플래시 메모리를 위한 효율적인 선반입과 비동기 쓰기 기법

(Efficient Prefetching and Asynchronous Writing for Flash Memory)

박 광 희 † 김 덕 환 ††
(Kwanghee Park) (Deok-Hwan Kim)

요 약 휴대용 저장장치로 각광 받고 있는 NAND 플래시 메모리의 용량이 커지면서 기존의 파일시스템과 플래시 메모리 컨트롤러 간의 중간 매개체 역할을 해주는 FTL(Flash Translation Layer)의 주소 변환 및 수명 관리 기법이 점차 중요해지고 있다. 본 논문에서는 연속적인 논리 주소 요청이 물리 주소가 인접한 경우의 값을 기록하는 연속성 카운터를 제안하여 주소 변환 횟수를 감소 시켰으며 이와 함께 자주 쓰이는 주소의 페이지들을 미리 주 메모리에 선반입하여 플래시 메모리의 입출력 성능을 향상시켰다. 또한 쓰기 빈도가 높은 주소를 예측하고 잦은 쓰기를 방지하기 위해 2비트 쓰기 예측과 비동기 쓰기 기법을 제시하여 쓰기 성능과 플래시 메모리의 수명을 향상 시켰다. 실험 결과 본 논문에서 제안하는 CFTL (Clustered Flash Translation Layer)이 기존 FTL들보다 주소 변환 성능이 최대 20%, 쓰기 시간을 최대 50% 이상 감소시켰다.

키워드 : 플래시 변환 계층, 군집형 해시 테이블, 선반입, 2비트 쓰기 예측

Abstract According to the size of NAND flash memory as the storage system of mobile device becomes large, the performance of address translation and life cycle management in FTL (Flash Translation Layer) to interact with file system becomes very important. In this paper, we propose the continuity counters, which represent the number of continuous physical blocks whose logical addresses are consecutive, to reduce the number of address translation. Furthermore we propose the prefetching method which preloads frequently accessed pages into main memory to enhance I/O performance of flash memory. Besides, we use the 2-bit write prediction and asynchronous writing method to predict addresses repeatedly referenced from host and prevent from writing overhead. The experiments show that the proposed method improves the I/O performance and extends the life cycle of flash memory. As a result, proposed CFTL (Clustered Flash Translation Layer)'s performance of address translation is faster 20% than conventional FTLs. Furthermore, CFTL is reduced about 50% writing time than that of conventional FTLs.

Key words : FTL, Clustered hash table, Prefetch, 2-bit write prediction

· 이 논문은 지식경제부와 한국산업기술재단의 전략기술인력양성사업의 일환과 2007년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2007-313-D00632)

† 학생회원 : 인하대학교 전자공학과
khpark@iesl.inha.ac.kr
†† 정 회 원 : 인하대학교 전자공학과 교수
deokhwan@inha.ac.kr

논문접수 : 2008년 3월 17일
심사완료 : 2008년 11월 22일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제2호(2009.2)

1. 서 론

최근 대용량 NAND 플래시 메모리가 매년 출시되면서 향후 자기 디스크를 대체할 새로운 저장장치로 각광받고 있다[1]. NAND 플래시 메모리의 장점으로 일반 하드 디스크에 비해 랜덤 액세스가 가능해 입출력 속도가 빠르고 충격에 강해 임베디드 시스템이나 산업용 시스템에 유용하다[2]. 하지만 NAND 플래시 메모리는 기존에 수요를 이루었던 자기 디스크와 다르게 읽기/쓰기(페이지, 512B~4KB), 지우기(블록, 32KB~256KB)의 단위가 서로 다르게 구성되어 있어 제자리 갱신(In-place update)이 불가능한 특성이 존재한다. 또한 현재

널리 사용하는 일반적인 파일시스템은 플래시 메모리보다는 자기 디스크에 최적화 되어 있었기 때문에 메모리의 특성을 고려한 중간 계층인 FTL(Flash Translation Layer)이라는 소프트웨어가 필요하다. FTL은 해당 논리 주소의 데이타가 새로이 갱신 될 때마다 다른 물리 페이지에 갱신하기 때문에 변경되는 페이지의 물리 주소를 저장하는 주소 변환 테이블을 포함하고 있다. 또한 잦은 갱신으로 인해 발생하는 무효(Invalid) 블록들을 삭제하고 유효한(Valid) 블록들을 관리하는 공간 재활용 기능을 포함한다.

현재 FTL의 입출력 성능을 높이고자 LFS(Log-structure File System)의 기법들을 차용한 다양한 기법 들이 나왔다[3,4]. 이와 함께 본 연구 그룹에서도 SSD(Solid State Drive)와 같은 대용량 NAND 플래시 메모리 저장장치에서의 주소 변환 성능 향상을 위해 군집형 해시 테이블을 주소 변환 테이블 자료구조로 개발하여 주소 변환 성능이 탁월함을 증명하였다[5]. 이러한 연구 외에도 플래시 메모리의 수명 향상을 위해 블록들의 재사용을 관리하는 기법에 대한 연구도 활발히 이루어지고 있다[6].

아울러 본 논문에서는 이와 같은 FTL의 주 분야 연구 외에도 공간적 지역성(Spatial Locality)을 이용한 페이지 단위 선반입 기법과 연속성 카운터를 이용하여 FTL의 성능을 향상시키고 운영체제와 FTL 간의 입출력 단위가 서로 다름으로 인해 발생하는 과부하를 감소시키는 방법을 다음과 같이 제시하고자 한다.

첫째, 군집형 해시 테이블은 물리적, 논리적 공간적 지역성(Spatial Locality)을 유지한다. 공간적 지역성이 향상되면 플래시 메모리에서는 빠른 성능을 나타내는 순차적인 접근이 가능하다. 또한 하드디스크에서는 암(Arm)의 이동거리가 감소하고 버스트 모드를 오래 유지 할 수 있으므로 전송률이 향상되어 입출력 성능에 효율적이게 된다. 추가적으로 읽을 데이타를 예측하여 미리 데이타를 가져오는 선반입 기법은 저장장치 입출력 성능향상에 도움을 준다. 따라서 본 논문에서는 군집형 해시 테이블에서 공간적 지역성을 가지고 있는 버켓 내의 모든 페이지 데이타를 선반입하는 기법을 제시한다. 그리고 반복적인 주소 변환을 감소시키기 위해 군집형 해시 테이블에 연속성 카운터를 설정하고 연속성 카운터에 해당하는 물리 블록에 관한 주소 변환 요청의 경우 주소 변환 루틴을 거치지 않는 기법을 제안한다.

둘째, 운영체제에서 저장 장치의 반복적인 데이타 갱신의 과부화를 감소시키기 위해 리눅스에서는 페이지 캐시, BSD 계열에서는 통합 버퍼 캐시(Unified Buffer Cache)와 같은 메인 메모리의 자료구조를 이용한다. 이는 모든 데이타를 우선 메인 메모리에 저장하여 반복적인 데이타 갱신을 최소화하고 사용이 예측 되는 데이타를 미리 선반입하여 저장장치와의 응답 대기 시간 및 과부하를 최소화 시키는 것에 목적이 있다. 가장 널리 쓰이는 인텔 32비트 CPU의 경우 리눅스 커널의 페이지 캐시의 크기가 4KB로 설정되어 있다. 그 이유는 가상 메모리의 페이지를 이용하기 때문에 크기가 동일하다. 하지만 SLC의 NAND 플래시 메모리 같은 경우 현재 출시된 플래시 메모리 대부분이 512B 또는 2KB 크기의 페이지 단위를 사용한다. 즉 페이지 캐시로 인해 1~7개 이상의 페이지가 새로이 갱신이 되지 않았음에도 불구하고 4KB 페이지 크기의 데이타가 갱신되는 과부하를 발생시킨다. 따라서 본 논문에서는 페이지 캐시와 플래시 메모리의 페이지 크기 차이를 극복하기 위한 방법으로 2비트 쓰기 예측 기법을 통한 비동기 쓰기 방법을 제시한다.

또한 USB 휴대용 플래시 메모리의 데이타 입출력을 추출하기 위해 리눅스 블록 I/O Trace를 사용하여 워크로드를 구성하였다. 성능을 평가한 결과 본 논문에서 제안한 선반입 및 연속성 카운터 기법이 기존 FTL보다 주소 변환 성능이 향상됨을 알 수 있고 2비트 쓰기 예측 기법을 통한 비동기 쓰기 기법이 페이지 캐시로 인한 필요 없는 페이지 갱신을 방지하는 결과를 가져왔다.

논문의 구성은 다음과 같다. 2장에서는 FTL의 관련 연구를 소개하고, 3장은 제안하는 CFTL(Clustered Flash Translation Layer)의 구조와 연속성 카운터, 그리고 선반입을 추가한 내용을 기술한다. 이와 함께 4장에서는 리눅스 커널의 페이지 캐시와의 효율적인 동기화를 위하여 2비트 쓰기 예측 기법을 이용한 비동기 쓰기 기법을 설명한다. 5장에서는 몇 가지 워크로드를 통해 실험 결과를 분석하고, 마지막으로 6장에서 본 논문의 결론을 맺는다.

2. NAND 플래시 메모리의 특성 및 관련 연구

첫째 NAND 플래시 메모리는 페이지(512B~4KB)와 블록(16KB 또는 256KB) 단위로 구성되어 있다. 페이지는 NAND 플래시 메모리를 구성하는 최소 단위로 일반적으로 페이지 32~64개가 하나의 블록을 구성한다.

둘째 각 블록들은 갱신되는 횟수가 제한되어 있는 한계를 가지고 있으며 이를 마모도(Wear level)라고 한다. SLC에서는 약 10만 번 그리고 MLC(Multi Level Cell)에서는 약 1만 번 정도의 수명을 가지고 있다[7,8].

셋째 NAND 플래시 메모리는 자기 디스크와 달리 읽기/쓰기/삭제 3개의 동작으로 구분되며 읽기/쓰기 동작은 페이지(512B~4KB) 단위이지만 삭제 동작은 블록(16KB~256KB) 단위이다. 넷째 블록 단위의 삭제는 페이지 32~64개를 지우는 속도이므로 표 1과 같이 읽기/

표 1 NAND 플래시 메모리의 수행작업 속도 비교

수행작업	시간
읽기	25 μs
쓰기	226~300 μs
삭제	1.5~2 ms

쓰기 속도보다 매우 느리다.

기존에 사용하던 하드 디스크 전용 파일시스템은 이와 같은 NAND 플래시 메모리의 특성을 고려하지 않고 설계되었기 때문에 기존의 파일시스템과 NAND 플래시 메모리와의 호환을 위해서는 FTL(Flash Translation Layer)이라는 미들웨어를 거쳐서 수행해야 한다. FTL이 수행하는 역할은 다음과 같다.

첫째 제자리 갱신을 하기 위해서는 쓰기 전 지우기를 수행해야 하므로 이에 따른 삭제 비용이 매우 높다. 그러므로 NAND 플래시 메모리는 제자리 갱신 대신 외부 갱신(Out-place update)를 한다. 외부 갱신 할 때마다 파일시스템의 논리 주소(LBA: Logical Block Address)와 사상되는 플래시 메모리의 물리 주소(PBA: Physical Block Address)가 갱신되므로 주소 변환 테이블 기능을 포함한다. 둘째 NAND 플래시 메모리를 오래 쓰기 위해서 모든 블록의 마모도를 평준화 시키는 기능을 가지고 있다. 셋째 삭제 속도가 읽기, 쓰기보다 시간이 더 오래 걸리므로 플래시 메모리의 유효하지 않은 블록을 일괄적으로 삭제하는 블록 수거 기법(Garbage Collection) 기능을 포함한다.

플래시 메모리의 성능 향상을 위해 많은 연구들이 진행 되었다. 또한 이러한 기법은 대부분 로그 기반의 파일시스템(Log-structured File System) 기법을 차용하는 경우가 많고 해당 로그의 갱신의 위치에 따른 다양한 연구 결과도 존재한다.

[3]은 BAST(Block Associative Sector Translation) 기법이라고도 많이 불리며, 로그 블록을 일정 영역에 할당하고 갱신이 발생하는 블록을 로그 블록으로 대체하여 순차적으로 사용하는 기법이 있으며, [4]는 BAST에서 한 개의 블록이 로그 블록 하나를 모두 사용하는 경우 로그 블록의 페이지 사용률이 낮다는 것을 고려하여 블록단위로 로그 블록을 할당하지 않고 외부 갱신이 되는 순서대로 로그 블록을 채우는 방식을 사용하는 FAST(Fully Associative Sector Translation) 기법을 제시하였다.

본 연구에서는 로그 방식을 사용하지 않고 교체 블록 기법을 사용하는 일반적인 NFTL(NAND Flash Translation Layer)과 AFTL(Adaptive Flash Translation Layer)을 기준으로 관련 연구를 좀 더 자세히 설명한다. 또한 로그 기법을 사용하는 BAST, FAST FTL이나 다음 설명하는 NFTL과 AFTL은 모두 페이지, 블록 단위의 주소 변환을 혼용하고 있다.

2.1 NFTL(NAND Flash Translation Layer)

NFTL은 흔히 말하는 NAND 플래시 메모리 전용 FTL이다[3,4,9-13]. 페이지 단위 주소 변환 기법의 가장 큰 단점인 주소 변환 테이블의 크기를 줄이기 위해 블록 단위의 주소 변환 방식을 적용했다. 그림 1과 같이 LBA(Logical Block Address)의 값을 블록 당 페이지

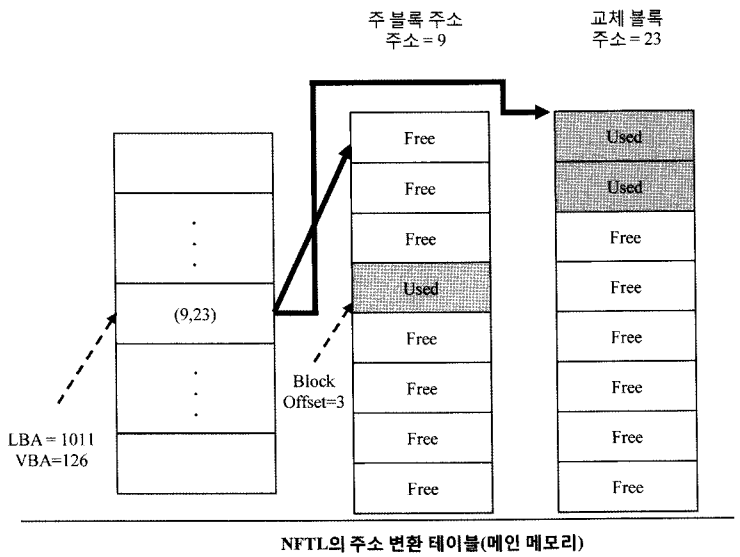


그림 1 NFTL에서 주소 변환 기법

개수로 나누어 몫은 VBA(Virtual Block Address)로 사용하고 나머지 값은 오프셋을 인덱스로 사용한다. 그리고 주소 변환 테이블로 접근 시 VBA의 값을 통해 주 블록(Primary block)과 교체 블록(Replacement block)의 주소를 가지고 있는 엔트리에 접근한다. 먼저 주 블록 PBA가 존재 할 경우 나머지 값인 블록 오프셋으로 페이지 단위 접근을 시도한다.

하지만 주 블록 내의 페이지 데이타가 갱신되어 유효하지 않아서 교체 블록을 참조하는 경우, NFTL의 방식이 크게 두 가지 방식으로 나뉠 수가 있다.

첫째는 교체 블록에 데이타를 쓸 때 주 블록과 동일한 오프셋이 가리키는 페이지에 데이타를 갱신하는 경우, 둘째는 주 블록의 오프셋과 상관 없이 교체 블록의 페이지에 순차적으로 데이타를 갱신하는 경우이다. 전자의 경우 불필요한 교체 블록이 많이 발생하는 단점이 있기 때문에 본 논문에서는 후자의 방법인 교체 블록을 순차적으로 채우는 방법을 설명한다.

앞에서 언급한 바와 같이 데이타가 갱신되어 교체 블록을 참조하는 경우 그림 1과 같이 NFTL은 교체 블록의 스페어 영역을 순차적으로 검색하여 LBA와 매칭되는 해당 페이지를 찾는다. 이러한 블록 단위 주소 변환 기법은 주소 변환 테이블의 크기가 작은 대신 주 블록에서 주소 검색을 실패 할 경우 교체 블록을 순차적으로 다시 검색해야 하는 오버헤드가 존재한다.

2.2 AFTL(Adaptive Flash Translation Layer)

AFTL은 대만 국립 대학교에서 제안한 FTL로 그림 2와 같이 주소 변환 테이블이 블록 단위 해시 테이블과 페이지 단위 해시 테이블로 이루어져 있다[14]. 기본적으로 블록 단위 해시 테이블에서는 NFTL과 같이 블록 단위의 주소 변환 기법을 사용하고, 페이지 단위 해시 테이블에서는 FTL과 같은 페이지 단위의 주소 변환 기법을 사용한다. 페이지 단위의 주소 변환 기법을 사용하는 경우는 교체 블록의 데이타가 유효 상태일 경우 이 데이타들이 앞으로 자주 쓰일 수 있다고 가정하여 페이지 단위 해시 테이블에 저장하여 페이지 단위로 주소를 변환한다. 주 블록일 경우는 오프셋을 이용하여 바로 찾기 때문에 블록 단위 해시 테이블을 사용한다. 페이지 단위 해시 테이블의 크기의 제한이 있으므로 페이지-블록 테이블 간의 교체 기법이 필요한데 이때 LRU를 적용하여 일정 기간 동안 사용하지 않는 슬롯을 블록 단위 해시 테이블로 이동하는 기법을 사용한다.

3. CFTL의 주소 변환 기법

3.1 선반입 및 연속성 카운터를 사용하는 CFTL

CFTL의 주소 변환 기법의 장점은 다음과 같다. 첫째 군집형 해시 테이블은 선형 테이블 기법과 해시 테이블을 혼용하는 기법으로 메모리 사용량이 기존 해시 테이블보다 적고 유사한 주소들을 동일한 버켓(Bucket)에

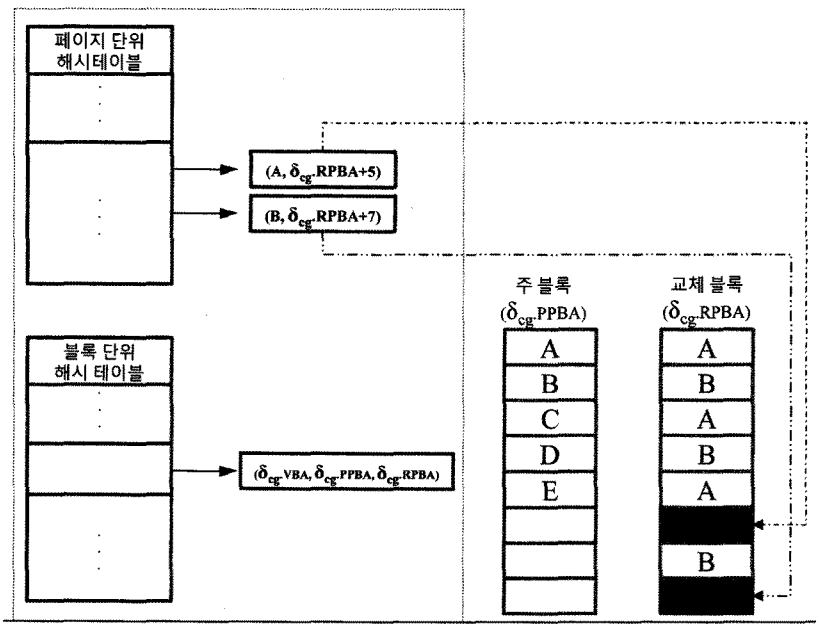


그림 2 AFTL에서 주소 변환 기법

그룹화하므로 지역성 확보가 가능하다[5]. 둘째 2단계 페이지 단위 군집형 해시 테이블은 TLB(Translation Lookaside Buffer)와 같은 캐시를 차용한 기법으로 이와 같은 2단계 테이블은 미스 페널티를 최소화 시키고 상대적으로 주소변환 오버헤드가 큰 블록 단위 군집형 해시 테이블의 접근 빈도를 감소시키는 효과가 있다. 셋째 블록 단위 군집형 해시 테이블에서 주소 변환 성능을 향상시키기 위해 연속성 카운터를 사용하여 연속적인 LBA의 주소변환 요청 시 주소 변환 부담을 줄이는 방법을 제안한다. 넷째 주소 변환 요청 시 페이지 단위 군집형 해시 테이블에 요청된 LBA가 존재할 경우 그 LBA가 소속된 버켓의 전체 페이지를 미리 선반입하여 주소 변환 성능을 향상 시킨다.

그림 3은 CFTL의 주소 변환 방법의 전체 알고리즘을 자료구조와 흐름도로 나타낸 것이다. 우선 특정 LBA의 주소변환 요청이 호스트로부터 들어오면 가장 먼저 1차 페이지 단위 군집형 해시 테이블을 검색하고 매칭되는 PBA를 찾으면 주소 변환은 종료된다.

만약 1차 페이지 단위 군집형 해시 테이블에서 검색에 실패 했을 경우 2차 페이지 단위 군집형 해시 테이블을 검색하여 매칭되는 PBA를 찾으면 주소 변환은 종료 된다. 하지만 2차 페이지 단위 군집형 해시 테이블에서도 검색에 실패 한 경우 최종적으로 블록 단위 군집형 해시 테이블에서 매칭되는 주소 변환 정보를 찾는다. 블록 단위 군집형 해시 테이블은 NFTL과 동일하게 우선 LBA를 VBA와 오프셋으로 변환한다. 이렇게 구해진 VBA를 해싱하여 PPBA와 RPBA의 주소가 저장 되어 있는 버켓을 찾는다. 그리고 매칭된 PPBA와 오프셋에 있는 페이지 데이타가 유효하지 않을 경우 RPBA를 순차적으로 검색하여 해당 물리 주소를 찾는다.

3.2 군집형 해시 테이블

기존의 선형 테이블은 주소의 크기가 확장될 때마다 주소 변환 속도 및 메모리 사용량이 정비례하여 증가하므로 대형 플래시 메모리에 적합하지 않다. 또한 그림 4의 일반 해시 테이블과 같이 Previous, Next 포인터를 포함한 이중 연결 리스트를 사용하는 경우 포인터로 인한 메모리 소비로 인해 선형 테이블보다 메모리 사용량이 필요 이상으로 증가한다. 이러한 문제점을 해결하기 위해 본 논문에서는 군집형 해시 테이블을 제안한다. 군집형 해시 테이블은 CFTL에서 사용하는 주소 변환 자료구조이며 썬마이크로시스템즈 사의 64비트 Ultra-Sparc의 운영체제인 솔라리스에서 가상 메모리 페이지 기법에서 처음 적용했던 자료구조이다[15]. 군집형 해시 테이블은 그림 5와 같이 여러 엔트리들의 집합인 버켓들로 구성되어 있으며 하나의 버켓 안의 각 엔트리를 서브블록이라고 한다. 일반 해시 테이블과 달리 하나의 버켓에 여러 개의 서브블록들이 1차원 선형 배열 형식으로 존재하는 것이 특징이다. 각 서브블록은 해당 논리 주소에 대응하는 물리 주소 및 여러 정보를 포함하고 있다. 각 버켓은 다수의 서브블록들을 포함하기 때문에

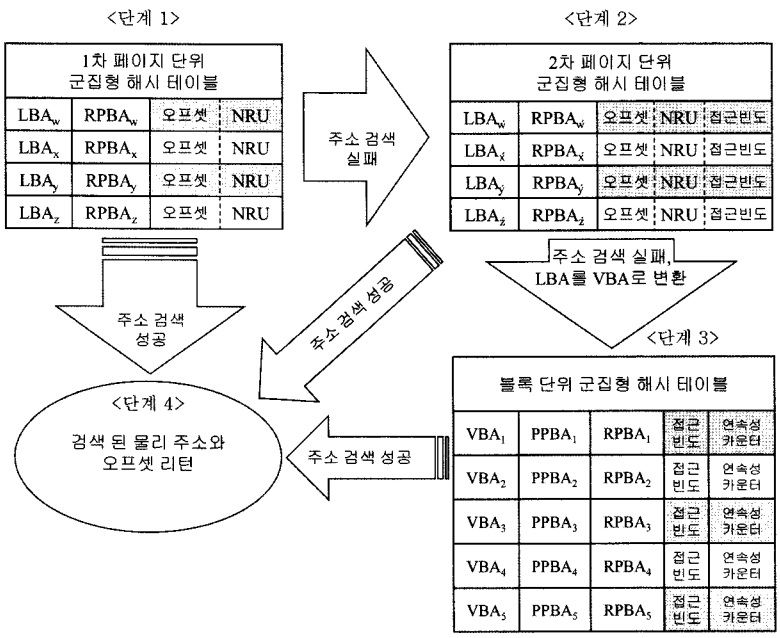


그림 3 CFTL에서 주소 변환 테이블 구조 및 흐름도

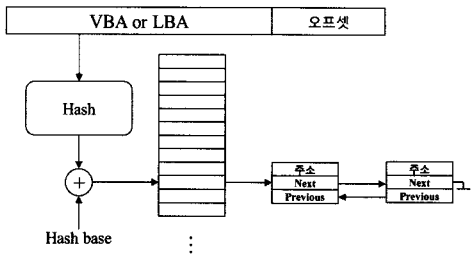


그림 4 일반 이중 연결 해시 테이블

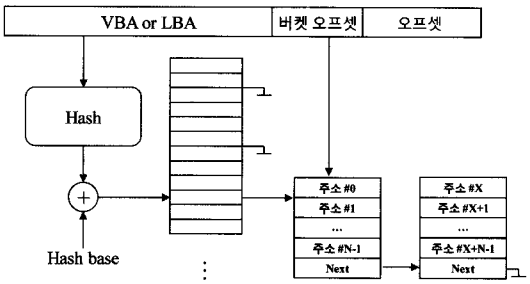


그림 5 군집형 해시 테이블

한 버켓 당 포함하는 서브블록의 개수를 서브블록 팩터(Subblock factor)라 칭한다. 따라서 기존 해시 테이블과 달리 각 주소마다 포인터를 필요로 하는 것이 아니고 각 버켓 별로 포인터를 사용하기 때문에 포인터로 인한 메모리 사용량을 1/서브블록 팩터 만큼 줄일 수 있다. 이와 함께 연속적인 논리 주소에 대응되는 물리 주소들이 한 버켓의 서브블록들에 저장되거나 다음에 설명 될 연속성 카운터를 이용한 연속적인 물리 주소 정보가 저장되어 논리적, 물리적 공간 지역성을 유지 시킬 수 있다.

3.3 연속성 카운터

본 논문에서 제안하는 연속성 카운터는 그림 6과 같이 블록 단위 군집형 해시 테이블 내의 서브블록에 저장되는 정수 값이다. 연속적인 VBA를 통해 접근할 블록의 물리 주소(PBA)들이 인접하는 경우 연속성 카운터는 블록의 개수를 기입하는데 사용한다. 이를 통해 주소변환 메커니즘을 다시 반복하지 않고도 주소 변환이 가능하게 한다. 예를 들어 VBA가 128, PPBA가 386, 연속성 카운터가 8일 경우 VBA의 증가 값이 8 이하인 경우 주소 변환 없이 PPBA의 값을 얻을 수 있다.

VBA	PPBA	RPBA	연속성 카운터	접근 빈도
128	386	800	3	2
131	433	810	3	3
...				
135	220	820	0	1
136	230	750	3	4
Next pointer				

그림 6 연속성 카운터

3.4 2단계 군집형 해시 테이블을 이용한 슬롯 교체 기법

CFTL의 성능을 높이기 위해 하드웨어 캐시 종류 중의 하나인 TLB(Translation Lookaside Buffer)의 개념을 차용하여 2단계 페이지 단위 군집형 해시 테이블을 구현하였다. 기존의 Software TLB 기법을 차용하기 때문에 각 테이블 별로 접근 속도를 향상 시키지는 못하지만 2단계 페이지 단위 군집형 해시 테이블을 적용 할 경우 블록 단위 군집형 해시테이블을 검색하는 미스 페널티를 감소시킬 수 있다. 또한 적중 성공률 보정이 가능하므로 블록 단위 군집형 해시 테이블의 접근 확률 또한 감소시킬 수 있다[16]. 아울러 군집형 테이블의 효율적인 적중률을 유지하기 위해서는 각 테이블간의 이동이 필요 할 수밖에 없다. 따라서 블록-페이지 테이블 또는 페이지-블록 테이블 간의 이동 정책이 필요하다. 알고리즘 1(a)는 NRU 비트 값에 따른 페이지-블록 강등 기법을 설명하고 있다. 알고리즘 1(b)는 블록-페이지 승급 기법 중 읽기 동작 시에 NRU의 참조 비트를 1로 설정하는 방법을 설명하고 있다. 블록-페이지 강등 기법을 예를 들어 설명하자면 호스트로부터 특정 주소 요청이 빈번하게 발생되는 경우, 블록 단위 군집형 해시 테이블의 접근 빈도가 임계치 값을 초과되어 2차 페이지 단위 군집형 해시 테이블로 승급한다. 이와 함께 블록 단위 군집형 해시 테이블의 주소 정보를 담은 서브블록은 초기화 된다. 페이지-페이지 간 승급 정책에서도 이러한 방법이 동일하게 적용된다.

그리고 페이지-블록 강등 정책을 위해 NRU(Not Recently Used) 또는 세컨드 찬스라고 불리는 운영체제의 페이지 교체 기법을 적용한다. 일정 시간 동안 또는 접근이 되지 않을 경우 또는 블록 수거 동작을 수행하는 경우 페이지 단위 군집형 해시 테이블의 해당 주소 정보를 블록 단위 군집형 해시 테이블로 이동시키는 정책을 사용한다[17]. NRU는 유사 LRU 기법(Approximate LRU)이다. NRU는 LRU와 달리 참조, 수정 비트라는 비트 벡터를 사용하며 시스템에서 정해진 주기마다 비트에 따라 최근에 사용하지 않는 슬롯들을 분별할 수 있는 방법이다. 두 개의 비트를 통해 참조된 상황과 수정이 되었는지를 판단하여 표 2와 같이 페이지-블록 간 강등 기법에 적용할 수 있다.

표 2 NRU에서 슬롯 교체 정책

참조 비트	수정 비트	정책
0	0	안 쓰이는 주소로 강등 대상
0	1	갱신된 주소로 현 상태 유지
1	0	참조된 주소로 현 상태 유지
1	1	자주 쓰이는 주소로 현 상태 유지

```
function cleaning time
{
        Search for slots with NRU bits (0,0) in L1 and L2 page-clustered hash tables
        Demote the selected slots to the block-clustered hash table
        NRU bits of all slots are set to (0,0) in L1 and L2 page-clustered hash tables
}
```

(a) 페이지-블록 간 강등 알고리즘

```
function read operation
{
                .....
        If any slot with given address is found in block-clustered hash table
        or L2 page-clustered hash table
        {
                Increase hit count of the corresponding slot
        }

        If the slot is found in L1 or L2 page-clustered hash tables
        {
                NRU reference bit of the corresponding slot is set to 1.
        }

        If hit count of the corresponding slot is greater than the threshold
        {
                Promote the slot to upper clustered hash table
        }
                .....
}
```

(b) 블록-페이지 간 승급 알고리즘

알고리즘 1 군집형 해시 테이블 간의 승급, 강등 알고리즘

예를 들어 특정 주소를 가지고 있는 슬롯의 NRU의 참조, 수정 비트 벡터가 모두 0인 경우는 그 주소는 더 이상 사용되지 않는 것으로 판단하고 해당 슬롯 정보를 페이지 단위 군집형 해시 테이블에서 블록 단위 군집형 해시 테이블로 이동시키고 페이지 단위 군집형 해시 테이블에서 삭제한다.

3.5 버켓의 연속된 페이지 선반입하기

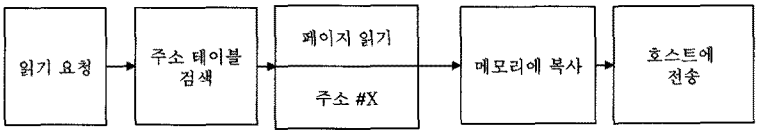
페이지 단위 군집형 해시테이블에 속해 있는 주소 정보들은 과거에 접근 빈도가 높고 NRU 기법을 통해 현재에도 쓰이고 있다는 특징을 갖고 있다. 또한 페이지 단위 군집형 해시 테이블은 앞에서도 언급했듯이 논리적인 주소들의 공간적 지역성을 유지한다. 현재 대부분의 FTL은 일반적으로 LBA 요청 시 그 때마다 주소를 변환하고 데이타를 전송하고 있다. 본 논문에서 제안하는 선반입 기법은 페이지 단위 군집형 해시 테이블에서 하나의 버켓 내에 자주 쓰이는 논리 주소에 해당하는 페이지들을 선반입하는 것이다. 다시 말해서 특정 버켓의 LBA에 대한 읽기 요청 시 인접한 서브블록들의 물리 페이지들을 선반입 할 수 있다. 그림 7(a)는 기존 FTL에서 읽기 동작이 실행되는 순서를 나타낸다. 호스트로부터 읽기 요청이 오면 주소 변환 테이블을 검색하고 페이지를 읽어서 메모리에 복사하고 호스트에게 데이타를 전송한다. 하지만 그림 7(b)는 군집형 해시 테이블의 특정 버켓의 서브블록들이 가리키고 있는 PBA의 페이지들을 선반입하는 방식을 보여준다. 이 경우 반복적인 메모리 적재 및 플래시 메모리에 매 시간마다 명령어를 내리는 것을 방지한다. 또한 다수의 페이지를 선반입하므로 다음 읽기 수행 시에 읽기 동작의 속도를 단축시킬 수 있다는 장점이 있다.

4. 효율적인 플래시 메모리 수명 관리 기법

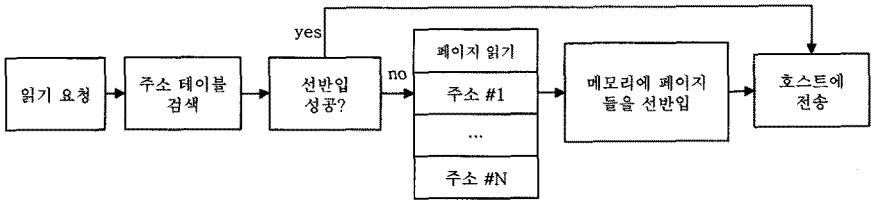
4.1 2비트 쓰기 예측을 통한 비동기 쓰기 기법

운영체제에서 파일의 잦은 갱신을 방지하기 위해 갱신된 데이타를 메인 메모리에 우선 저장하고 일정 시간 후에 동기화된 쓰기 요청이 발생하는 경우 저장장치에 데이타를 쓰는 정책을 제공한다. 이러한 정책으로 인해 사용되는 메인 메모리의 공간을 리눅스에서는 페이지 캐시라 한다. 페이지 캐시는 데이타가 갱신되어 쓰기 요청이 반복될 때 저장장치의 쓰기 횟수를 감소시키는 역할을 한다. 이러한 역할을 통하여 운영체제 수준에서 블록 디바이스에 동작 명령을 최소화 시켜 오버헤드를 감소시킬 수 있다.

하지만 이러한 운영체제 수준에서의 페이지 캐시 기법의 문제점은 물리 저장 장치의 최소 단위로 데이타를 관리하는 것이 아닌 페이지 캐시의 최소 단위인 4KB를 기준으로 한다. 하지만 대부분 블록 디바이스의 최소 단



(a) 일반 FTL에서 읽기 동작



(b) CFTL에서 선반입을 포함하는 읽기 동작

그림 7 일반 FTL과 CFTL의 읽기 동작 비교

위인 섹터는 512B이다. 즉, 쓰기 단위가 약 1/8로 작으며 더욱이 플래시 메모리와 같은 경우 쓰기 동작의 최소 단위인 페이지가 512B 또는 2KB의 수준임을 감안한다면 페이지 캐시 수준의 4KB 반복적인 쓰기 요청은 플래시 메모리에 적합하지 않음을 유추 할 수 있다. 예를 들어 페이지 캐시는 4KB 영역 중에 일부 영역이 갱신 된다면 플래시 메모리의 페이지를 많게는 8개 적게는 2개의 데이타를 갱신해야 하는 것이다. 따라서 플래시 메모리 쓰기 단위에 적합한 비동기 쓰기 기법을 제시해야 할 필요가 있다.

CFTL에서는 플래시 메모리의 쓰기 최소 단위에 적합한 비동기 쓰기 방법을 제시하기 위해 컴퓨터 구조에서 사용하는 2비트 분기 예측(2 bit branch prediction) 기법을 적용했다[16]. 이 기법은 분기 명령어에 의해 해당 명령어 순서가 두 가지 경우로 갈라지는 경우 과거 분기 패턴을 가지고 다음 명령어들의 순서를 선택하는 방법으로 널리 알려져 있다.

이를 CFTL에서는 2비트 쓰기 예측이라고 부르며 그림 8과 같이 4가지의 상태로 나눈다. 2 비트를 통하여 반복되는 해당 주소는 다음 요청 후에도 다시 들어 올 수 있다고 판단한다. 예측 테이블의 비트 벡터(2비트) 값이 10, 11인 경우 자주 갱신되는 LBA로 판단하고 휘발성 메모리인 쓰기 버퍼에 갱신된 페이지를 적재하여 지연된 쓰기를 적용한다.

그림 9와 같이 LBA x_1, x_2, x_3는 VBA X로, LBA y_1, y_2는 VBA Y로, LBA z_1는 VBA Z로 LBA w_1,

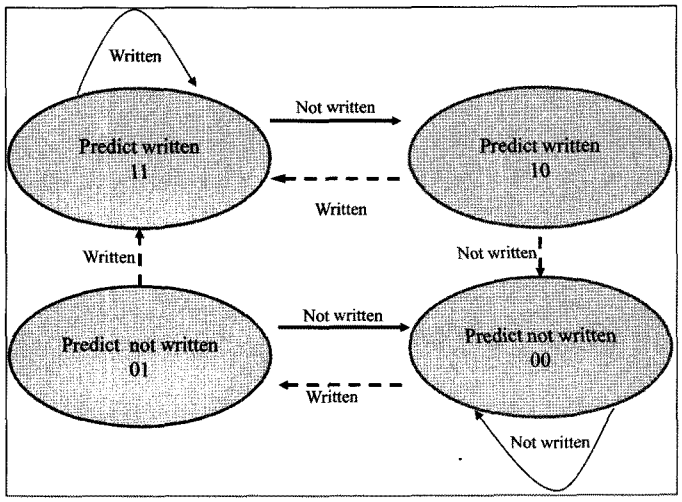


그림 8 2비트 쓰기 예측 기법

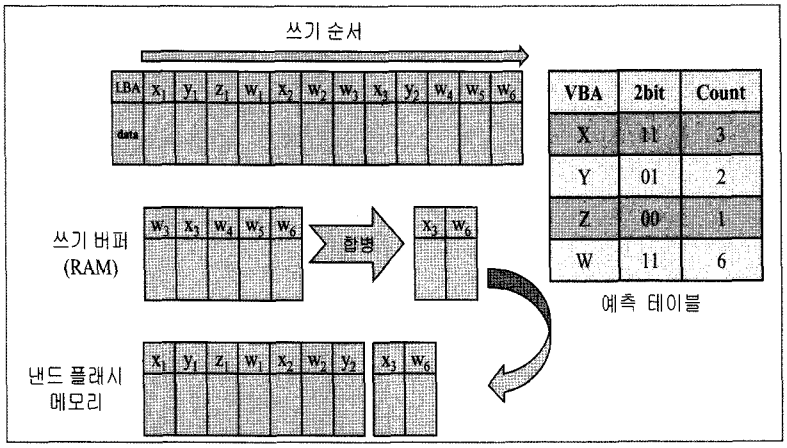


그림 9 2비트 쓰기 예측 기법을 이용한 비동기 쓰기 사례

w_2, w_3, w_4, w_5, w_6은 VBA W로 각각 수렴한다. 예를 들어 VBA X에 속하는 LBA x는 x_1, x_2, x_3 총 3번 쓰기 요청이 들어왔고, 쓰기 요청이 들어오는 동안 2 bit 쓰기 비트가 00 -> 01 -> 11로 변경되면서 3번째 쓰기 요청인 x_3은 NAND 플래시 메모리에 바로 쓰이는 것이 아니라 쓰기 버퍼에 적재된다. 쓰기 순서에 따라 이러한 요청들이 반복되면 반복되는 LBA들의 데이타는 최신 데이타만 NAND 플래시 메모리에 쓰게 된다. 쓰기 버퍼는 가득 차거나, 커널 타이머에 의해 일괄적으로 플래시 메모리에 적재한다. 이러한 쓰기 지연 기법은 플래시 메모리에 쓰기 횟수를 감소시키고 플래시 메모리의 수명을 향상시킬 수 있다.

4.2 블록 수거 기법(Garbage Collection)

일반적으로 NFTL의 블록 수거 기법은 기존의 주, 교체 블록의 유효한 데이타를 새로운 주 블록에 통합한 후 두 개의 블록을 지우는 작업을 한다. 그림 10과 같이 CFTL에서 블록 수거 기법은 기본적으로 NFTL에 기반을 두며 이와 함께 블록 수거 기법의 오버헤드를 감소시키는 방법은 다음과 같다. 첫째 주소 변환 요청 시에 선반입 한 페이지가 히트 할 경우 남는 시간을 블록 수거 기법에 할당 할 수 있다. 둘째 쓰기 동작 시에 2비트 쓰기 예측을 통해 쓰기가 한번 지연될 경우 플래시 메모리에 쓰는 시간(226㎲)을 절약 할 수 있으므로 약 9번 이상 쓰기 지연이 진행 될 경우 블록 하나를 지울 수 있는 시간인 2ms 이상의 시간을 절약하므로 블록 수거 기법에 할당 할 수 있다.

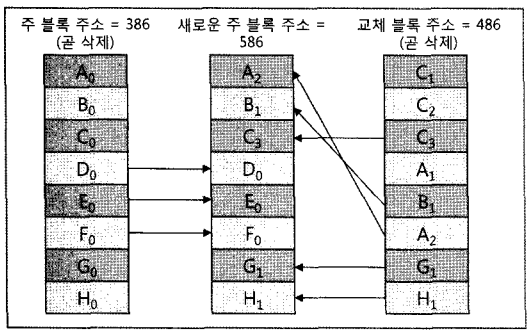


그림 10 CFTL의 블록 수거 기법

5. 성능 평가

본 장에서는 제시한 주소 성능 향상 기법을 적용한 CFTL의 성능을 비교한다. 그리고 2비트 쓰기 예측 기법이 플래시 메모리의 수명에 영향을 주는지를 실험한다. 또한 쓰기 시간에 관한 실험을 통해 블록 수거 기법의 오버헤드를 해결 할 수 있는지를 측정한다.

CFTL의 환경은 리눅스 2.6.17의 MTD(Memory Technology Device)를 사용하여 실험하였다[18]. 성능 평가를 위해 1차 페이지 단위 군집형 해시 테이블, 2차 페이지 단위 군집형 해시 테이블의 슬롯의 비율은 1:4로 할당하였다. 예를 들어 1차 페이지 단위 군집형 해시 테이블의 최대 버켓의 개수를 MSFS(Maximum Short Fine-grained Slot)라고 하며 2차 페이지 단위 군집형 해시 테이블의 최대 버켓의 개수를 MLFS(Maximum Long Fine-grained Slot)라고 한다. 이 두 개를 합친 페이지 단위 군집형 해시 테이블의 버켓의 총합을 MFS (Maximum Fine-grained Slot)이라고 칭한다. 본 실험에서 1차 페이지 단위 군집형 해시 테이블과 2차 페이지 단위 군집형 해시 테이블의 비율을 1:4로 설정했기 때문에 MSFS가 2,500개일 경우, MLFS의 개수는 10,000개를 갖는 것을 의미한다.

아울러 블록-페이지 테이블 간 승급을 위한 접근 빈도의 값을 여러 번 테스트한 결과, 임계값을 5로 설정하

는 경우 가장 좋은 결과를 나타내었다. 따라서 임계치는 5로 고정적인 값을 설정하였다.

또한 군집형 해시 테이블에는 한 버켓 당 몇 개의 서브블록을 포함하는지를 조절 할 수 있는 서브블록 팩터가 존재한다. 이는 서브블록이 증가하면 할수록 메모리 사용량이 감소되지만 성능 또한 저하되는 상쇄관계가 발생한다. 본 실험에서는 메모리 사용량이 효율적이면서 성능에서도 좋은 성능을 보였던 서브블록 팩터 값을 8로 설정하여 CFTL의 성능을 측정하였으며, MSFS는 2,500개, MLFS는 10,000개로 설정했다.

마지막으로 2비트 쓰기 예측 기법을 사용하기 위해서는 예측테이블의 크기 및 쓰기 버퍼의 크기에 따라 성능이 변할 것으로 예상되므로 예측 테이블의 슬롯 수는 각각 1024, 2048, 4096개로 설정하였다.

5.1 주소 변환 시간 분석

주소 변환 시간 및 CFTL의 전체적인 성능을 측정하기 위해 512MB USB 휴대용 저장장치의 입출력 내역을 3일간 리눅스 Block I/O Trace를 이용하여 추출 하였으며, 이를 이용하여 벤치마크 워크로드를 구성하였다.

본 USB 휴대용 저장장치 워크로드는 순차적인 페이지들의 집합이 약 54% 정도 되는 것으로 측정됐으며, USB 휴대용 저장장치 워크로드에서 선반입 적중률은 USB 휴대용 저장장치 워크로드 기준으로 약 67.51%로 나타났다. 적중률의 계산 할 때 한 버켓의 전체 페이지들을 선반입 했을 경우 한 개의 페이지만 적중하더라도 성공한 것으로 간주했다.

그림 11에서 보인 바와 같이 벤치마크 워크로드를 10회 반복하여 평균값을 낸 결과 순수 CFTL의 주소 변환 속도가 NFTL보다 약 13% 빠르고 AFTL보다 약 8% 빠른 것으로 나타났다.

연속성 카운터와 선반입 기법을 각각 적용한 경우, 연속성 카운터를 사용한 CFTL의 경우 NFTL보다 각각 약 15% 주소 변환 및 사상 기능이 향상되고 AFTL보다 각각 약 10% 이상 향상되었다. 그리고 선반입 기법을 사용한 CFTL의 경우 NFTL보다 각각 약 16% 주소 변환 및 사상 기능이 향상되고 AFTL보다 각각 약 9% 이상 향상되었다.

마지막으로 연속성 카운터와 선반입 기법을 동시에 사용하여 주소 변환 시간을 측정한 결과 NFTL보다 약 20% 그리고 AFTL보다 약 16% 이상의 주소 변환 시간 향상을 볼 수 있어 두 가지 기능이 합해졌을 경우 블록 단위 군집 해시 테이블과 페이지 단위 군집형 해시 테이블의 성능이 더욱 좋아짐을 알 수 있었다.

5.2 2비트 쓰기 예측을 이용한 비동기 쓰기 성능 측정

본 실험에서는 2비트 쓰기 예측 기법과 쓰기 버퍼를 이용하여 커널 수준의 페이지 캐시로 인한 불필요한 갱신 및 마모도 감소 방지에 주안점을 두었다.

그림 12는 본 USB 휴대용 저장장치에서 기존 데이타 블록의 외부 갱신 발생 입출력만을 따로 추출하여 만든 벤치 마크 워크로드를 이용하여 각 FTL의 쓰기 횟수 측정 결과를 나타낸다. 본 쓰기 횟수는 블록 수거로 인해 발생하는 쓰기 횟수를 포함한 값이다. 기존 데이타 블록의 외부 갱신으로 인한 쓰기 횟수 측정 결과 NFTL은 약 16,000번 그리고 AFTL은 저자가 실험 내용을 분석하고 주장한 대로 블록 수거를 지연 시키는 현상을 가져오기 때문에 쓰기 횟수가 15,000번 정도로 감소 한 것을 볼 수 있다. CFTL 또한 페이지 단위 군집형 해시 테이블에 데이타를 포함하고 있어 블록 수거 횟수를 감소시키는 결과를 가져 온 것과 함께 2비트 쓰기 예측 기법을 이용한 비동기 쓰기로 인해 예측 테이블 슬롯의 개수에 따라 쓰기 횟수가 각각 10,000~12,500번 정도로 감소 한 것을 볼 수 있다. 쓰기 횟수를 NFTL을 100% 기준으로 볼 때 AFTL은 약 6%, CFTL은 약 26%~30% 정도 쓰기 횟수를 감소시키는 것을 알 수 있다.

그림 13은 새로운 논리 블록에 데이타를 대량으로 썼을 경우를 예측 테이블에서 LBA를 VBA로 그룹화된 해당 데이타들이 2비트 쓰기 예측으로 인해 얼마나 쓰

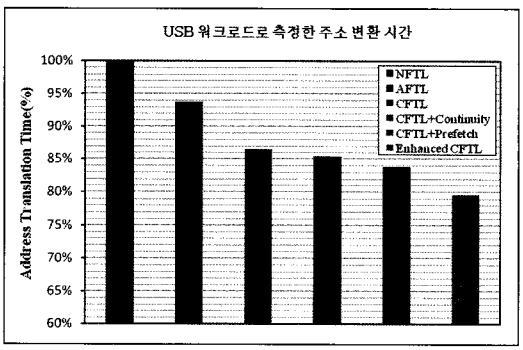


그림 11 각 FTL의 주소 변환 시간 비교

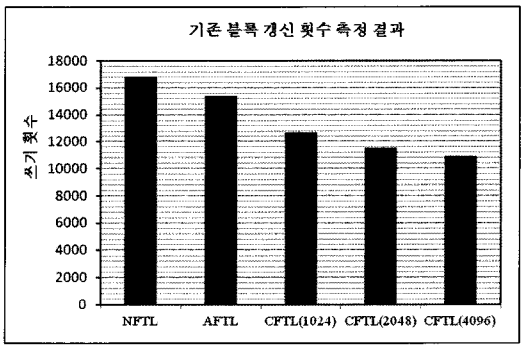


그림 12 외부 갱신으로 인한 기존 블록 갱신 횟수 측정 결과

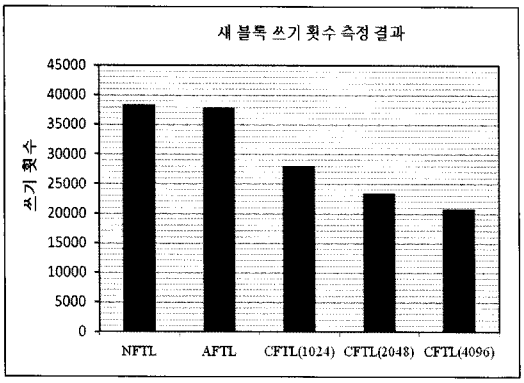


그림 13 새로운 논리 블록에 반복 쓰기 횟수 측정 결과

기 횟수가 얼마나 감소 될 수 있는지를 측정한 결과이다. NFTL은 총 약 38,500번의 쓰기 횟수가 있었으며, AFTL은 약 38,000번, 그리고 CFTL은 예측 테이블의 슬롯의 개수에 따라 각각 약 28,000, 약 23,000, 약 21,000번으로 나타났다. NFTL의 쓰기 횟수를 100%로 봤을 때 AFTL은 약 1%, CFTL은 예측 테이블 크기에 따라 각각 약 27%, 39%, 46% 정도의 쓰기 횟수 감소를 볼 수 있다. 2비트 쓰기 예측 기법을 사용하는 비동기 쓰기 기법은 해당 데이타를 블록 단위로 예측 테이블에 설정하여 쓰기 버퍼에 데이타를 우선적으로 쓰게 되어 쓰기 횟수를 감소시키는 것으로 보인다.

그림 14는 기존 NFTL, AFTL과 2비트 쓰기 예측 기법을 사용한 CFTL의 쓰기 시간을 각각 USB 플래시 메모리 워크로드, 기존 블록 외부 갱신, 새로 쓰기 테스트에 따라 측정한 것이다. 그 결과 CFTL이 NFTL, AFTL 보다 쓰기 시간이 USB 휴대용 플래시 메모리 워크로드에서 약 10% 이상, 기존 블록 외부 갱신, 새로 쓰기 벤치마크에서 최대 50% 이상 감소했음을 알 수 있다. 또한 단축된 결과는 블록 수거 기법의 오버헤드를 감소시키는데 사용할 수 있음을 증명한다.

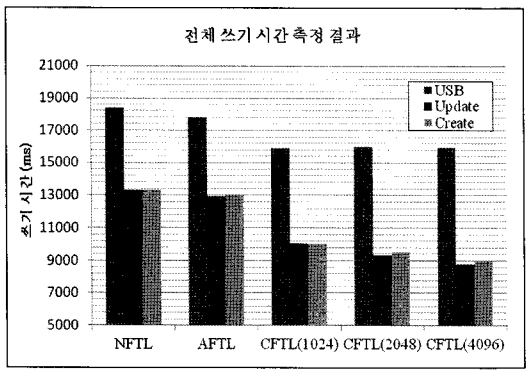


그림 14 각 FTL 별 쓰기 시간 측정 실험

5.3 비동기 쓰기로 인한 메모리 사용량 분석

표 3은 2비트 쓰기 예측 기법에서 사용하는 예측 테이블과 쓰기 버퍼의 메모리 사용량을 측정한 것이다. 예측 테이블의 슬롯의 개수가 각각 1024, 2048, 4096개인 경우 메모리 사용량이 8KB, 16KB, 32KB까지 증가하고 쓰기 버퍼가 사용하는 메모리 사용량은 각각 512KB, 1MB, 2MB으로 증가한다.

표 3 2비트 쓰기 예측 기법에 따른 메모리 사용량

슬롯 수	메모리 사용량	쓰기 버퍼 크기
1024	8KB	512KB
2048	16KB	1MB
4096	32KB	2MB

6. 결 론

플래시 메모리의 특징과 기존 파일시스템 간의 보완 역할을 해주는 FTL은 플래시 파일 시스템이 보편화되고 안정화되기 전까지 널리 쓰일 것으로 예상된다. 과거 FTL의 기능의 주요 기능인 주소 변환 및 사상 기능, 블록 수거 및 재활용 기능 외에도 플래시 메모리의 성능을 극대화 시킬 수 있는 다양한 연구가 필요하다. 본 논문에서는 이러한 새로운 시도로 CFTL의 주소 변환 및 사상 기법 외에 군집형 해시 테이블을 이용한 선반입 정책, 물리적으로 인접한 블록의 정보를 저장하는 연속성 카운터를 이용한 주소 변환 및 사상 기능을 최소화 시키는 방법, 리눅스 커널의 입출력 단위와 플래시 메모리의 페이지 단위를 맞추기 위한 2비트 쓰기 예측을 이용한 비동기 쓰기 방법을 제시했다. 주소 변환 시간에서는 기존 FTL에 비해 최대 17%의 성능 향상을 나타냈으며, 쓰기 횟수 및 성능에서는 약 30% 이상의 성능 향상을 이끌어 냈다.

또한 군집형 해시 테이블을 이용하여 버켓 단위로 페이지 데이타를 선반입하는 방법은 다른 FTL에 적용하기에는 다소 어려운 부분이 있지만, 2비트 쓰기 예측을 통한 리눅스 커널과의 입출력 단위 최적화 및 연속성 정보를 이용한 차기 주소에 대한 대응 기법은 다른 FTL에도 적용 할 수 있을 것으로 보인다.

참 고 문 헌

[1] Gartner Dataquest, "Forecast: Memory, Worldwide, 2001-2011(3Q07 Update)," Nov. 2007.

[2] E. Gal, S. Toledo, "Algorithm and Data Structure for Flash Memories," ACM Computing Surveys, Vol.37, Issue 2, pp. 138-163, 2005.

[3] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," IEEE Transaction on

Consumer Electronics, Vol.48, No.2, pp 366-375, 2002.
[4] S. W. Lee, W. K. Choi, D. J. Park, "FAST: An Efficient Flash Translation Layer for Flash Memory," EUC 2006 Workshops, Lecture Notes in Computer Science, No.4097 pp 879-887, Aug. 2006.
[5] K-H. Park and D-H. Kim, "A Clustered Flash Translation Layer for CompactFlash Systems," IEEE International Conference on Consumer Electronics, Las Vegas, NV, U. S. A., Jan. 2008.
[6] 이종민, 김성훈, 안성준, 이동희, 노삼혁, "NAND 플래시 메모리 저장장치에서 블록 재활용 기법의 비용 기반 최적화", 정보과학회논문지: 컴퓨팅의 실제 및 레터 제13권 제7호, pp. 508-519, 2007년 12월.
[7] Samsung Electronics, "K9K8G08U1A," Data Sheet of NAND Flash Memory.
[8] Samsung Electronics, "K9F1208U0C," Data Sheet of NAND Flash Memory.
[9] Intel Corporation, "Understanding the Flash Translation Layer (FTL) Specification," 1998.
[10] Intel Corporation, "Software Concerns of Implementing a Resident Flash Disk," 1995.
[11] Ban, A. 1995. Flash file system. US patent 5,404,485. Filed March 8, 1993; Issued April 4, 1995; Assigned to M-Systems.
[12] Ban, A. 1999. Flash file system optimized for page-mode flash technologies. US patent 5,937,425. Filed October 16, 1997; Issued August 10, 1999; Assigned to M-Systems.
[13] Ban, A. 2004. Wear leveling of static areas in flash memory. US patent 6,732,221. Filed June 1, 2001; Issued May 4, 2004; Assigned to M-Systems.
[14] C-H. Wu and T-W. Kuo, "An Adaptive Two-Level Management for the Flash Translation Layer in Embedded Systems," IEEE/ACM International Conference on Computer Aided Design, San Jose, CA, Nov. 2006.
[15] M. Talluri, M. D. Hill and Y. A. Khalidi, "A new Page Table for 64-bit address spaces," Proceedings of the 15th ACM Symposium on Operating Systems Principles, Dec 1995.
[16] John L. Hennessy and David A. Patterson, "Computer Architecture: A Quantitative Approach 3/E," Morgan Kaufmann Publishers.
[17] A. Silberschatz, P. B. Galvin, and G. Gagne, "Operating System Concepts, 7th Edition," John Wiley & Sons, Inc., pp. 315-326, 2005.
[18] MTD, "Memory Technology Device (MTD) subsystem for Linux," http://www.linux-mtd.infradead.org

박 광 희

2005년 용인대학교 컴퓨터정보처리학과 학사. 2004년 인프니스 기술연구소 연구원. 2004년~2005년 인테그라 정보통신 기업부설연구소 연구원. 2008년 인하대학교 전자공학과 석사. 2008년~현재 인하대학교 전자공학과 박사과정. 관심분야는 임베디드 시스템 소프트웨어, 하이브리드 저장장치, 차량용 소프트웨어 등

김 덕 환

1987년 서울대학교 계산통계학과 학사 1995년 한국과학기술원 정보통신공학과 석사. 2003년 한국과학기술원 정보통신공학과 박사. 1987년~1997년 LG전자 통신기기연구소 선임연구원. 1997년~2006년 동양공업전문대학 전산경영기술공학부 부교수. 2004년 University of Arizona, Tucson 박사후 연구원. 2006년~현재 인하대학교 전자공학과 부교수. 관심분야는 임베디드 시스템, 시스템 소프트웨어, 멀티미디어, 패턴인식, 데이타마이닝 등