* by 1
8 **end**

3.3 Hot data 검사기

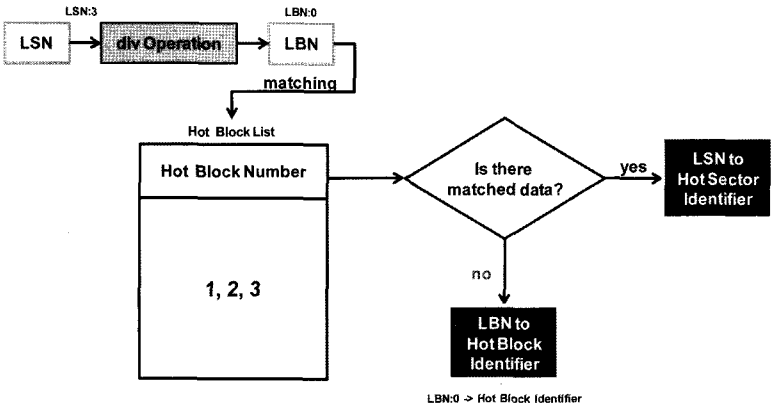
최초의 쓰기연산 요구는 Hot 섹터 목록 검사기에 들어와 이미 기존에 Hot 섹터로 분류되었는지 여부를 판별하기 위해서 Hot 섹터 목록과 비교하게 된다. 검사과정을 통해서 일치되는 데이타가 있을 경우에는 LSN을 섹터사상 단계로 넘기게 되고 일치되는 데이타가 없을 경우에는 LSN을 Hot 블록 목록 검사기로 넘긴다. 그림 2의 (a)에서는 LSN 4가 들어와 기존의 Hot 섹터 목록과 비교한 결과 일치하는 정보를 확인하여 LSN 4를 섹터사상 단계로 넘겨주었다. Hot 블록 목록 검사기로 LSN이 넘어 오게 되면 일단 나누기 연산을 통해서 해당 논리섹터주소에 해당되는 논리블록주소(LBN)를 산출한다. 산출된 LBN을 Hot 블록 목록과 비교하여 일치하는 데이타가 있을 경우는 LSN을 Hot 섹터 검출기로 보내고, 그렇지 않은 경우에는 LBN을 Hot 블록 검출기로 보낸다. 그림 2의 (b)에서는 LBN 3과 일치되는 데이타가 없었기 때문에 LBN을 Hot 블록 검출기로 보내는 것을 볼 수 있다.

3.4 Hot data 검출기

Hot Data를 추출하기 위해서 Multihash-Framework의 기본 방식은 유지하되, 기존의 Hot Data 검출기들이 Hot 블록만을 추출하여 Hot Data로 정의하였던 것에 비해서, 본 논문에서 제안하는 시스템은 Hot 블록 내에도 지역성이 존재한다는 판단 하에 Hot 섹터를 추출하기 위해서 2번의 검출과정을 거치게 된다. Hot 블록 검출기로 들어온 LBN은 Hot 블록 검출을 위한 Multi-hash 함수로 들어가게 되고 각각의 해싱된 값에 따라서 카운트를 증가시키게 된다. 그림 3에서는 총 4개의 해시함수와 4bit로 이루어진 카운터로 구성되고, 해시 함수가 가리키는 4곳의 카운터의 상위 2bit에 해당되는 MSB(Most Significant Bits)가 모두 00이 아닐 시에는



(a) Hot 섹터 목록 검사기



(b) Hot 블록 목록 검사기

그림 2 Hot 섹터 목록 검사기와 Hot 블록 목록 검사기

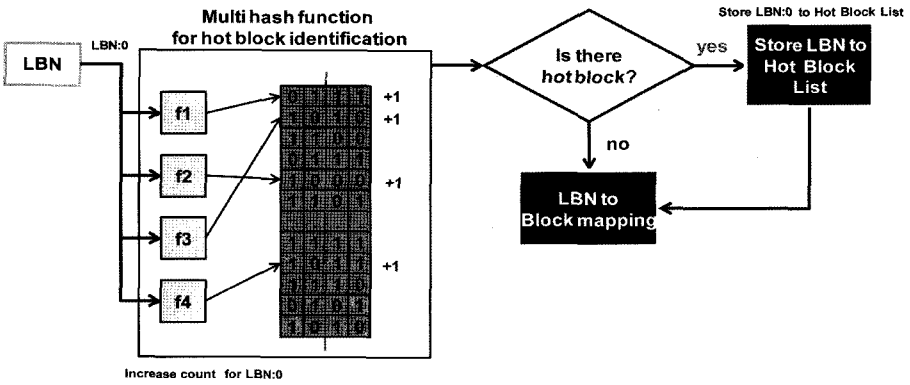


그림 3 Hot 블록 검출기

충분한 입력이 있었던 것으로 판단하여 Hot 블록으로 인식하게 되고 인식된 Hot 블록의 LBN은 Hot 블록 목록에 저장되게 된다. 그림 3에서는 LBN 0이 들어와 Hot 블록 목록에 저장되었다. Hot 섹터 검출기로 넘어올 수 있는 LSN은 이미 Hot 블록으로 검출되어 Hot 블록 목록에 저장된 블록 안에 존재하는 섹터이다. Hot 섹터 검출기는 Hot 블록 검출기와 동일한 방식으로 검출과정이 이루어지고, 검출된 Hot 섹터는 바로 섹터 사상기법을 적용하여 사상시키는 것이 아니라, LSN을 Hot 섹터 목록에 저장하고 추후에 다시 해당 LSN에 쓰기요구가 들어왔을 때 섹터사상기법이 적용되게 된다. 그림 4에서는 LSN 33이 들어와 Hot 섹터로 검출된 이후에 Hot 섹터 목록에 자신의 LSN을 저장하고 LBN으로 변환되어 블록사상 단계로 자신의 LBN과 LSN을

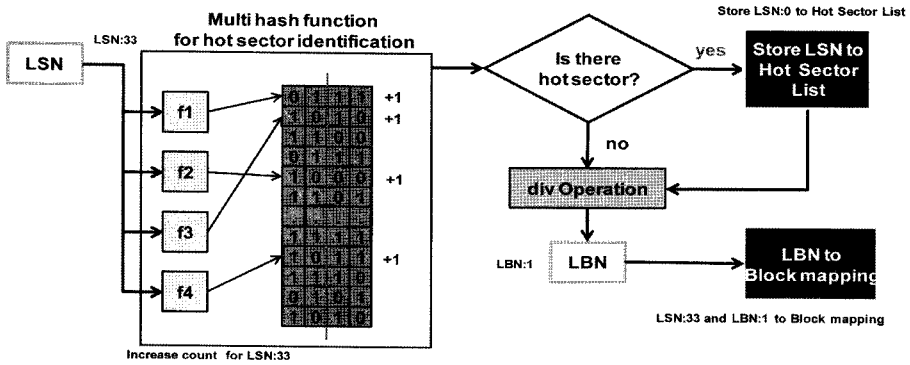


그림 4 Hot 섹터 검출기

넘겨 주게 된다.

3.5 블록사상

블록사상단계에서는 기본적으로 FAST의 블록사상 기법을 이용한다. 입력 받은 LBN을 LBN-to-PBN 사상테이블의 정보와 비교하여 최초로 들어온 경우에는 프리블록풀(Free Block Pool) 을 통해서 새로운 블록을 할당하고 해당 오프셋의 위치에 데이타를 적어주는 기본 블록쓰기 연산을 수행한다. 만약, 기존에 이미 할당된 블록이 있다면 해당 섹터가 비어있는지를 확인한다. 만약 비어있다면 기본 블록쓰기 연산을 수행하고, 그렇지 않다면, 로그블록 테이블을 확인하여 할당된 로그블록이 있는지 여부를 살핀다. 만약, 할당된 로그블록이 없다면, 새로운 로그블록을 프리블록풀을 통해서 할당하고 빈 공간에 순차적으로 적어 준 후 로그블록을 위한 LSN-to-PSN 사상테이블을 업데이트해주는 로그 쓰기 연산을 수행한다. 기존에 이미 할당된 로그블록이 있다면, 로그블록에 빈 공간이 있는지 여부를 살핀다. 만약 로그 블록에 빈 공간이 있다면 로그 쓰기연산을 수행하고, 그렇지 않다면, 합병 연산을 수행한다. 합병연산은 기존의 데이타 블록의 유효한 데이타와 해당 블록에 대한 로그블록상의 유효한 데이타를 새로운 데이타 블록에 옮겨 적고, 로그블록을 위한 LSN-to-PSN 사상테이블을 업데이트해주는 것을 의미한다. 그림 5에서는 LSN 33이 들어와 기존에 할당되어 있던 물리 블록 주소를 확인한 후에 기본 쓰기연산을 수행하는 과정을 보여준다.

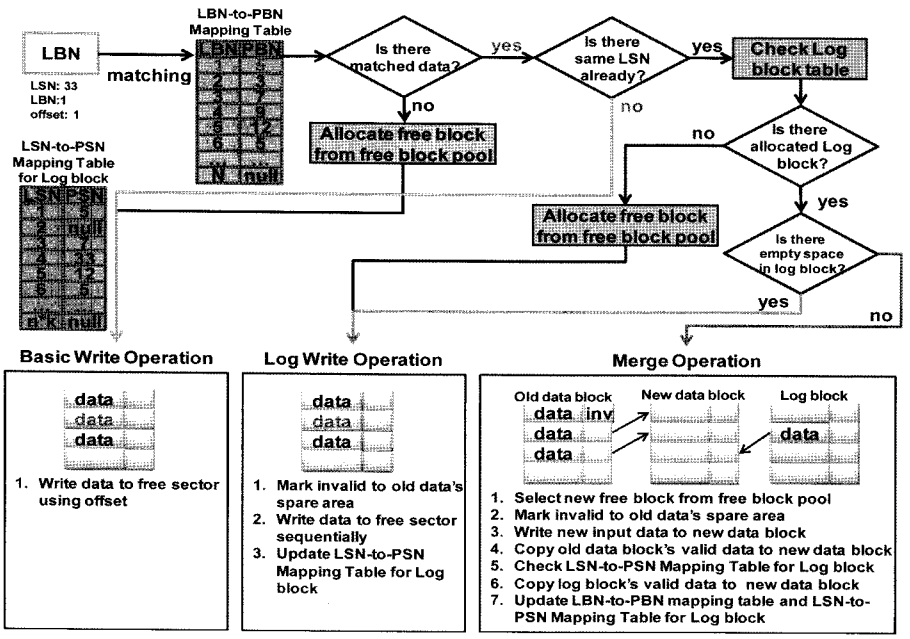


그림 5 블록사상기법

3.6 섹터사상

이미 Hot 섹터로 인식된 섹터들이 입력으로 들어오게 되는 섹터 사상 단계에서는 해당 LSN을 LSN-to-PSN 사상테이블과 비교한 후 일치하는 값이 있는 지 여부를 살핀다. 일치하는 값이 없다면 새롭게 Hot섹터로 분류된 데이타이기 때문에, 새롭게 블록을 할당해 주어야 한다. 할당은 기존에 섹터사상을 위해 사용되고 있던 블록에서 가장 빈 공간이 많은 곳을 우선하여 할당하며, 기존에 섹터사상을 위해서 사용되던 블록들에 빈 공간이 존재 하지 않을 경우에는 새로운 블록을 할당하여 준다. 할당이 끝나면 빈 섹터에 순차적으로 데이타를 적은 후 LSN-to-PSN 사상테이블을 업데이트 하는 기본 섹터 쓰기연산을 수행한다. 만약에, 기존에 이미 쓰기 연산이 실행된 바가 있는 데이타라면, 해당 블록에 빈 공간이 있는지 여부를 살피고, 빈 공간이 있다면 해당 블록의 오래된 데이타 섹터를 무효화 처리한 후 빈 섹터에 쓰기 연산을 수행하고 LSN-to-PSN 사상테이블을 업데이트 하는 겹쳐쓰기연산을 실행한다. 만약, 해당블록에 더 이상의 빈 공간이 없는 경우라면 새로운 블록을 프리블록풀로부터 할당 받고, 데이타를 쓴 후 LSN-to-PSN 사상테이블을 업데이트 하는 할당 연산을 수행한다. 그림 6에서는 LSN 4가 들어와서 기존에 물리섹터 주소 33에 해당되는 데이타가 있었기 때문에 기존의 주소를 무효화 처리하고 빈 공간에 새로운 데이타를 적어주는 겹쳐쓰기연산이 수행되는 과정을 보여준다.

3.7 적응형 플래시 전환 계층의 동작 예

본 장에서는 AFTL의 동작을 예를 통하여 살펴보고자 한다. 본 예에서는 블록당 섹터의 수를 4로 가정하였고, 1개의 로그 쓰기용 로그블록과 1개의 Hot data를 위한 Hot data 블록을 두었다.

먼저, 겹쳐쓰기를 발생시킨 데이타가 Cold data라면, 해당 데이타는 로그블록에 삽입된다. 그림 7의 case 1이 바로 이 경우에 해당하며, 섹터 1이 로그블록에 삽입되었다. 만약 겹쳐쓰기를 발생한 데이타가 Hot 블록 목록에 있는 경우라면 AFTL은 Hot 섹터 검출기를 호출하게 된다. 그림 상의 case 2, 3이 이에 해당하며, AFTL은 Hot 섹터 검출기를 불러와 섹터 4와 섹터 5에 해당하는 카운터를 1씩 증가 시키게 된다. 본 예에서는 섹터 4의 경우에 앞서 3.2절에서 언급한 바 있는 reckon이 임계값을 넘어서 데이타는 로그블록에 저장되고, Hot 섹터 목록에 4를 추가시키는 상황을 보여주고 있다. 이에 반해 case 3의 경우는 아직 reckon 값이 임계치를 넘어서지 않아서 단순히 로그블록에 데이타를 기록하는 과정만을 거친다. 마지막으로, 만약 입력된 데이타가 기존에 이미 Hot 섹터로 검출된 경우에는, AFTL은 해당 데이타를 Hot data 블록에 쓴다. 그림 상의 case 4와 5의 섹터 4와 섹터 8의 경우가 이에 해당한다. 섹터 4는 로그 블록에 쓰여져 있기 때문에 AFTL은 로그블록 내의 데이타에 유효하지 않은 데이타라고(invalid data) 체크를 한 후에 Hot data 블록에 섹터 4의 데이타를 적는다. Case 5의 섹터 8의 경우는 이미 hot 섹터로 분류되어 Hot 섹터 목록에 섹터 8의 정보가 들어 있지만, 로그 블록 내에는 데이타가 존재 하지 않는다. 이는 로그 블록의 고갈 등으로 합병(merge) 연산이 발생한 이

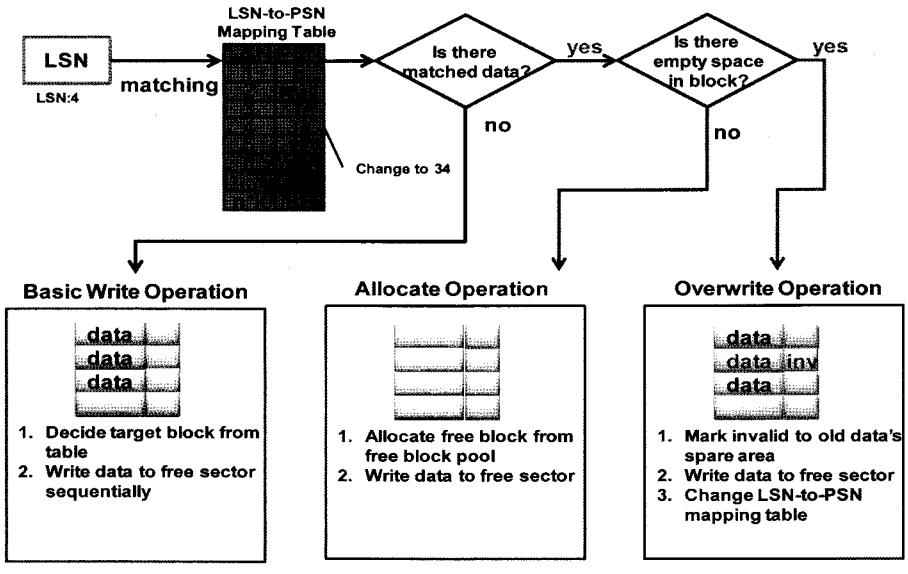


그림 6 섹터사상기법

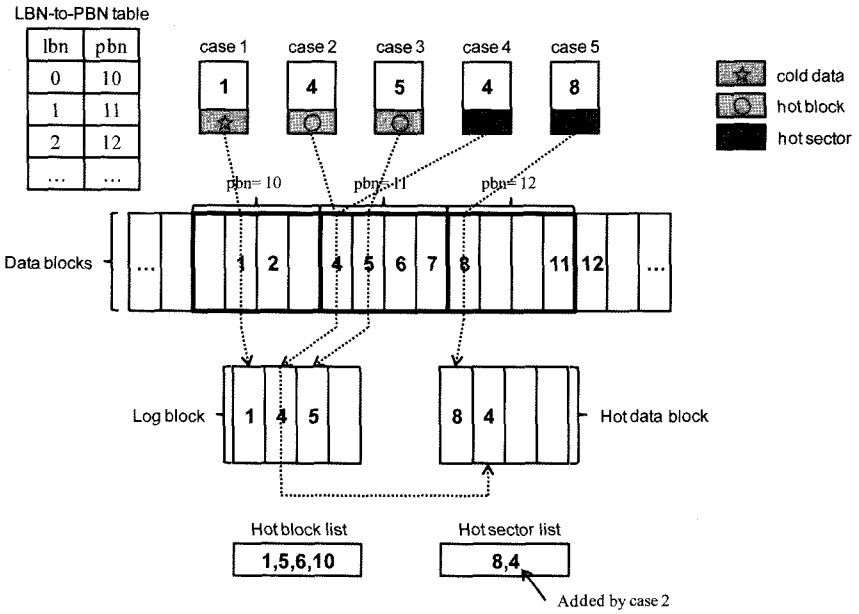


그림 7 AFTL의 동작 예

후의 최초 입력인 경우이다. 따라서 AFTL은 섹터 8의 데이타를 Hot data 블록에 바로 적어 준다.

4. 성능평가 및 분석

제안된 시스템을 평가하기 위하여 FAST기법과 비교하였다. AFTL과 FAST 각각의 기법을 시뮬레이터로 구현하여 다양한 트레이스(trace) 파일들을 적용함으로서 성능을 평가하였다. 각각의 트레이스 파일 마다 읽기 연산이 수행된 횟수, 쓰기연산이 수행된 횟수, 삭제연산이 수행된 횟수와 함께 각각의 연산에 필요한 시간을 계산적으로 구해낸 총 소모시간을 산출하여 비교하였으며, 이는 제안된 시스템의 성능을 충분히 보여준다. 사용된 트레이스 파일은 BAST(Block Associative Sector Translation layer)[2]와 FAST에서 성능평가를 위하여 사용되었던 파일로써, 각 Linux, Symbian OS, 2종의 디지털 카메라를 통해서 수집되었다.

그림 8과 표 1을 통해서 살펴 볼 수 있듯이 각각의 트레이스 파일들은 서로 다른 섹터 범위와 쓰기횟수를

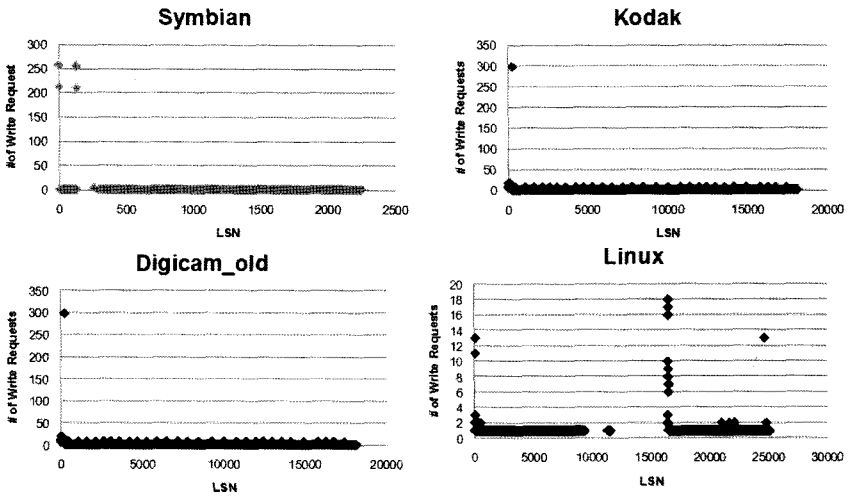


그림 8 트레이스 파일 별 쓰기 상황에서의 지역성 분석

표 1 실험에 사용된 트레이스 파일 분석

Description	Workload from digital camera (A company)	Workload from digital camera (B company)	Workload from Linux O/S	Workload from Symbian O/S
# of writes	2,199,200	3,144,800	1,890,000	404,900
Range of LSN	0~18107	0~27758	0~25101	0~2242
# of Hot Sector	162	176	42	9
Ratio of Hot sector	0.89%	0.63%	0.16%	0.4%
Writes by Hot sector	250300	238700	54200	196900
Ratio of Hot sector write	11.4%	7.6%	2.8%	48.7%

가지고 있으며 Hot data의 비율과 Hot data가 일으킨 쓰기연산의 횟수 또한 2.8%~54%로 다양한 범위를 지니고 있다. Symbian OS의 트레이스 데이타가 가장 지역성이 높은 쓰기연산을 수행하고 있고, Linux를 통해 얻은 트레이스 데이타가 가장 균등한 지역성을 지닌 쓰기연산을 수행하고 있는 것을 표를 통해서 알 수 있다. 또한, 다양한 환경에서 얻어낸 트레이스 파일들임에도 불구하고 분명히 Hot data가 존재하는 것을 그림 8를 통해서 알 수 있다. 이상의 트레이스 파일을 통해서 시뮬레이션을 수행한 결과 그림 9와 표 2의 결과를 얻어낼 수 있었다. 읽기, 쓰기, 삭제 연산의 그래프의 단위는 횟수이며, 소모시간 그래프에서의 단위는 1초이다.

표 2를 통해 보여지듯이, A사의 디지털 카메라 소모시간 트레이스에서는 읽기 연산 49.3%, 쓰기연산 19.7%, 삭제연산14.7% 감소의 결과를 보였고, 이를 토대로 산출한 소모시간은 17.9% 향상된 결과를 보인다. B사의 디지털 카메라 트레이스에서는 읽기연산 46.0%, 쓰기연산 13.8%, 삭제 연산 11.3%의 감소를 가져왔고, 소모시간은 14.5% 향상되었다. Linux 트레이스에서는 읽기연산 23.2%, 쓰기연산 2.8%, 삭제연산 1.6%의 감소를 가져왔고, 이는 총 소모시간 3.04%의 성능 향상으로 수행시간이 22초 감소한 것을 의미한다. Symbian 트레이스에서는 읽기연산 86.2%, 쓰기연산 17.0%의 성능향상을 가져왔으나, 삭제연산은 5.3% 증가하였다. 이를 총 소모시간으로 산출하면, 14.5%의 성능 향상을 가져 온 것을 확인 할 수 있다.

예상한 바와 같이 데이타의 지역성이 높을수록 더 많은 성능향상을 가져 오게 되는데, 표 2와 그림 9를 통해서 알 수 있듯이 우리가 제안하는 시스템은 삭제연산의 횟수를 대폭 감소시키는 것을 볼 수 있다. 다만 섹터사상을 위해 고정적인 크기의 공간을 이용하였기 때문에 Hot data에 의한 쓰기연산이 매우 높은 비중을 차지하는 Symbian 트레이스에서는 저장장소의 빠른 고갈로 인해 삭제연산이 늘어 난 것을 확인 할 수 있다. 한 가지 주목할 만한 점은 읽기와 쓰기연산의 횟수가 비약적으로 감소하였다는 점이다. 이는 섹터 사상단계로 들어가는 입력 값이 많아 졌다는 것을 의미하는데, 즉, 기존의 플래시 전환 계층들보다 효과적으로 섹터사상기법이

표 2 트레이스 파일 별 성능향상 비교

Digital Cam. A

	FAST	AFTL	Performance Enhancement (%)
READ	1505433	762724	49.33
WRITE	3153149	2627133	19.69
ERASE	110973	94615	14.74
Elapsed Time	1114.879	915.5054	17.88

Digital Cam. B

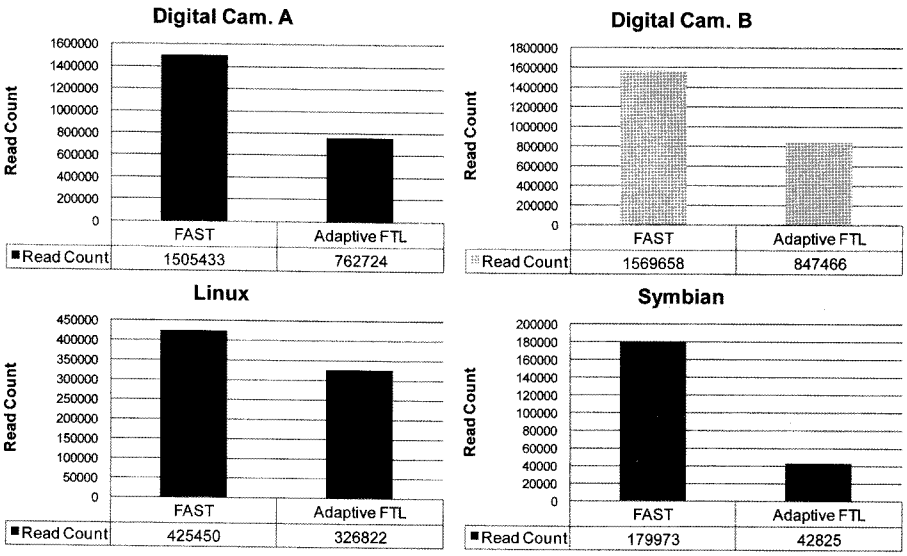
	FAST	AFTL	Performance Enhancement (%)
READ	1569658	847466	46.01
WRITE	4206127	3626250	13.79
ERASE	142945	126789	11.31
Elapsed Time	1461.277	1248.669	14.55

Linux

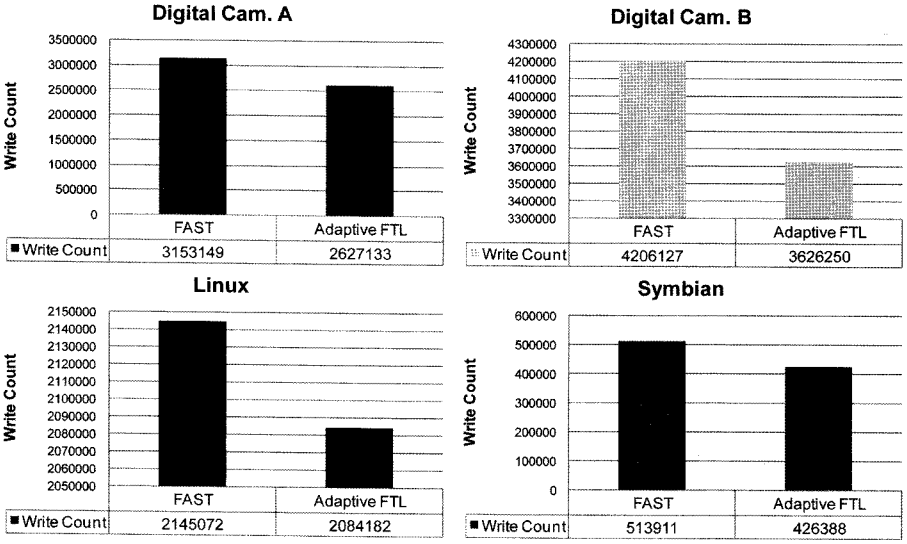
	FAST	AFTL	Performance Enhancement (%)
READ	425450	326822	23.18
WRITE	2145072	2084182	2.84
ERASE	70243	69095	1.64
Elapsed Time	726.3914	704.3480	3.04

Symbian

	FAST	AFTL	Performance Enhancement (%)
READ	179973	42825	86.21
WRITE	513911	426388	17.03
ERASE	18708	19704	-5.32
Elapsed Time	180.5954	154.3689	14.53



(a) 읽기 연산 비교 (단위: 횟수)



(b) 쓰기 연산 비교 (단위: 횟수)

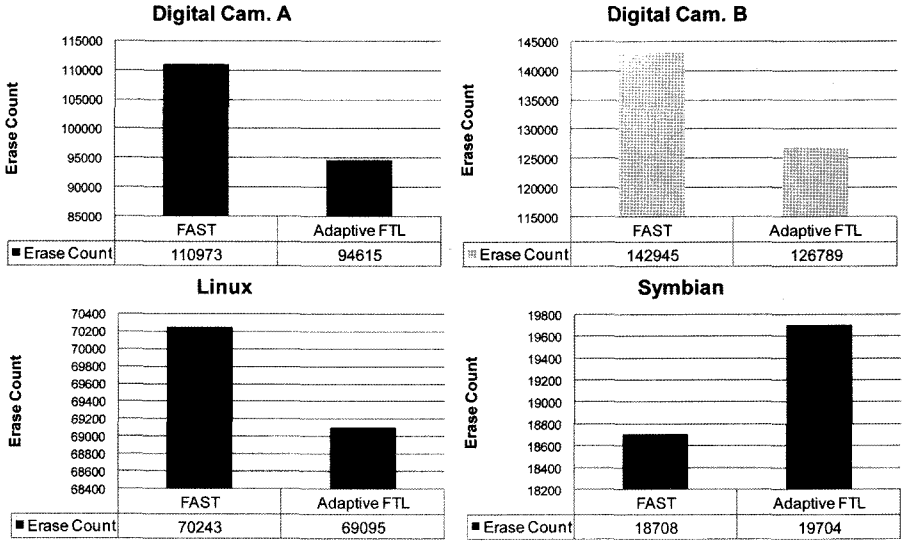
그림 9 성능 평가

사용되었다는 것을 의미하는 것이다.

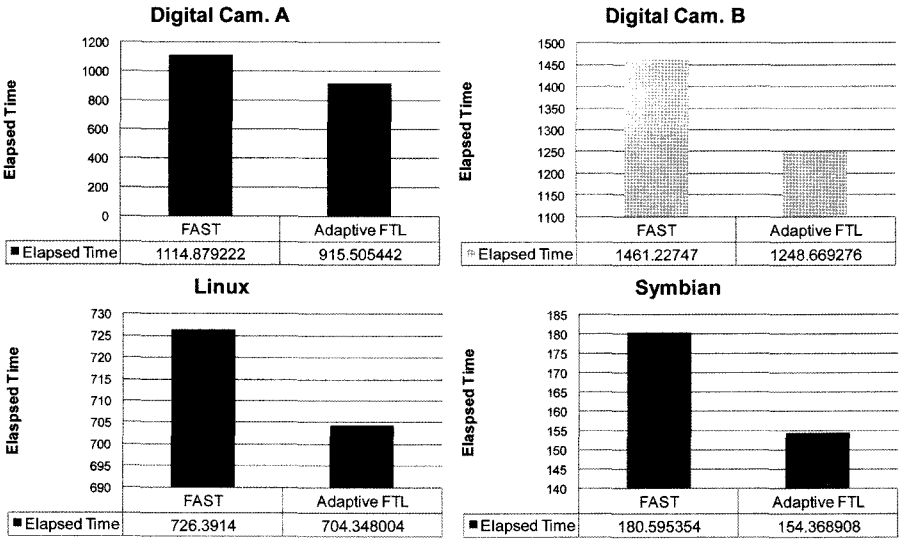
5. 결 론

본 논문에서 우리는 기존에 이미 우수한 성능으로 알려져 있던 FAST기법 보다 뛰어난 성능의 플래시 전환 계층인 AFTL을 제안하였다. 이러한 성능향상을 가져올 수 있었던 이유는 Hot data를 고려한 효과적인 적응형 사상기법을 이용하였기 때문이다. AFTL은 Hot data와 Cold data를 구분하여 저장함으로써 불필요하게 발생되었던 Copy-back 연산과 삭제연산을 대폭 줄여준다. 또한, AFTL은 속도 면에서 우수한 성능을 보이지만 사상테이블의 크기 문제 때문에 실질적인 시스템에 적용하기 어려웠던 섹터사상을 최대한 활용할 수 있는 적응형의 사상기법을 통하여, 기존의 연구들 보다 넓은 범위의 입력 값을 보다 적은 크기의 사상정보를 이용하여 섹터 사상을 수행함으로써, 읽기와 쓰기연산 또한 대폭 감소 시켜주는 성능을 보여준다.

향후, 제안된 시스템의 신뢰성을 높이기 위해서 전원



(c) 삭제 연산 비교 (단위: 횟수)



(d) 총 소모시간 비교 (단위: 초)

그림 9 성능 평가 (계속)

오류 등의 상황에서의 효율적인 대처방안을 고려하고, Symbian 트레이스에서의 삭제연산의 증가의 원인이 되었던, 고정적인 섹터사상영역이 가져오는 문제점을 해결하게 위해서 동적인 공간할당을 위한 연구를 수행하고자 한다.

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] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min and Yookun Cho, "A Space-Efficient Flash Translation Layer for CompactFlash Systems," IEEE Transactions on Consumer Electronics, Vol.48, No.2, May 2002.

[3] 이현섭, 강원식, 이동하, 이동호, "플래시 메모리상에 B+트리를 위한 효율적인 색인 버퍼 관리정책", 한국정보과학회 학술발표논문집, Vol.33, No.2(C), 2006.

[4] Takayuki Shinohara, "Flash Memory Card with Block Memory Address Arrangement," United States Patent, no. 5,905,993, 1999.
[5] Amir Ban, Rama hasharon, Israel, "Flash File System Optimized for Page-Mode Flash Technologies," United States Patent, no. 5,937,425, 1999.
[6] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sang-Won Park and Ha-Joo Song, "FAST: A Log Buffer based Flash Translation Layer using Fully Associative Sector Translation," The 2005 US-Korea Conference on Science, Technology, & Entrepreneurship, 2005.
[7] Tae-Sun Chung, Dong-Joo Park, Sangwon Park, Dong-Ho Lee, Sang-Won Lee, Ha-Joo Song, "System Software for Flash Memory: A Survey," The 2006 IFIP International Conference on Embedded and Ubiquitous Computing, Lecture Note in Computer Science (LNCS), Vol.4096, pp. 394-404, Springer-Verlag, 2006.
[8] Chang, L. P. and Kuo, T.W, "An Efficient Management Scheme for Large-Scale Flash Memory Storage Systems," In Proceedings of the ACM Symposium on Applied Computing, 2004.
[9] Jen-Wei Hsieh and Tei-Wei Kuo, "Efficient identification of Hot Data for Flash Memory Storage Systems," ACM Transactions on Storage, Vol.2, No.1, pp. 22-40, 2006.

윤 현 식

2006년 한양대학교 컴퓨터공학과 졸업(학사). 2007년 현재 한양대학교 컴퓨터공학과(석사). 관심분야는 데이타베이스, 플래시메모리용 시스템소프트웨어 등

주 영 도

1983년 한양대학교 전자통신공학과 졸업(학사). 1988년 미국 University of South Florida 컴퓨터공학과 졸업(석사). 1995년 미국 Florida State University 컴퓨터과학과 졸업(박사). 1984년~1985년 한국무역협회 전자계산소 시스템 프로그래머. 1995년~2000년 KT 연구개발본부 연구팀/실장. 2000년~2005년 시스코 시스템즈 코리아 통신사업부 담당 상무. 2005년~2006년 화웨이 기술 코리아 부사장. 2007년~현재 강남대학교 컴퓨터미디어공학부 교수. 관심분야는 데이타베이스, 지능형시스템, 초고속통신망, 통신망관리 등

이 동 호

1995년 홍익대학교 컴퓨터공학과 졸업(학사). 1997년 서울대학교 컴퓨터공학과 졸업(석사). 2001년 서울대학교 컴퓨터공학과 졸업(박사). 2001년~2004년 삼성전자 책임 연구원. 2004년~현재 한양대학교 컴퓨터공학과 조교수. 관심분야는 데이타베이스, 멀티미디어 정보검색, 플래시메모리용 시스템소프트웨어 등