NAND 플래시 기반 모바일 저장장치를 위한 사상 테이블 캐싱 기법

A Mapping Table Caching Scheme for NAND Flash-based Mobile Storage Devices

양수현(Soo-Hyeon Yang)*, 류연승(Yeonseung Ryu)**

초 록

최근 모바일 컴퓨터를 사용한 온라인 금융 거래, 온라인 쇼핑과 같은 e-비즈니스가 널리 확산되고 있다. 대부분의 모바일 컴퓨터는 데이터 저장을 위해 NAND 플래시 메모리 기반의 저장장치를 사용한다. 플래시 메모리 저장장치는 그 내부에 Flash Translation Layer (FTL)이라는 소프트웨어가 사용되고 있다. FTL은 파일 시스템으로부터 요청되는 논리 주소를 플래시 메모리의 물리 주소로 변환하며 이를 위하여 사상 테이블을 사용한다. 기존 FTL은 매우 큰 주소 사상 테이블을 RAM에 유지해야 하는 문제점을 가지고 있다. 이를 해결하기 위하여 본 논문에서는 새로운 사상 테이블의 캐싱 기법을 제안하였다. 트레이스 기반의 시뮬레이션을 통해 제안한 사상 테이블 캐싱 기법은 공간 비용을 대폭 줄이고 시간 비용은 크게 증가하지 않음을 알 수 있었다. 특히, e-비즈니스 환경의 온라인 트랜잭션 워크로드에서 많은 공간 비용 절감 효과를 보였다.

ABSTRACT

Recently e-business such as online financial trade and online shopping using mobile computes are widely spread. Most of mobile computers use NAND flash memory-based storage devices for storing data. Flash memory storage devices use a software called flash translation layer to translate logical address from a file system to physical address of flash memory by using mapping tables. The legacy FTLs have a problem that they must maintain very large mapping tables in the RAM. In order to address this issues, in this paper, we proposed a new caching scheme of mapping tables. We showed through the trace-driven simulations that the proposed caching scheme reduces the space overhead dramatically but does not increase the time overhead. In the case of online transaction workload in e-business environment, in particular, the proposed scheme manifests better performance in reducing the space overhead.

키워드 : 모바일 저장장치, 플래시 메모리, 사상 테이블, 캐싱
Mobile storage device, Flash memory, Mapping table, Caching

이 논문은 2008년 정부(교육과학기술부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임 (KRF-2008-314-D00344).

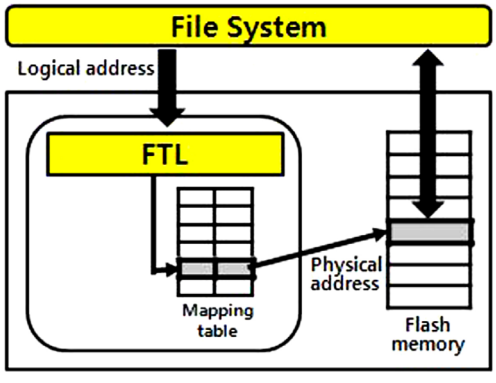
* 명지대학교 컴퓨터공학과
** 명지대학교 컴퓨터공학과

2010년 09월 27일 접수, 2010년 10월 16일 심사완료 후 2010년 11월 05일 게재확정.

1. 서 론

최근 스마트폰의 폭발적인 확산으로 인해 모바일 컴퓨터를 이용한 금융 거래, 온라인 쇼핑, 소셜 네트워킹 등의 e-비즈니스가 점차 확대될 전망이다. 모바일 컴퓨터의 저장장치는 일반적으로 NAND 플래시 메모리가 널리 사용되고 있는데, 이는 플래시 메모리가 하드 디스크와 비교하여 무게가 가볍고, 전력 소비가 적으며, 빠른 I/O가 가능하다는 장점을 가지고 있기 때문이다.

플래시 메모리 기반의 저장장치는 플래시 메모리를 하드디스크 처럼 사용하기 위해 Flash Translation Layer(FTL)이라는 특별한 소프트웨어를 사용한다(Intel[4])(Chung et al.[10])(Shin et al.[7]). FTL은 결함(bad) 블록 관리, 논리 주소의 물리 주소로의 변환, 에러 검사 등의 기능을 수행한다. <그림 1>은 플래시 메모리 저장장치의 내부 구조를 보여주고 있다. FTL은 파일 시스템으로부터 요청되는 논리 주소를 플래시 메모리의 물리 주소로 변환하고, 변환된 물리 주소에 데이터의 입출력을 수행한다. FTL은 논리 주소와 사상된 물리 블록의 정보를 관리하기 위해 주소 사상 테이블(address mapping table)을 사용한다(Gal et al.[3]). 최근 플래시 메모리가 대용량화되고 있으며 이에 따라 사상 테이블로 인해 요구되는 저장 공간의 크기도 커지는 문제점이 발생하고 있다. 특히, 사상 테이블의 빠른 접근을 위하여 FTL은 주소 사상 테이블을 RAM에 유지하는데 사상 테이블로 인해 요구되는 RAM의 크기도 커지는 문제점이 있다. 따라서, 주소 사상 테이블을 플래시 메모리에 저장하고 테이블의 필요한 부분만 RAM에 캐싱하여 사용하는 기법이 필요하다.



〈그림 1〉 플래시 메모리 저장장치의 내부 구조

본 논문에서는 FTL의 주소 사상 테이블을 필요할 때마다 생성하고 RAM에 적재하는 사상 테이블 캐싱 기법을 연구하였다. 사상 테이블의 캐싱은 사상 테이블 전부를 RAM에 유지할 필요가 없어 공간 비용을 줄이는 대신에 캐시 부재가 발생했을 때 이를 처리하기 위한 시간 비용이 증가한다. 트레이스 기반의 시뮬레이션을 통해 본 논문에서 제안한 사상 테이블 캐싱 기법은 공간 비용을 대폭 줄이고 시간 비용은 크게 증가하지 않음을 보였다. 특히, e-비즈니스의 온라인 트랜잭션 워크로드에서는 많은 공간 비용 절감 효과를 보였다. 또한, 다양한 워크로드에서 사상 테이블의 캐시 적중률이 매우 높아 캐시 부재로 인한 수행 비용은 매우 작았다.

본 논문의 구성은 다음과 같다. 먼저, 제 2장에서는 플래시 메모리의 특징과 FTL에 대해 소개하고, FTL의 주소 사상 변환 알고리

증인 SAT 기법을 소개한다. 제 3장에서는 제안하는 사상 테이블 캐싱 기법을 설명하고 제 4장에서 실험 결과를 기술한다. 제 5장에서 결론과 향후연구를 기술한다.

〈표 1〉 플래시 메모리의 특징

항목	SLC	MLC
페이지 크기	2K + 64bytes	4K + 128bytes
블록 크기	128K + 4K bytes	512K + 16K bytes
페이지 읽기 시간	25us	60us
페이지 쓰기 시간	200us	800us
블록 삭제 시간	1.5ms	1.5ms
블록 내구성	100K cycles	10K cycles

2. 관련 연구

2.1 플래시 메모리 특징

NAND 플래시 메모리는 SLC(single-level-cell)과 MLC(multi-level-cell)의 두 유형이 있다. 한 메모리 셀(cell)이 한 비트 값(0과 1)만 저장할 수 있다면 SLC 플래시라고 하며, 한 메모리 셀이 여러 비트 값을 저장할 수 있다면 MLC 플래시라고 한다. <표 1>은 SLC와 MLC 플래시의 특징을 보여주고 있다. MLC은 SLC보다 용량이 크고 가격이 싸기 때문에 대용량 저장장치에 쓰인다. 그러나, MLC은 SLC보다 접근 속도가 느린 단점이 있다.

플래시 메모리의 여러 셀이 모여 한 페이지(page)를 구성한다. 페이지는 2KB에서 4KB 크기의 데이터 영역과 64바이트에서 128바이트 크기를 가지는 여유(spare) 영역으로 이루어진다. 여유 영역은 페이지나 블록의 ECC (error correction code) 또는 메타 정보를 저장하기 위해 사용한다. 여러 개의 페이지가 모여 한 블록(block)을 구성한다. 보통 64개에서 128개의 페이지가 한 블록을 구성한다.

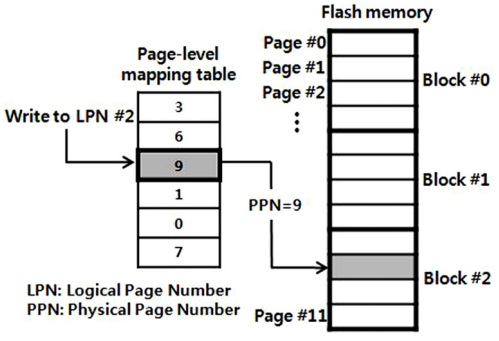
플래시 메모리는 기본적으로 읽기, 쓰기, 삭제의 세 가지 연산이 수행될 수 있다. 읽기와 쓰기의 단위는 페이지이고, 삭제의 단위는 블록이다. <표 1>에서 볼 수 있듯이 삭제 연산은 읽기 및 쓰기 연산보다 많이 느리다. 한 번 쓰여진 페이지는 삭제가 되어야 다시 데이터를 저장할 수 있다. 이런 특징을 erase-before-write 특징이라 부른다. 그런데, 삭제 연산은 블록에 대해 수행되므로 한 페이지만 변경하더라도 블록을 삭제해야 하므로 플래시 메모리에 대한 변경은 성능을 저하시키는 문제점이 있다. 또한 각 블록에 대해 수행할 수 있는 삭제 횟수가 제한된다. 일반적으로 10,000번 또는 100,000번까지 삭제할 수 있다. 만약 특정한 블록에 삭제 연산이 집중되면 해당 블록은 쓰이지 못하게 되며 플래시 메모리 장치의 수명에 영향을 미치게 된다. 이와 같은 특정 블록의 마모를 피하기 위해 삭제 연산이 모든 블록에 고르게 수행되는 것이 좋다. 이를 마모도 평준화(wear leveling)라고 부른다.

2.2 FTL(Flash Translation Layer)의 주소 변환

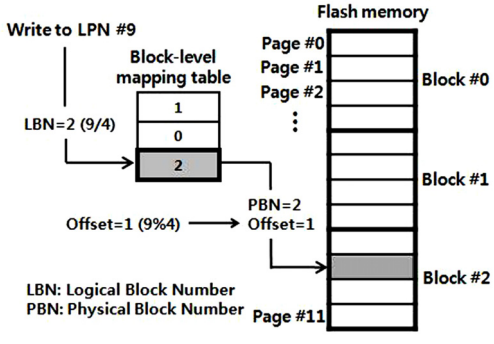
FTL에서 사용되는 주소 변환 기법은 페이지 수준(page-level) 사상, 블록 수준(block-level) 사상, 혼성(hybrid) 사상 기법의 3종류로 분류된다.

페이지 수준 사상 기법은 플래시 메모리의 페이지마다 한 엔트리를 가지는 페이지 사상 테이블(PMT : Page-level Mapping Table)을 유지하며, 파일 시스템에서 요청되는 논리 페이지 번호를 물리 페이지 번호로 변환한다(Gupta et al.[1]). <그림 2>는 페이지 수준 사상 기법의 예를 보여주고 있다. FTL이 논리 페이지 번호 2에 대한 쓰기 요청을 받아 물리 페이지 9에 저장하고 있다. 만약, FTL이 논리 페이지 번호 2에 대한 변경 요청을 받으면, 원래 페이지는 무효로 만들고, 새 페이지를 할당하여 요청된 데이터를 기록한다. 이러한 페이지 수준 사상 기법은 PMT의 크기가 매우 커질 수 있는 문제점을 가지고 있다. <표 2>는 3가지 사상 기법에 대하여 사상 테이블의 크기를 예를 들어 보여주고 있다. 한 블록의 크기는 512KB, 한 블록의 페이지 개수는 128개를 가정하였다. 또한, 사상 테이블의 각 엔트리의 크기는 8바이트라고 가정하였다. <표 2>에서 볼 수 있는 것처럼 PMT는 수십 MB 이상의 크기가 되므로 RAM이 많이 필요하게 되어 사용이 어렵다.

사상 테이블의 크기를 줄이기 위해 블록 수준 사상 기법을 사용할 수 있다. 블록 수준 사상 기법의 사상 테이블은 플래시 메모리의 블록마다 한 엔트리를 갖는다. <그림 3>은 블록 수준 사상 기법의 예를 보여주고 있다.



〈그림 2〉 페이지 수준 사상 기법



〈그림 3〉 블록 수준 사상 기법

블록 수준 사상 기법에서는 논리 주소가(bi, pi)로 구성된다. 이때, bi는 논리 블록 번호이고 pi는 블록 내 페이지 번호이다. FTL은 bi를 사용하여 물리 블록 번호로 사상한다. pi는 물리 블록 내의 페이지 번호가 된다. <그림 3>에서 FTL은 논리 페이지 번호 9에 대한 쓰기 요청을 받았다. FTL은 논리 페이지 번호로부터 논리 블록 번호 2와 블록 내 페이지 번호인 1을 구한다. 그리고, 물리 블록 번호 2의 두 번째 페이지(즉, 블록 내 페이지 번호가 1)에 데이터를 저장한다. 블록 수준 사상 기법은 데이터 변경 연산을 수행할 때 데이터 복사 연산을 해야 하는 문제점을 가

지고 있다. FTL이 데이터 변경 요청을 받으면 새 블록을 할당하고 새 블록 내의 페이지 pi에 요청된 데이터를 저장한다. 이때, 이전 블록에 있는 유효 페이지를 새 블록에 복사해야 하는 오버헤드가 요구된다. <표 2>에서 보는 것처럼 블록 사상 테이블(BMT)의 크기는 PMT의 크기보다 매우 작다. 그러나, 저장장치 용량이 매우 커지면 BMT의 크기도 수십 MB 이상의 크기가 되므로 여전히 사상 테이블의 크기는 문제가 된다.

〈표 2〉 저장장치 용량에 따른 사상 테이블의 크기

저장장치 용량	블록 개수	PMT 크기	BMT 크기	HMT 크기
32GB	216	64MB	512KB	512KB + 약 6MB
256GB	219	512MB	4MB	4MB + 약 50MB
1TB	221	2GB	16MB	16MB + 약 200MB

사상 테이블의 크기 문제와 데이터 복사의 문제 등을 해결하기 위해 혼성 사상 기법이 연구되어 왔다(Kim et al.[6])(Kang et al.[5], (Lee et al.[11])(Park et al.[2]). 혼성 사상 기법은 BMT를 유지하면서, 변경 블록들에 대해서 페이지 수준의 사상으로 관리하는 로그 블록을 할당하고 로그 블록에 변경된 데이터를 저장한다. 즉, 블록 수준 사상과 페이지 수준 사상이 혼용되는 방식이다. 혼성 사상 기법에서는 데이터의 변경 시에 유효 페이지를 복사해야 하는 오버헤드가 없다. 그러나, 저장장치의 용량이 매우 커지면, 사상 테이블의 크기가 커지는 문제점을 여전히 가지고 있다. 예를 들어, 전체 블록의 10%가 변경되는 경우에 혼성 사상 테이블(HMT)의 크기를 <표 2>에서 보이고 있다. 기본적으로 BMT를 유지하면서 추가로 각 변경 블록마다 로그 블록을 위한 하나의 PMT를 만든다고 가정하였다. 로그 블록을 위한 PMT의 크기는 1KB이다. 왜냐하면, 블록의 페이지 수가 128개이고 각 페이지마다 한 엔트리를 가지는데 엔트리의 크기가 8바이트 이므로 128×8 = 1024이기 때문이다. 저장장치 용량이 1TB인 경우 HMT의 크기는 200MB이상이 되어 많은 RAM이 필요하다. 또 다른 문제점으로는, 로그 블록이 다 채워지면 원래의 데이터 블록과 로그 블록의 유효 데이터를 새 데이터 블록에 복사하는 합병(merge) 연산을 수행해야 한다. 합병 연산은 많은 복사 연산을 수행해야 하는 문제점이 있다. 이를 해결하기 위하여 합병 연산을 수행하지 않는 SAT 기법이 연구되었다(Ryu[13]).

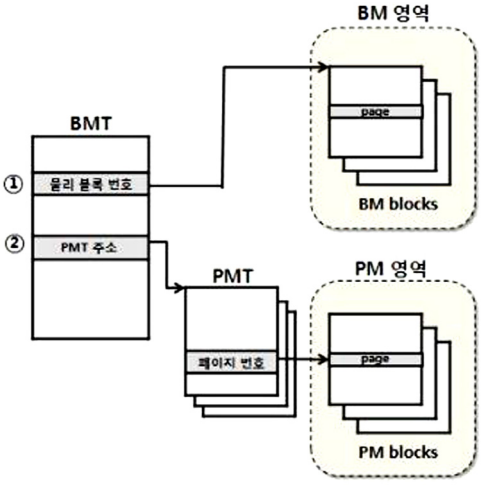
2.3 SAT(Switchable Address Translation)

SAT은 혼성 사상 기법으로서 변경이 없는 블록은 블록 사상 테이블로 관리하고 변경된 블록은 페이지 사상 테이블로 관리한다. SAT은 플래시 메모리의 블록을 BM(Block-·level management) 영역, PM(Page-level management) 영역, 가용(free) 블록 리스트로 구성한다. <그림 4>는 SAT 기법의 주소 사상 테이블을 도시하고 있다. SAT은 블록마다 한 엔트리를 갖는 블록 사상 테이블(BMT)을 관리한다. 논리 블록에 처음 데이터가 쓰여질 때 SAT은 가용 블록 리스트에

서 새 블록을 할당하여 저장한다. 이 블록은 BM 블록이 되어 BM 영역에 포함된다. BM 영역의 블록은 BMT로 관리된다. <그림 4>에서 ①은 BM 블록에 사상되는 논리 블록의 엔트리이다. 이 엔트리는 사상된 물리 블록의 번호를 갖는다.

논리 블록의 페이지가 변경되면 SAT은 가용 블록 리스트에서 새 블록을 할당하여 요청된 데이터를 저장한다. 이 블록은 PM 블록이 되어 PM 영역에 포함된다. 이때, 변경되는 BM 블록도 PM 블록으로 전환하여 페이지 사상 테이블(PMT)로 관리한다. PMT는 변경된 논리 블록을 위해 하나씩 동적으로 생성된다. <그림 4>에서 ②는 PM 블록으로 사상되는 논리 블록의 엔트리이다. 이 엔트리는 대응되는 PMT의 주소를 갖는다. 이 논리 블록에 쓰여지는 데이터는 PM 영역의 블록에 저장된다.

기존의 혼성 사상 기법과 달리 한 PM 블록이 다 사용되었을 때마다 합병 연산을 수행



〈그림 4〉 SAT 기법의 사상 테이블 구조

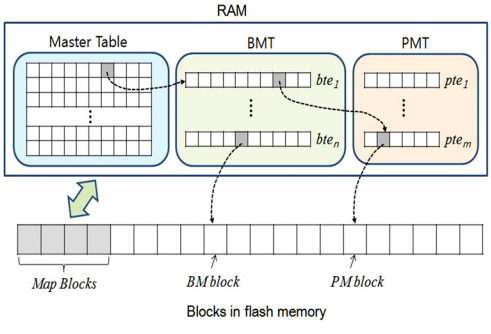
하지 않고 가용 블록 리스트에서 PM 블록을 새로 할당하여 사용한다. 따라서, 수행 비용이 큰 합병 연산을 수행하지 않아 FTL 성능이 향상될 수 있다.

3. 사상 테이블 캐싱 기법

본 논문에서 제안하는 사상 테이블 캐싱 기법은 SAT 기법에 기반을 둔다. 그러나, 일반적인 FTL에도 쉽게 적용이 가능하다. 제안한 사상 테이블 캐싱 기법은 사상 테이블을 플래시 메모리에 저장하며 특정 테이블 엔트리가 참조되어 필요해지면 해당 엔트리가 저장되어 있는 플래시 메모리의 페이지를 RAM에 캐싱하여 사용한다. 이처럼 사상 테이블 전부가 RAM에 적재되는 것이 아니므로 사상 테이블의 엔트리 위치를 관리하기 위한 별도의 테이블을 둔다.

<그림 5>는 사상 테이블이 RAM에 캐싱되었을 때의 구조를 보여주고 있다. 그림에서 상단은 RAM에 존재하는 테이블 내용이고 하단은 플래시 메모리의 블록이다. BMT와 PMT들은 플래시 메모리의 특정 영역인 맵 블록(map block)에 저장되어 있다. 맵 블록은 사상 테이블 또는 사상 테이블의 엔트리가 생성되거나 변경될 때마다 그 내용을 로그 방식으로 저장하는 영역이다. 맵 블록에 있는 사상 테이블의 한 엔트리가 참조되어 필요해지면 이를 RAM에 적재한다. 엔트리를 RAM에 적재할 때 엔트리가 저장되어 있는 플래시 메모리의 페이지 내용을 전부 RAM에 적재한다. 이것은 플래시 메모리의 읽기/쓰기 단위가 페이지이기 때문이다. 한

페이지 크기가 4KB이고 BMT의 한 엔트리 크기가 8바이트라면, 한 페이지에는 512개의 BMT 엔트리가 있게 된다. 캐시된 512개의 BMT 엔트리는 이제 RAM에서 곧바로 참조될 수 있다. 첫 512개의 BMT 엔트리가 저장된 페이지를 “BMT 페이지 0”이라하고, 다음 512개의 BMT 엔트리가 저장된 페이지를 “BMT 페이지 1”이라 부른다. 나머지 BMT 엔트리가 저장되는 페이지들도 순서대로 번호를 붙인다. RAM에 적재된 엔트리의 내용이 변경되면 데이터의 신뢰성(reliability)을 위하여 변경된 내용을 즉시 플래시 메모리의 맵 블록에 저장한다.



〈그림 5〉 캐싱된 사상 테이블의 구조

BMT 엔트리들의 현재 위치를 관리하기 위해 마스터 테이블(master table)을 RAM에 만든다. 마스터 테이블은 시스템 초기화 과정에서 플래시 메모리의 맵 블록을 스캔하여 BMT 엔트리에 대한 정보만을 수집함으로써 만들어진다. 마스터 테이블의 엔트리는 각 BMT 페이지의 현재 위치에 대한 정보를 갖는다. 예를 들어, 마스터 테이블의 엔트리 값이 0으로 시작한다면 나머지 값은 플래시 메모리의 물리 블록과 그 블록 내의 페이지 번호를 의미한다. 또한, 마스터 테이블의 엔트리 값이 1로 시작한다면 나머지 값은 RAM에 적재된 페이지의 메모리 주소를 의미한다. <표 2>에서 저장장치가 1TB인 경우, BMT는 221개의 엔트리가 있다. 한 페이지에 512개의 BMT 엔트리가 있다면 마스터 테이블은 4096개(= 221/29)의 엔트리를 갖는다. 마스터 테이블의 한 엔트리의 크기가 8바이트라면 마스터 테이블의 크기는 32KB가 된다. 이 크기는 매우 작기 때문에 마스터 테이블 전체 내용을 RAM에 상주시킨다.

PMT의 현재 위치는 BMT에서 관리한다. 제 2.3절의 SAT 기법에서 설명했듯이 BMT의 엔트리는 대응하는 논리 블록의 변경 여부에 따라 BM 블록 번호 또는 PMT의 주소를 갖는다. PMT의 크기는 일반적으로 플래시 메모리의 페이지 크기보다 작다. 예를 들어, 한 블록의 페이지 수가 128개이고 PMT 엔트리의 크기가 8바이트로 가정한다면 1 KB(= 128*8)가 된다. 이때, 플래시 메모리의 한 페이지가 4KB라면 4개의 PMT가 저장될 수 있다. 이처럼 한 PMT가 플래시 메모리의 페이지보다 작기 때문에, PMT가 새로 생성되거나 그 내용이 변경되면 PMT 전체를 플래시 메모리의 맵 블록에 저장한다. BMT의 엔트리가 PMT의 주소를 갖는 경우, 이 엔트리가 RAM에 적재되어 있다면, 엔트리는 RAM의 PMT 주소와 맵 블록 내의 PMT 주소(블록 번호, 블록 내 페이지 번호) 값을 갖는다. 그러나, 이 엔트리가 RAM에 없고 플래시 메모리의 맵 블록에만 저장된 상태라면 엔트리는 맵 블록 내의 PMT 주소만 갖는다.

4. 성능 평가

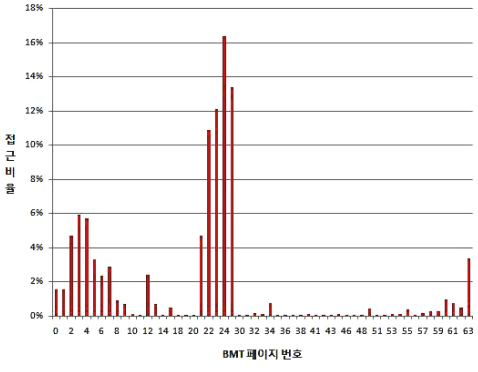
본 논문에서 제안한 사상 테이블 캐싱 기법을 평가하기 위하여 시뮬레이터를 C 언어로 구현하였다. 구현한 시뮬레이터는 실제 디스크 입출력 트레이스를 기반으로 수행되며 사상 테이블이 사용하는 RAM 크기를 계산한다. 플래시 메모리는 32GB 크기를 사용하였고 플래시 메모리의 파라미터 값을 <표 3>에서 보여주고 있다.

〈표 3〉 플래시 메모리 파라미터

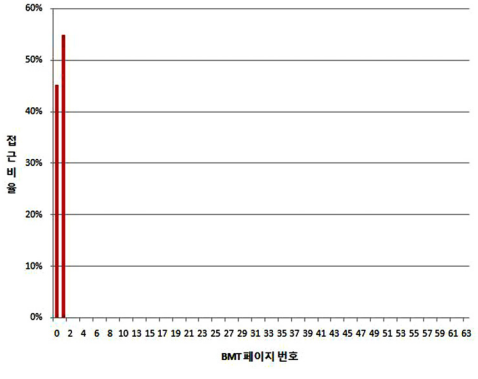
파라미터	값
페이지 크기	4KB
블록의 페이지 수	128
블록 수	65536

실험에 사용할 워크로드를 위해 윈도우즈 XP가 탑재된 노트북 PC에서 약 1주일 동안 여러 응용 프로그램(웹 브라우저, 미디어 플레이어, 편집 등)을 수행하면서 디스크 입출력 트레이스를 수집하였다. 트레이스 수집을 위해 diskmon 유틸리티를 사용하였다. 두 번의 수집 과정을 수행하였고 수집한 트레이스를 각각 PC-1, PC-2라고 부르겠다. 또한, Storage Performance Council(SPC)에서 제공하고 있는 OLTP 환경의 디스크 입출력 트레이스를 사용하였다(OLTP). OLTP 워크로드는 PC 워크로드보다 쓰기 비율이 높은데 변경된 블록 수는 현저하게 적어 매우 편중된(skewed) 쓰기 접근 패턴을 보이고 있다.

시뮬레이터에서 한 BMT 페이지에 512개의 BMT 엔트리가 저장된다. 총 BMT 페이지 수는 64개이다. <그림 6>과 <그림 7>은 BMT 페이지에 대한 접근 비율을 보여주고 있다. 일부 BMT 페이지들에 대한 접근이 편중되고 있음을 알 수 있다. 특히 OLTP의 경우 2개의 BMT 페이지에만 편중 접근되고 있다. 따라서, BMT 테이블 중에서 자주 접근되는 일부 엔트리들만 RAM에 적재하여도 적중률이 높게 된다.



〈그림 6〉 BMT 페이지의 접근 : PC-1 워크로드



〈그림 7〉 BMT 페이지의 접근 : OLTP 워크로드

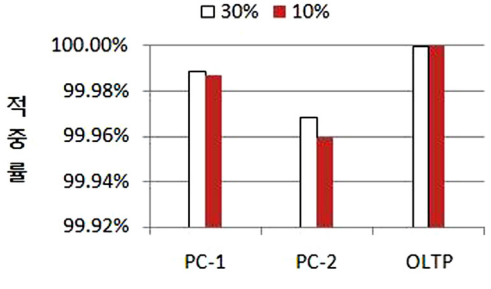
실험에서 RAM에 적재될 수 있는 BMT 페이지 한도를 총 BMT 페이지 수의 30%,

10%로 정하고, RAM에 적재된 개수가 이 한도를 넘으면 LRU 교체 정책을 사용하여 희생 BMT 페이지를 맵 블록에 저장하였다. <표 4>는 64개의 BMT 페이지 중에서 참조되어 RAM에 적재된 개수 및 교체된 BMT 페이지 수를 보여준다.

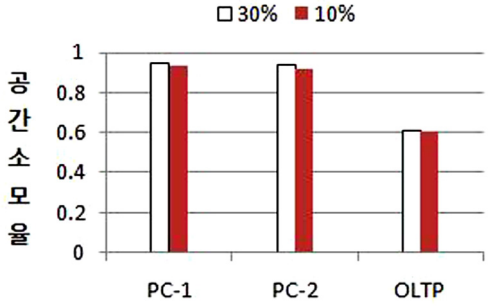
〈표 4〉 참조된 BMT 페이지 수와 교체된 페이지 수

워크로드	참조된 BMT 페이지 수	교체된 BMT 페이지 수	
		30%	10%
PC-1	55	127	147
PC-2	60	150	215
OLTP	2	0	0

<그림 8>은 BMT 페이지의 참조시 RAM에서 참조한 비율(참조 적중률)을 보여주고 있다. OLTP의 경우 캐시 적중률이 거의 100%에 이르고 있어 사상 테이블을 참조하기 위하여 플래시 메모리를 접근할 필요가 없다. 따라서, 기존 SAT 기법과 비교하여 사상 테이블의 참조 시간 비용이 차이가 없다. <그림 9>는 SAT 기법과 비교한 사상 테이블의 공간 소모율을 보여준다. OLTP의 경우 60%의 공간만 소모하여 공간 비용을 대폭 줄일 수 있다.



〈그림 8〉 BMT 페이지 참조 적중률



〈그림 9〉 사상 테이블의 공간 소모율

6. 결 론

본 논문에서는 e-비즈니스를 위해 사용되는 플래시 메모리 기반의 모바일 저장장치에서 사상 테이블의 캐싱 기법을 연구하였다. 기존 플래시 메모리 저장장치는 매우 큰 주소 사상 테이블을 RAM에 유지해야 하는 문제점을 가지고 있다. 사상 테이블의 캐싱은 사상 테이블 전부를 RAM에 유지할 필요가 없어 공간 비용을 줄이는 대신에 캐시 부재가 발생했을 때 이를 처리하기 위한 시간 비용이 증가한다. 트레이스 기반의 시뮬레이션을 통해 본 논문에서 제안한 사상 테이블 캐싱 기법은 공간 비용을 대폭 줄이고 시간 비용은 크게 증가하지 않음을 알 수 있었다. 특히, e-비즈니스의 온라인 트랜잭션 워크로드에서는 많은 공간 비용 절감 효과를 보였다. 또한, 다양한 워크로드에서 사상 테이블의 캐시 적중률이 매우 높아 캐시 부재로 인한 수행 비용은 매우 작았다. 향후 연구로는 e-비즈니스 데이터의 접근 패턴에 보다 특화된 사상 테이블 엔트리의 교체 알고리즘에 대한 연구가 필요하다.

참 고 문 헌

[1] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL : A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mapping," in Proc. International Conference on Architectural Support for Programming Languages and Operating Systems, 2009, pp. 229-240.

[2] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J. Kim, "A Reconfigurable FTL(flash translation layer) Architecture for NAND Flash-based Applications," ACM Transaction on Embedded Computing Systems, Vol. 7, No. 4, Jul. 2008.

[3] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories," ACM Computing Surveys, Vol. 37, No. 2, 2005.

[4] Intel Corp. "Understanding the Flash Translation Layer(FTL) Specification," 1998.

[5] J. Kang, H. Jo, J. Kim, and J. Lee, "A Superblock-based Flash Translation for NAND Flash Memory," in Proc. EMSOFT, 2006, pp. 161-170.

[6] J. Kim, J. Kim, S. Noh, S. Min, and Y. Cho, "A Space-efficient Flash Translation Layer for Compactflash Systems," IEEE Transactions on Consumer Electronics, Vol. 48, No. 2, May, 2002, pp. 366-375.

[7] J. Shin, Z. Xia, N. Xu, R. Gao, X. Cai, S. Maeng, and F. Hsu, "FTL Design Exploration in Reconfigurable High-performance SSD for Server Applications," in Proc. ACM Conference on Supercomputing, 2009.

[8] OLTP Trace from UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

[9] Samsung Electronics, NAND Flash Memory Data Sheet, http://www.sams ungelectronics.com.

[10] T. S. Chung, D. J. Park, S. Park, D. H. Lee, S. W. Lee, and H. J. Song, "A Survey of Flash Translation Layer," Journal of Systems Architecture, Vol. 55, No. 5-6, 2009.

[11] S. Lee, D. Park, T. Chung, D. Lee, S. Park, and H. Song, "A Log Buffer-based Flash Translation Layer using Fully-associative Sector Translation," ACM Transaction on Embedded Computing Systems, Vol. 6, No. 3, Jul. 2007.

[12] S. Lee, D. Shin, Y. Kim, and J. Kim, "LAST : Locality-aware Sector Translation for NAND Flash Memory-based Storage Systems," SIGOPS Operating Systems Review, Vol. 42, No. 6, Oct. 2008, pp. 36-42,

[13] Y. Ryu, "SAT : Switchable Address Translation for Flash Memory Storage," IEEE Computer Software and Applications Conference(COMPSAC), Jul. 2010.

저 자 소 개

류연승	(E-mail : ysryu@mju.ac.kr)
1990년	서울대학교 계산통계학과 졸업 (학사)
1992년	서울대학교 대학원 전산과학과 졸업 (석사)
1996년	서울대학교 대학원 전산과학과 졸업 (박사)
1996년~2000년	삼성전자 선임연구원
2000년~2002년	한림대학교 정보통신공학과 조교수
2003년~2009년	명지대학교 컴퓨터소프트웨어학과 부교수
2010년~현재	명지대학교 컴퓨터공학과 부교수
관심분야	모바일 시스템, 스토리지 시스템, 멀티미디어 시스템

양수현	(E-mail : sh871201@naver.com)
2007~현재	명지대학교 컴퓨터공학과 (학사과정)
관심분야	모바일 시스템