플래시 메모리상에서 시스템 소프트웨어의 효율적인 버퍼 페이지 교체 기법

(An Efficient Buffer Page Replacement Strategy for System Software on Flash Memory)

박 종 민 † 박 동 주 ††

(Jong-Min Park) (Dong-Joo Park)

요 약 플래시 메모리는 오늘날 다양한 형태로 우리 생활의 일부를 차지하고 있다. 이동식 저장매체, 유비쿼터스 컴퓨팅 환경과 휴대전화기, MP3플레이어, 개인정보단말기(PDA) 등의 모바일 제품 등에 광범위하게 사용되고 있다. 이처럼 많은 분야에서 사용되는 주된 이유는 플래시 메모리가 저전력, 비휘발성, 고성능, 물리적 안정성, 휴대성 등의 장점을 갖기 때문이다. 더불어 최근에는 기가바이트급 플래시 메모리도 개발되어 하드디스크의 자리를 대체할 수 있는 상황에 이르렀다. 하지만, 플래시 메모리는 하드디스크와 달리 이미 데이타가 기록된 섹터에 대해 덮어쓰기가 되지 않는다는 특성을 갖고 있다. 데이타를 덮어쓰기 위해서는 해당 섹터가 포함된 블록을 지우고(소거) 쓰기 작업을 수행해야 한다. 이로 인해 플래시 메모리의 데이타 읽기/쓰기/소거에 비용이 하드 디스크와 같이 동일한 것이 아니라 각각 다르다[1][5][6]. 이러한 특성이 고려되지 않은 기존의 OS, DBMS 등과 같은 시스템 소프트웨어에서 사용되는 교체 기법은 플래시 메모리 상에서 비효율성을 가질 수 있다. 그러므로 플래시 메모리상에서는 플래시 메모리의 특성을 고려한 효율적인 버퍼 교체 기법이 필요하다. 본 논문에서는 플래시 메모리의 특성을 고려한 버퍼 페이지 교체기법을 제안하며, 제안된 기법과 기존 기법들과의 성능 평가를 수행한다. 지프분포와 실제 워크로드를 사용한 성능평가는 플래시 메모리의 특성을 고려한 버퍼 페이지 교체 기법의 필요성을 입증한다.

키워드 : 플래시 메모리, 버퍼 교체 기법

Abstract Flash memory has penetrated our life in various forms. For example, flash memory is important storage component of ubiquitous computing or mobile products such as cell phone, MP3 player, PDA, and portable storage kits. Behind of the wide acceptance as memory is many advantages of flash memory: for instances, low power consumption, nonvolatile, stability and portability. In addition to mentioned strengths, the recent development of gigabyte range capacity flash memory makes a careful prediction that the flash memory might replace some of storage area dominated by hard disks. In order to have overwriting function, one block must be erased before overwriting is performed. This difference results in the cost of reading, writing and erasing in flash memory[1][5][6]. Since this difference has not been considered in traditional buffer replacement technologies adopted in system software such as OS and DBMS, a new buffer replacement strategy becomes necessary. In this paper, a new buffer replacement strategy, reflecting difference I/O cost and applicable to flash memory, suggest and compares with other buffer replacement strategies using workloads as Zipfian distribution and real data.

Key words : flash memory, buffer replacement

1. 서 론

소비전력이 적고, 사용 중 전원공급이 중단되어도 저장된 정보가 사라지지 않는 비휘발성의 특성을 지닌 플래시 메모리는 하드디스크와 같은 기존의 디스크에 비해 데이타 접근 성능이 좋고 크기가 매우 작다. 또 물리적인 충격에 있어서도 하드디스크에 비해 강하고 무게

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음

† 학생회원 : 숭실대학교 컴퓨터학부
jmpark@computing.ssu.ac.kr
†† 정 회 원 : 숭실대학교 컴퓨터학부
djpark@computing.ssu.ac.kr

논문접수 : 2006년 1월 25일
심사완료 : 2006년 12월 11일

역시 가볍다는 장점을 갖고 있다. 이러한 특성을 가진 플래시 메모리는 최근 MP3 플레이어, 휴대 전화기, 개인정보단말기(PDA), 디지털 카메라/캠코더 등의 휴대용 정보기기들의 보조기억장치로 광범위하게 사용되기 시작했다[1].

최근 플래시 메모리의 사용 분야는 다양해지고, 다양한 형태로 우리 주위에서 사용되고 있다. 더불어 플래시 메모리의 저장 용량도 급속도로 대용량화 되어가는 추세이다. 최근 기가바이트급의 플래시 메모리가 개발되었고, 이러한 플래시 메모리의 대용량화 추세는 기존 하드디스크의 역할이 플래시 메모리로 대체 될 수 있음을 예견한다. 하드디스크의 역할을 대체한다는 것은 운영체제나 데이타베이스 관리 시스템과 같은 시스템 소프트웨어의 설치 및 사용이 플래시 메모리 상에서 이루어질 수 있음을 말한다. 일반적으로 운영체제나 데이타베이스 관리시스템과 같은 시스템 소프트웨어는 시스템의 메인 메모리 공간 일부를 메모리 버퍼(memory buffer) 영역으로 할당한다. 할당된 메모리 버퍼는 프로그램 실행 시 빈번하게 접근되는 디스크 페이지를 보관하여 동일 페이지 접근이 발생하면 메모리 버퍼에 보관된 페이지로 접근하여 디스크 접근에 필요한 I/O 비용을 줄일 수 있다. 메모리 버퍼에 여분의 저장 공간이 없다면 다른 페이지를 버퍼에 보관하기위해 기존 버퍼에 보관된 페이지 중 교체 대상 페이지(victim page)를 선정해야만 한다. 이때 어떠한 기준에 의해 교체 대상 페이지를 선정할 것인지에 따라 메모리 버퍼의 성능인 버퍼 적중률이 달라진다. 이때 사용되는 기준을 버퍼 페이지 교체 기법이라 말하며, 버퍼 교체 기법이 하드디스크가 아닌 플래시 메모리에서 사용하게 되는 경우 하드디스크와 다른 플래시 메모리의 특성을 고려해야 할 필요가 있다. 본 논문에서는 플래시 메모리의 특성을 고려한 버퍼 페이지 교체 기법을 제안하고, 기존 기법들과의 성능 평가를 통해 제안하는 기법의 효율성을 입증한다.

본 논문은 다음과 같은 구성을 갖는다. 2장에서는 하드디스크에서 사용되는 대표적인 페이지 교체 기법과 그러한 기법들이 플래시 메모리에서 사용될 경우 발생할 수 있는 문제점들과 플래시 메모리의 특성에 대해 설명한다. 3장에서는 본 논문이 제안하는 기법을 실험하는 시스템 구조와 메모리 주소 변환 계층(FTL - Flash Translation Layer)을 사용한 FLRU(Flash LRU) 기법에 대해 설명한다. 4장에서는 기존 하드 디스크에서 사용하는 기법들과 비교한 성능평가 결과를 분석하며, 5장에서 결론을 맺으며 마무리한다.

2. 관련연구 및 연구동기

그림 1은 시스템 상의 어플리케이션이 저장매체에 데

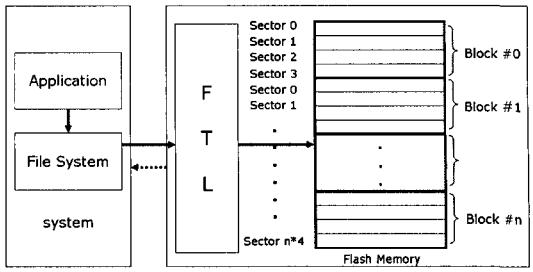


그림 1 블록 매핑과 플래시 메모리 구조

이타를 읽기/쓰기/소거 위해 연산을 명령했을 때, 명령이 파일시스템을 통해서 저장 매체로 전달되는 모습을 나타낸다. 파일 시스템으로부터 전달되는 명령은 읽기(데이타버퍼, LSN), 쓰기(데이타버퍼, LSN), 소거(LSN)과 같은 형태로 갖게 된다. 데이타버퍼는 읽기/쓰기의 데이타를 담는 곳이고, LSN(Logical Sector Number)은 플래시 메모리의 논리적 주소를 나타낸다. 실선 부분은 읽기/쓰기/소거 연산 명령의 흐름이고, 점선 부분은 플래시 메모리의 응답흐름이다.

실선 부분은 플래시 메모리의 구성요소인 플래시 메모리 변환 계층(이하 FTL - Flash Translation Layer)을 경유하게 된다. FTL은 논리적 주소 번지를 플래시 메모리의 물리적 주소 번지로 변화시켜 주는 역할을 한다. 지금까지 다양한 FTL이 개발되었으며, 각기 다른 장, 단점을 지니고 있다. 반대로 점선에서는 FTL을 거치지 않는데, 데이타버퍼 부분의 데이타만 시스템에 응답되면 되기 때문이다. 이렇게 사용되는 플래시 메모리의 구조에 대해 자세히 알아본다.

하드디스크에서 메모리 버퍼가 필요한 시스템 소프트웨어가 사용 될 경우 시스템의 메인 메모리의 일부를 메모리 버퍼로 사용한다. 메모리 버퍼에서 사용되는 페이지 교체 기법은 기존에 많이 존재하고, 그 중 대표적인 기법들로 LRU[2], LFU[2], MFU[2]가 있다. 이 기법들은 메모리 버퍼의 페이지 중 교체 대상 페이지를 선정하는데 다음과 같은 기준을 사용한다. 버퍼에 보관되는 시점, 보관되는 시간, 보관 후 참조되는 횟수를 기준으로 만들어진 것이다. 앞의 기준들이 고려된 이유는 버퍼 사용의 목적이 접근 빈도수가 높은 페이지를 버퍼에 보관하여 디스크 I/O 비용을 줄여 단위 시간당 프로세스 처리율을 높이기 위함이다. 메모리 버퍼에 보관된 시점, 시간, 참조 횟수만으로 교체 대상 페이지를 선정하는 주된 이유는 하드디스크의 물리적 특성 때문이다. 하드디스크의 읽기/쓰기/소거/덮어쓰기 연산을 수행하는데 사용되는 비용이 동일하기 때문이다. 즉, 특정 위치의 데이타를 읽기/쓰기/소거/덮어쓰기 연산에 사용되는 비용이 동일하다. 따라서 버퍼 영역에 보관된 페이지 중

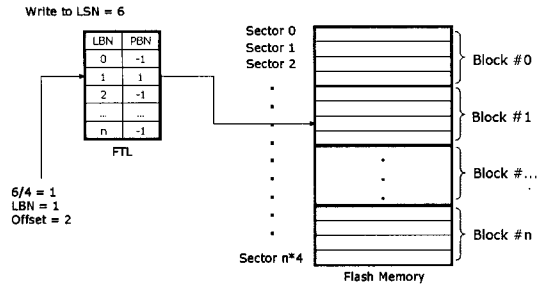


그림 2 블록 매핑과 플래시 메모리 구조

교체 대상 페이지를 선정할 때 연산의 종류는 무관하다는 뜻이 된다.

하지만 플래시 메모리는 물리적 구조와 특성이 하드디스크와 다르다. 플래시 메모리의 구조는 그림 2와 같다.

그림 2에서 플래시 메모리 부분은 최소 데이타 연산 단위인 섹터와 다수의 섹터(그림 2의 경우 4개)가 모인 블록들로 구성된다. 연산이 이루어지는 과정으로 설명하면 읽기/쓰기는 섹터단위로 수행하며, 소거 단위는 블록 단위로 수행을 한다. 그로인해 특정 데이타의 소거와 덮어쓰기 연산이 발생하는 두 가지 경우에는 반드시 소거가 수행된다. 소거는 블록 단위로 수행되므로 소거 대상이 아닌 섹터의 데이타는 소거되어선 안 되므로 여분의 블록(spare block)으로 소거 대상이 아닌 데이타를 복사한 후 원본 블록을 삭제한다. 덮어쓰기 연산은 하드디스크일 경우라면 섹터단위로 수행하겠지만 플래시 메모리의 경우는 물리적 특성으로 인해 불가능하다. 그래서 덮어쓰려는 데이타를 여분의 블록의 섹터에 쓰고, 원본 블록의 유효한 섹터들의 데이타를 복사해서 여분의 블록에 쓰기연산을 수행 후 원본 블록을 삭제한다. 이러한 플래시 메모리의 특성들로 인해 플래시 메모리는 읽기 연산의 비용을 1이라 가정하면 쓰기 연산의 경우는 읽기 연산의 약 7배의 비용을 가진다. 덮어쓰기가 발생할 경우에는 동일 페이지에 덮어쓰기가 이루어지지 않으므로 소거연산을 수행하게 되고, 이때 소거연산은 읽기 연산의 약 65배의 비용이다[1,5,6]. 따라서 하드디스크가 아닌 플래시 메모리에서 버퍼 페이지 교체 기법을 사용하려면 기존 기법들과 달리 페이지의 연산 비용을 고려해야한다. 기존 하드디스크의 메모리 버퍼에서 사용한 기법들을 플래시 메모리상에서 사용하게 되면 예상치 못한 비용을 초래할 수 있다. 예를 들어 교체 대상으로 선정된 페이지가 주로 덮어쓰기 연산을 갖는 페이지들로 이루어질 경우 비용은 증가하게 된다. 비용의 증가는 CPU 사용 증가를 야기하므로 이는 전체적인 시스템 성능에 영향을 미치게 된다.

따라서 플래시 메모리상의 버퍼 페이지 교체 기법은 기존 하드디스크에서 사용하던 페이지 교체 기법과는 달리 플래시 메모리의 특성과 버퍼 페이지의 교체 예상 비용을 고려한 새로운 버퍼 페이지 교체 기법이 필요하다.

3. FLRU(Flash Memory LRU) 기법: 플래시 메모리 버퍼 교체 기법

플래시 메모리 상에서 버퍼 영역을 사용하는 시스템의 구조는 그림 3과 같다. 크게 시스템 영역과 플래시 메모리 영역으로 구분이 되며, 시스템 영역에는 버퍼 관리자와 메모리 버퍼가 존재한다. 그리고 플래시 메모리에는 FTL과 플래시 메모리의 물리적 공간이 존재한다. 여기서 어플리케이션은 일반적으로 시스템에서 사용되는 어플리케이션 프로그램을 뜻하며, 프로그램 수행 중 액세스되는 데이타들을 메모리 버퍼 공간에 저장하여 I/O비용을 절감시킬 수 있도록 버퍼 교체 전략에 맞추어 메모리 버퍼 공간을 효율적으로 활용하는 역할을 한다. FTL은 플래시 메모리의 자체에 있는 소프트웨어 모듈로써 시스템 상에서 논리적으로 표현한 주소를 물리적 저장매체인 플래시 메모리로 사상시켜주는 역할을 한다.

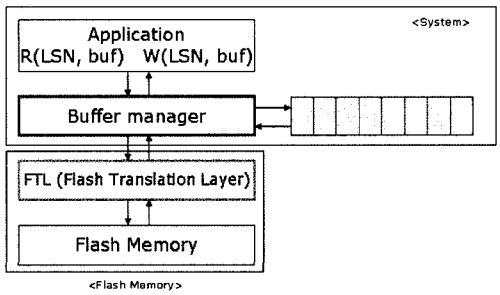


그림 3 시스템 구조

3.1 버퍼 관리자(Buffer Manager)

어플리케이션 프로그램은 읽기나 쓰기 명령을 버퍼 관리자에게 보내게 되고 버퍼 관리자는 읽기, 쓰기의 데이타가 현재 버퍼 영역에 저장되어있는지를 파악한다. 만약 버퍼 영역에 해당 페이지가 존재하지 않으면 버퍼 관리자는 FTL을 통해 플래시 메모리에 물리적 공간의 해당 페이지를 파악하여 어플리케이션 쪽으로 보냄과 동시에 버퍼 영역에 저장시킨다. 버퍼 관리자가 플래시 메모리의 물리적 공간에서 데이타를 읽어드려 버퍼 영역에 저장할 때에는 저장하는 시점과 FTL을 통해 예측된 ERC(Expected Replacement Cost - 버퍼 페이지를 플래시 메모리로 교체하는데 예상되는 비용)를 페이지마다 저장하게 된다. 저장된 정보는 버퍼 영역에 더 이상 페이지를 저장할 공간이 없게 되었을 때 교체 대상

페이지를 선정하는데 사용된다. 버퍼 영역에 새로운 페이지를 저장할 공간이 없을 경우에 버퍼 관리자는 버퍼 페이지의 저장 시점, 저장된 시간, 참조 횟수 ERC 값으로 교체 대상 페이지를 선정한다. 이와 같은 정보들로 교체대상 페이지를 선정하는 이유는 고전적 기법들이 단점을 가지고 있기 때문이다.

사용의 단순함과 우수한 이식성을 장점으로 가진 LRU 기법은 이러한 장점으로 인해 광범위하게 사용되고는 있다. 광범위하게 사용되는 만큼 LRU기법은 대부분의 패턴에서 우수한 성능을 가진다. 하지만 순차 혹은 순차 반복적인 참조 패턴을 가진 경우에는 LFU에 비하여 우수하지 못한 단점을 가지고 있다. MRU 기법은 순차 적인 패턴에 대해 우수함을 가지는 장점이 있지만 다양한 패턴에 대해서는 효율성이 떨어지는 단점을 가지고 있다. LFU는 반복적인 패턴에 대해 매우 우수함을 가지고 있는 장점이 있지만, 임의 패턴의 경우 많은 수의 반복이 있던 페이지들이 시간에 지나면서 참조되지 않게 될 때 참조횟수가 많기 때문에 버퍼 영역에서 자리를 차지하게 될 수 있다. 때문에 버퍼 영역의 활용도가 저하되는 단점을 갖는다. 각각 고전기법들의 장단점 중 장점 부분만을 FLRU에 적용하고자 앞서 말한 기준들을 FLRU에서 사용하고자 한다.

3.2 플래시 메모리 주소 변환 계층 - FTL(Flash Transition Layer)

플래시 메모리는 파일 시스템에서 명령하는 연산과 논리적 주소를 받아 물리적 플래시 메모리 공간에 해당 연산을 수행한다. 논리적 주소를 물리적 주소로 사상시키는 플래시 메모리의 구성요소가 바로 플래시 메모리 주소 변환 계층(이하 FTL)이다. 그림 2에서 FTL의 테이블을 보면 파일 시스템으로부터 6번 LSN에 쓰기 명령이 내려왔다는 것을 의미하며, LSN은 논리적 주소를 의미한다. 6번 섹터에 쓰기 명령이 내려오면, 블록 당 섹터수인 4로 나누어 몫을 논리적 블록(LBN - Logical Block Number)으로 나머지(offset)를 해당 블록의 저장 위치 섹터로 나타내어 준다. FTL을 블록 단위로 나타내는 이유는 섹터 단위로 구성되면 사용의 편의성은 있으나, FTL 테이블의 크기가 증가하게 된다. FTL은 플래시 메모리 안의 SRAM(Static Random Access Memory)에 저장되므로 크기가 증가하면 플래시 메모리의 생산원가 역시 상승하게 되므로 블록 단위로 FTL을 구성한다. 블록 단위로 구성된 FTL을 통해 LSN 6번을 쓰기 한 이후 6번 덮어쓰기가 이루어진다면 FTL을 통해 LSN 6번에 사용여부를 알 수 있고, 이를 통해 쓰기인지 덮어쓰기인지 판단할 수 있다. FLRU기법은 FTL을 통해 버퍼에 보관된 페이지들이 수행할 연산을 예상하여 FLRU 기법에 적용한다.

3.3 FLRU - 플래시 메모리 버퍼 페이지 교체 기법

FTL을 통해 얻은 버퍼 페이지의 교체 예상 비용을 ERC(Expected Replacement Cost)라 표현하며 버퍼에 보관된 페이지들의 가중치를 판단하는 요소로 사용한다. FLRU 기법에서 사용하는 요소들로는 2장에서 언급했던 기법들과 마찬가지로 버퍼 페이지의 참조된 시간, 참조 횟수를 사용하고, 버퍼에 보관된 페이지들의 ERC를 추가한다. 버퍼에 보관된 페이지의 ERC를 얻어내기 위해선 플래시 메모리의 FTL을 통해 플래시 메모리의 물리적 공간의 상태를 파악하여, 읽기의 경우는 1, 쓰기의 경우는 7, 덮어쓰기의 경우는 65 중 결정된다. (이때, FTL을 통한 상태 판단 비용은 플래시 메모리의 S-RAM에 저장 되므로 I/O비용으로 간주하지 않는다).

$$C = \left(\frac{t_c - t_{flavg}}{f_i \times ERC} \right)$$

수식 1. FLRU 기법

수식 1은 FLRU 기법을 수식으로 표현한 것이다. 먼저 ERC는 버퍼에 보관된 페이지가 교체 대상으로 선정되었을 때 해당 페이지가 쓰일 섹터의 상태에 따라 크기가 달라진다. 메모리 버퍼에 읽기 연산으로 보관된 페이지가 교체 대상 페이지로 선정되면 교체 비용은 발생하지 않는다. 하지만 쓰기 상태면 플래시 메모리의 상태에 따라 쓰기/덮어쓰기 두 경우가 발생 할 수 있다. 앞서 설명했듯이 쓰기일 경우는 읽기의 7배의 비용이 발생하므로 ERC값은 8에 상태를 추적하는데 발생하는 비용을 더한 값이 된다. 덮어쓰기 경우에는 ERC의 값 65에 상태 추적하는데 발생하는 비용을 더해준다. 분모 부분은 ERC값과 버퍼 페이지 참조 횟수(f_i)를 곱해 각 페이지의 가중치를 갖게 된다.

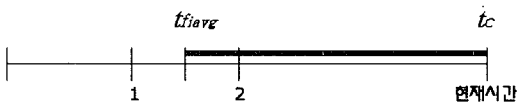


그림 4 버퍼 페이지 시간 개념도

그림 4는 특정 페이지가 시점 1일 때 처음 버퍼에 보관되고 시점 2일 때 참조가 발생했을 경우 두 번의 참조 시간의 평균 시점(t_{flavg})과 현재 시점(t_c)의 차이를 실선으로 표현 한 것이다. 수식 1에서 분자 부분에 해당하는 값(실선)은 LRU 기법과 유사함을 알 수 있다. FLRU기법은 우수한 버퍼 교체 기법으로 알려진 LRU 기법을 기반으로 하여 분자부분이 구성되었고, 분모 부분을 플래시 메모리 특성을 고려한 ERC값을 사용하여 구성하였다. 다시 말해 LRU기법으로 나온 값을 플래시

메모리 특성을 고려한 가중치로 나누어 값 C를 얻는다. 이렇게 얻은 C 값이 교체 대상 페이지 선정 여부를 가리는 값이 된다. 다시 말해 C 값이 큰 페이지를 교체 대상으로 선정한다. FLRU 기법의 가장 큰 변수가 되는 *ERC*값은 웹 캐싱 기법의 대다수가 중요 변수로 웹 문서의 크기를 적용하는 방식에서 착안했다[4]. Hyper-G[4]라는 웹 캐싱 기법은 웹 오브젝트를 캐싱하는데 있어서 LRU, LFU, SIZE의 요소들로 캐싱할 오브젝트를 선정한다. FLRU는 이 기법의 SIZE 요소를 사용하듯이 예상비용 *ERC*를 사용한다.

FLRU 기법에서는 페이지가 플래시 메모리에 저장되는데 사용되는 예상 비용을 *ERC*로 측정하고, *ERC*값을 적용한 후 FLRU 기법을 통해 구해지는 C 값은 버퍼에 보관된 모든 페이지 마다 존재하게 된다. 새로운 버퍼 페이지가 필요한 시점에서 보관된 모든 페이지 중 C 값이 가장 큰 페이지가 교체 대상 페이지로 선정된다. 교체 대상 페이지가 플래시 메모리에 저장되는데 사용되는 비용은 *ERC*가 된다.

4. 성능평가

4.1 지프 분포 워크로드

성능평가를 위해 사용된 시스템은 CPU PentiumⅣ 2기가헤르츠, 512메가 메인 메모리, 120기가의 하드 디스크를 가진 Linux 시스템이었고, 2장에서 언급한 페이지 교체 기법들과 FLRU의 성능 비교 평가를 위해 사용한 FTL은 미쓰비시사의 FTL[7]을 C언어로 구현하여 사용했다. 이전 FTL은 동일 논리적 번지에 쓰기 연산을 수행하고 덮어쓰기가 일어나면 반드시 블록 교체와 소거 연산이 일어나며 비용 역시 증가한다. 미쓰비시 FTL은 덮어쓰기 시 발생하는 블록 소거 연산을 지연시키기 위해 미리 지정한 수의 스페어 섹터(spare sector)에 덮어쓰기 데이타를 입력한다. 그래서 종전의 덮어쓰기로 인해 발생했던 블록 소거를 지연시키므로 소거 연산의 비용도 스페어 섹터 수만큼 줄일 수 있다. 하지만 블록마다 지정해 둔 스페어 섹터를 유지하기 때문에 플래시 메모리의 공간 활용률이 떨어진다. 이러한 문제점은 본 실험의 결과에 크게 작용하지 않는 부분이다.

성능 평가에서 사용된 워크로드는 지프 분포(Zipfian distribution)[3]를 통해 실험 공간의 최대 주소번지를 감안한 0부터 5000사이의 수 1만개를 생성했다. 지프 분포는 지프 값에 따라 달라지며, 0.1, 0.5, 0.9 세 가지의 지프 값을 사용했다. 0.1일 경우는 생성되는 수가 균등하게 분포된다. 다시 말해 0부터 5000사이 대부분의 수가 1회 이상 나타난다. 이와 반대로 0.9일 경우는 0부터~5000사이 일부분의 수가 50% 이상 나타난다. 두 경우 중간 수준으로 반복 되는 수를 0.5에서 볼 수 있다. 이렇게 지프 분포에 의해 생성된 수들을 연산의 입력으로 사용되는 논리적 섹터 번지(LSN)로 사용했다. 플래시 메모리는 연산에 따라 비용이 다르기 때문에 읽기/쓰기 연산을 나타내는 구분자가 필요하여 이 역시 지프 값으로 생성하여 1만개의 LSN에 표시 후 실험했다. 다양한 버퍼 페이지를 설정 후 측정한 값은 교체 대상으로 선정된 페이지들의 교체 예상 비용 ERC의 누적 값과 버퍼 페이지 적중 수를 측정했다.

그림 5는 지프 값 0.5의 실험 결과 중 예상 페이지 교체 비용과 버퍼 적중 횟수를 보인다. 128개 이하의 버퍼 페이지 수를 가지는 경우에는 기법들의 성능 차이가 크지 않았다. 이러한 결과는 워크로드 패턴 중 중복 LSN이 버퍼 페이지의 수 보다 큰 간격으로 나타나기 때문이다. 그러나 중복 LSN을 포함할 수 있는 버퍼 페이지 수가 128개 이상이 되면서부터는 FLRU가 최소의 페이지 교체 비용을 가지는 것을 알 수 있다.

지프 값 0.5일 경우 버퍼 적중 횟수를 보게 되면, 모든 버퍼 페이지에서 FLRU가 높은 버퍼 적중 횟수를 보인다. 증가율로 보면 버퍼 페이지 256개의 경우 FLRU가 두 번째 높은 LFU에 비해 약 21% 증가했다. 지프 값 0.5일 경우에서 예상 페이지 교체 비용과 버퍼 적중 횟수에서 다른 기법들에 비해 FLRU가 우수함을

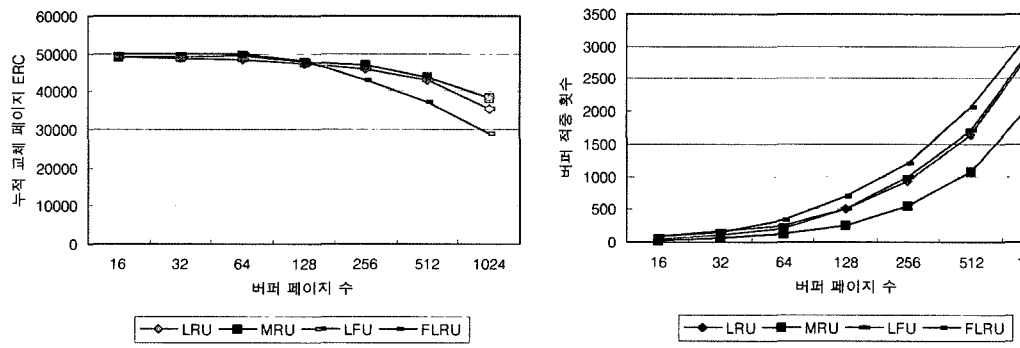


그림 5 지프 값 0.5일 경우 예상 페이지 교체 비용과 버퍼 적중 횟수

알 수 있다.

그림에는 없지만 지프 값이 0.1인 경우의 결과를 보게 되면 전체적으로 비슷한 수준으로 교체 비용과 버퍼 적중 횟수를 보여준다. 이러한 이유는 지프 값 0.1의 워크로드가 균등하게 존재하므로 다양한 수의 LSN이 적은 수의 반복을 갖는다. 그러므로 버퍼의 활용도가 떨어져 실험대상의 모든 기법들의 결과가 비슷한 수준을 보여준다.

지프 값 0.9인 경우는 0.1과 0.5에 비해 일부 LSN이 많은 수의 중복 횟수를 갖는다. 그림 6은 지프 값 0.9의 페이지 교체 예상 비용과 버퍼 적중 횟수를 나타낸 그림이다. 0.1과 0.5의 결과에 비해 전체적으로 누적 교체 페이지 ERC가 줄어들었음을 알 수 있다. 버퍼 페이지가 256개의 경우 FLRU가 LRU의 약 71%, LFU의 약 76% 수준으로 페이지 교체 비용이 감소함을 보여준다. 지프 값 0.9의 버퍼 적중 횟수는 128개의 버퍼 페이지일 경우 LFU의 약 13% 증가한다. 버퍼 페이지 수 1024개의 경우는 워크로드의 전체 LSN 중 특정수들이 빈번하게 중복되기 때문에 MRU가 아닌 다른 기법들의 결과가 비슷하게 나타난다. 하지만 그 이하의 경우는 거의 모든 경우 모든 교체 기법보다 향상됨을 보인다.

높은 지프 값에 따른 분포의 경우는 FLRU와 LFU가 높은 페이지 적중횟수를 보이는데 이는 제한된 버퍼 페이지 수와 특정 LSN의 잦은 반복으로 인해 많은 적중수를 보이게 되었다는 것을 알려주는 것이다. 실제 데이타에서는 버퍼 적중 횟수는 고려하지 않고, 누적 교체 페이지에 관한 부분으로 실험하였다.

4.2 실제 데이타 워크로드

지프 분포를 사용하여 인위적으로 만든 워크로드가 아닌 실제 OS상에서 버퍼 메모리를 사용하는 어플리케이션들의 참조 패턴을 워크로드로 사용한 실험의 결과이다. 워크로드는 다음과 같은 구성을 갖는다.

- postgres : UC 버클리에서 개발한 RDBMS
- gnuplot : GNU 플로팅 프로그램
- cscope : C 소스코드 실행 도구
- glimpse : 텍스트 정보 검색 도구
- cpp : C 소스코드 컴파일러

위에 열거한 어플리케이션들은 실제 FreeBSD 운영체제 상에서 사용되는 어플리케이션들이다. 이들 어플리케이션 프로그램들이 사용되면서 참조 데이타에 접근하는 패턴들을 실험의 워크로드로 사용할 것이다. 앞서 실험에 사용했던 지프 분포 워크로드들은 순차적 혹은 반복적이지 않은 균등한 형태였지만, 이번 실험에 사용할 워크로드들은 순차적이면서 반복적인 형태이다. 일반적으로 LRU 기법은 단순성과 이식의 편리함으로 인해 광범위하게 사용되지만, 규칙적으로 반복되는 패턴에 대해서는 좋지 않은 형태를 가지게 된다[8]. 본 논문에서 제안하는 FLRU 기법 역시 LRU의 기본 형태에 플래시 메모리의 특성을 고려한 형태의 기법이기 때문에 어떠한 결과가 나올지 실험을 통해 알아본다.

그림 7은 RDBMS인 postgres 워크로드를 사용했을 때의 교체 페이지의 ERC 값을 누적 시킨 그래프이다. 대부분의 기법이 비슷한 수치의 ERC 값을 보이며 버퍼 페이지 수 128개와 512개 사이 일 때 FLRU의 성능이 우수함을 알 수 있다. gnuplot과 cscope 역시 비슷한 수치를 보이며 별다른 차이를 갖지 않았다. 세 가지 워

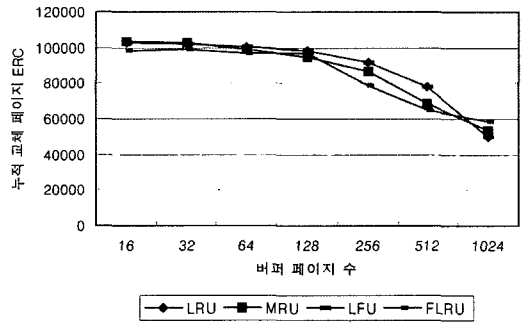
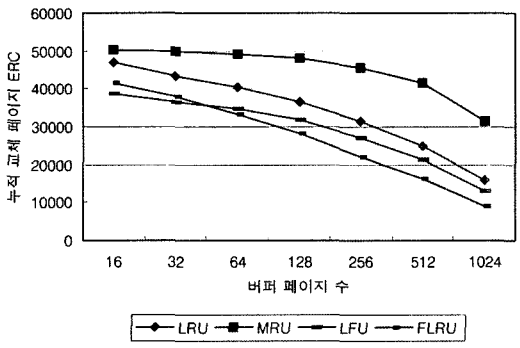


그림 7 postgres 워크로드



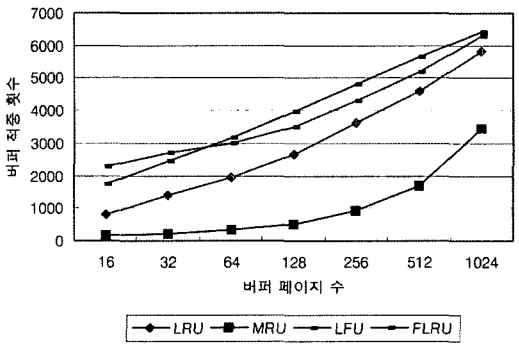


그림 6 지프 값 0.9일 경우 예상 페이지 교체 비용과 버퍼 적중 횟수

크로드의 공통점은 순차적이며 반복적인 형태를 가지며 그 외의 데이타 접근의 빈도수가 매우 적은 형태를 갖는다.

그림 8은 postgres, gnuplot, cscope의 워크로드를 그래프로 나타낸 것이다. x축은 워크로드가 나타나는 시간이고, y축은 시간에 따른 접근 번지(LSN)이 된다. 순차 반복적인 패턴을 중심으로 그 외의 번지 접근의 빈도수는 높지 않다. 실험을 통해 나타난 결과는 FLRU와 LFU 혹은 MRU의 성능이 비슷한 수치로 나타났다.

그림 9의 glimpse와 cpp 워크로드는 앞의 세 워크로드와 달리 더욱 다양한 참조번지를 가진다. 지프 분포를 사용한 워크로드도 특정번지에 집중되는 형태가 아닌 다양한 번지에 균등하게 분포함을 보여준다. postgres, gnuplot, cscope 워크로드에 비해 glimpse와 cpp가 좀 더 균등한 형태를 가짐을 알 수 있다. 접근 번지가 균등하게 분포되면 될수록 FLRU의 성능이 우수하게 나타났다. 실제 사용되는 어플리케이션에서의 워크로드 형태는 순차적, 반복적, 임의적 등의 형태로 나타날 수 있다.

그림 10은 glimpse와 cpp 워크로드를 사용했을 때의 누적 교체페이지 ERC를 측정한 결과를 보여준다. 그림 9에서 상대적으로 더 균등한 cpp의 결과를 보게 되면 FLRU기법이 다른 기법들에 비해 월등히 좋은 성능을

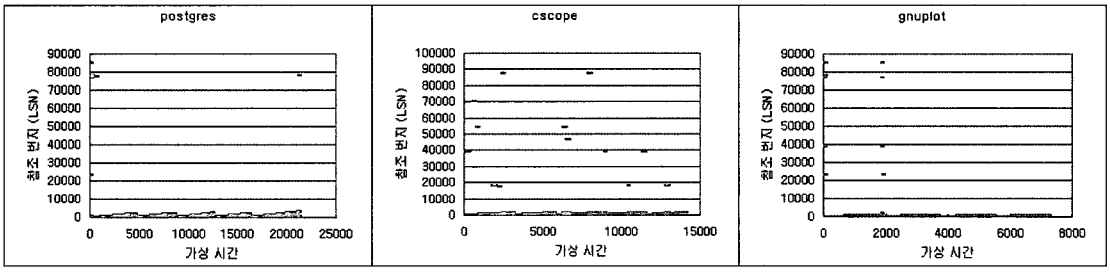


그림 8 postgres, gnuplot, cscope 워크로드 형태

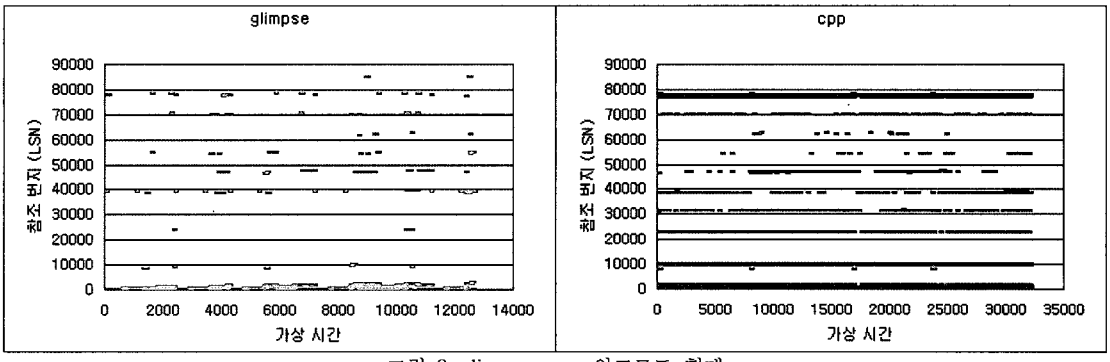


그림 9 glimpse, cpp 워크로드 형태

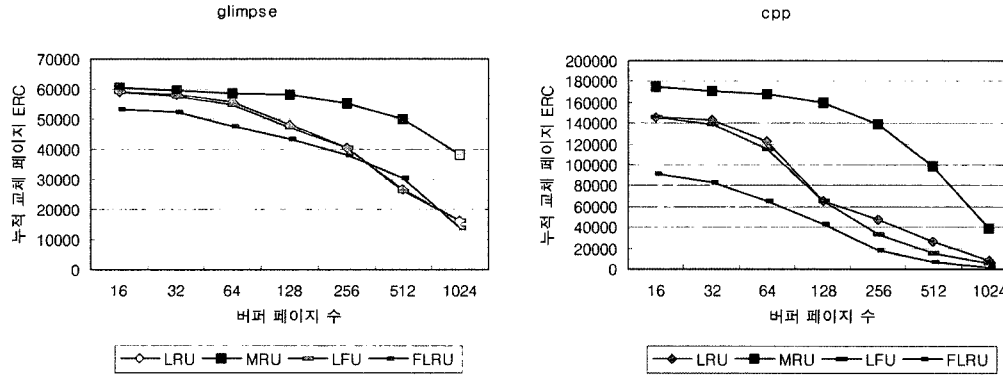


그림 10 glimpse, cpp 워크로드

보여준다. 이는 균등하게 나타나는 워크로드의 경우 FLRU기법이 우수하게 버퍼 교체 기법을 수행함을 보여주는 결과가 된다.

실험을 통해 얻을 수 있는 가장 큰 결론은 플래시 메모리 상에서 버퍼 영역을 사용할 때 기존 기법이 아닌 플래시 메모리의 특성을 고려하는 방법이 필요하다는 것이고, FLRU는 그 부분을 고려했기 때문에 다른 기법들과 차별화를 가질 수 있는 점이다. 또 다양한 워크로드들을 대상으로 실험했기 때문에 FLRU가 우수한 경우와 순차 반복적인 워크로드의 경우와 같은 크게 차이나지 않는 부분도 파악할 수 있었다. 이를 통해 기존에 제안되었던 UBM[8]과 같은 형태로 패턴을 파악하여 버퍼 교체 기법을 결정하는 기법과 유사한 형태로 활용할 수 있다.

5. 결 론

기존 하드디스크에서 사용되었던 버퍼 교체 기법들과 FLRU 기법을 플래시 메모리상에서 비교해 보고 동일 워크로드를 수행하는데 소요되는 비용을 측정했을 때 거의 모든 워크로드에서 FLRU가 가장 적은 비용을 나타냈다. 비용이 적다는 것은 시스템의 CPU가 처리해야 할 프로세스가 줄어들고, 단위시간당 업무 프로세스 수의 증가를 나타낸다. 그와 반대로 페이지 교체 비용을 고려하지 않고, 메모리 버퍼를 사용한다면 메모리 버퍼의 사용은 본래 목적을 벗어나게 되며, 전체적인 시스템 성능 또한 저하된다.

본 논문에서는 실험을 통해 플래시 메모리 상에서의 버퍼 페이지 교체 기법은 플래시 메모리 특성에 따라 각기 다른 연산비용을 고려해야 한다는 것을 입증했다. 향후에는 플래시 메모리의 특징을 고려하는 다양한 기법들이 연구되어야 할 것이다. 예를 들어 버퍼 페이지 교체 기법을 사용함에 있어서 플래시 메모리의 데이타 접근 시간을 측정하고, 그 시간이 덮어쓰기 연산이나 쓰기 연산 시간에 비해 작다면 버퍼 영역에는 읽기 연산에 사용되는 페이지는 버퍼에 보관하지 않는 것이 좋을 수 있다. 이러한 점을 고려한 교체 기법도 필요하다.

최근 기가바이트급 플래시 메모리가 개발된 후 플래시 메모리의 용량은 점점 커질 것이고, 자연스럽게 하드디스크에 비해 많은 장점을 갖는 플래시 메모리가 하드디스크를 대체할 것을 예상하여 앞으로 활발한 연구가 진행되어야 한다.

참 고 문 헌

[1] Tae-Sun Chung, Dong-Joo Park, Yeonseung Ryu, Sugwon Hong, "LSTAFF: System Software for Large Block Flash Memory," Asia Simulation Conference, pp. 704-712. 2004.

[2] A. Silberschatz, P. B. Galvin, and G. Gagne, "Operating System Concepts sixth edition," Wiley, 2003.

[3] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K Page Replacement Algorithm For Database Disk Buffering," ACM Special Interest Group on Management Of Data 1993 Annual Conference, pp. 297-306, 1993.

[4] S. Podlipnig, L. Böszörményi, "A Survey of Web cache replacement strategies," ACM Comput. Surv., Vol. 35 No. 4, pp. 374-398, 2003.

[5] J.Kim, J.M. Kim, S.H. Noh, S.L. 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, 2002.

[6] Chin-Hsien Wu, Li-Pin Chang, and Tei-Wei Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems," Real-Time Computing Systems and Applications, pp. 409-430, 2003.

[7] Takayuki Shinohara, "Flash Memory Card with Bock Memory Address Arrangement," U.S. Patent, Patent Number 5,905,993, May 18, 1999.

[8] Jong Min Kim, Jongmoo Choi, Jesung Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, Chong Sang Kim, "A Low-Overhead High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References," Operating Systems Design and Implementation, pp. 119-134, 2000.

박 종 민

2002년 경원대학교 전자계산과(학사). 2006년 숭실대학교 대학원 컴퓨터학과(석사) 2006년~현재 로지스을 유로지스넷(주) 관심분야는 데이타베이스, 플래시 메모리, 유비쿼터스 컴퓨팅

박 동 주

1995년 서울대학교 컴퓨터공학과(학사) 1997년 서울대학교 컴퓨터공학과(석사) 2001년 서울대학교 컴퓨터공학부(박사) 2001년~2004년 삼성전자 책임연구원. 2004년~현재 숭실대학교 컴퓨터학부 교수 관심분야는 데이타베이스, 멀티미디어 시스템, 플래시 메모리, 내장형 소프트웨어