DBpia

논문 99-24-5A-7 한국통신학회논문지 '99-5 Vol.24 No.5A

이동컴퓨터를 위한 플래시메모리 클리닝 정책

정회원 민 용 기*, 박 승 규*

Cleaning Policies in Flash Memory for Mobile Computers

Yong-ki Min*, Seung-kyu Park* *Regular Members*

요 약

최근 이동 통신기술의 발달로 시간과 장소를 가리지 않고 정보에 접근하는 이동 컴퓨터가 각광을 받고 있다. 이동 컴퓨터는 크기와 무게 등의 물리적인 한계 때문에 낮은 전력소모, 경량, 견고함 등을 요구한다. 따라서, 기존의 디스크 드라이브, DRAM, SRAM, ROM등의 저장매체보다는 비휘발성, 고집적, 빠른 읽기 접근시간의 특징을 갖는 플래시메모리(Flash memory)가 각광받고 있다. 그러나 플래시메모리는 느린 쓰기 접근시간(write access time)과 세그먼트 단위의 소거작업(erase operation)으로 제자리 쓰기가 불가능하고, 또한 소거작업의 횟수에 제한이 사용 수명에도 한계가 있다. 본 논문에서는 이러한 단점을 보완하기 위해 수정된 방향성 클리닝 정책을 제시하였다. 이 방식은 클리닝 작업의 횟수를 줄여 사용 수명을 늘리고 또한 전반적인 메모리 접근 시간을 개선하였다. 시뮬레이션을 수행하여 수정된 방향성 클리닝 정책이 기존의 여러 방식보다 더 좋은 성능을 나타냄을 보였다.

ABSTRACT

Recent studies in mobile computing technology aim to support ubiquitous communications by reducing physical sizes and power consumptions. Flash memories are one of best candidates to meet such requirements. The features of nonvolatility, low power consumption, and fast access time for read operations are sufficient grounds to support flash memory as major components of mobile systems. The special memory management, however, should be taken to fully take advantage of those features. The write operation to a word must follow the cleaning operation of a whole block including the word. And the number of cleanings is limited. This paper presents efficient cleaning policies of flash memory blocks which reduce the number of block cleanings and approximate to equal life time of blocks. The simulated results show that the proposed policies have good performances.

I. 서 론

최근 이동 통신, 무선 랜, 위성통신 기술의 발달로 시간과 장소를 가리지 않고 정보를 교환할 수 있는 기술이 급속히 발달하고 있다. 아직은 노트북이나 PDAs(personal digital assistants) 혹은 HPC (Hand-held PC)등이 사용되지만 이는 네트웍 기능과 성능이 보완되어야할 부분이 많다. 가까운 미래에는 수많은 사람들이 보다 강력한 이동 컴퓨터를 사용하게 되어 현재보다 많고 다양한 분야에 새롭게 응용될 것이다. 이러한 이동 컴퓨터는 휴대성, 무선 데이터 통신, 소비전력, 사용자 인터페이스, 보안, 위치정보 이용 등의 사항이 중요하게 다루어졌다.

이동 컴퓨터는 개인의 휴대가 가능하여야 하므로 그 크기가 작고 무게도 가벼워야 한다. 이러한 약점 때문에 크기, 소모전력, 내구성 등에서 기존의 컴퓨터와는 다른 제약을 가진다[2,4,9]. 따라서 디스플레이, 입력장치, 저장장치는 그 크기, 무게, 소비전력에 따라 선택의 폭이 좁아질 수밖에 없다. 특히 소비전력

* 아주대학교 정보 및 컴퓨터공학부(Sparky@madang.ajou.ac.kr)
논문번호 : 98516-1127, 접수일자 : 1998년 11월 27일
※ 본 연구는 한국과학재단 특정기초연구비(961-0100-001-2)지원으로 수행되었으며 지원에 감사드립니다.

www.dbpia.co.kr

은 이동 컴퓨터에서 매우 중요시된다. 이 소비전력은 모든 기기에 영향을 줄뿐만 아니라 개인이 장시간 휴대를 가능하게 하는 중요한 변수로 작용된다. 이런 이유 이와 같은 요구를 충족하기 위한 기술연구가 많이 이루어져 왔다. 첫 번째로는 디스크의 사용을 억제하는 연구가 있는데, 디스크가 사용되지 않는 동안 회전을 중지시켜 소비전력을 줄이는 방법이다. 두 번째로는 CPU의 사용을 줄이는 방법이 있을 수 있다. 사용하지 않는 동안 사용을 줄여 소비전력을 줄일 수 있다. 세 번째로는 통신에 관련하여 통신에 소모되는 전력을 줄이는 방법이 있다. 수신보다 송신에 전력소모가 더 많은 점을 이용하여 소모전력을 줄인다. 그리고 네 번째로 소비전력이 많은 장치를 대치하기도 한다. 소비전력이 많은 디스크 대신에 적은 플래시메모리를 사용해서 소비전력을 줄인다[1,4].

플래시메모리를 저장 매체로 사용하는 연구는 eNVy시스템이 있다[10]. eNVy는 플래시메모리를 일반적인 워크스테이션 디스크 드라이브의 대체 장치로 하여 사용하였다. 플래시메모리를 매우 많은 수를 사용하여 2기가 바이트의 메모리를 구성하였다. 플래시메모리의 특성으로 인해 copy-on-write구조를 가지고 있어 log-structured file system과 유사하다[7]. 그리고 세그먼트 클리닝 시에 많은 연산을 필요로 하여 MMU(Memory Management Unit)를 따로 가지고 있다. 쓰기가 느린 것을 개선하기 위해 SRAM을 쓰기 버퍼(write buffer)로 쓴다. 이는 이동 컴퓨터에 쓰기에는 크기가 너무 크고 MMU를 사용하는 것에도 무리가 있다. 또한 클리닝 정책에서 매우 많은 연산을 필요로 하기 때문에 이동 컴퓨팅에 적합하지 않다. 세그먼트를 클리닝 할 때마다 클리닝 비용과 클리닝 빈도를 곱하고 이를 전체 세그먼트의 평균과 비교해야하므로 매우 많은 연산을 할 수밖에 없고 따로 MMU를 가지기 어려운 이동 컴퓨터에서는 적용시키기 어렵다.

[3]에서는 이동컴퓨터에서 디스크 드라이브를 쓸 때 전력소모를 줄이기 위해 플래시메모리를 캐쉬로 이용하였다. 사용자가 디스크 드라이브를 사용하지 않을 때 디스크의 회전을 중지시켜 전력을 회전에 필요한 전력을 아끼도록 하고 사용자가 디스크를 다시 쓰려고 할 때 디스크의 회전을 다시 작동시키는데, 이때 디스크 드라이브가 완전히 동작하기 전까지 작업을 플래시메모리가 대신하도록 하였다. 클리닝 작업은 단순히 FIFO와 LRU를 사용하였다. 클리닝 작업보다는 전력의 소모와 응답시간을 주요하게 다루었다.

Ⅱ. 플래시메모리의 특성

1. 성능 및 액세스 특성

플래시메모리는 그 구조가 간단하여서 집적도가 높다. 그리고 DRAM과 비슷한 읽기 성능을 나타낸다. 읽기(read)가 16ns - 85ns정도이고, 쓰기(write)가 4-10 micro second정도의 시간이 소요된다[11]. 전력소모의 경우 DRAM이 동작상태인 경우 약 수백 mA인데 반해 플래시메모리는 동작상태에서 30-50 mA로 전력소모가 적다. 그리고 기존 메모리와 비교해 큰 장점으로는 전원을 제거해도 데이터가 지워지지 않는 비휘발성을 가지고 있는 점이다. 또한 디스크와 같이 일단 램으로 옮겨 프로그램을 실행하지 않고 플래시메모리에 저장되어 있는 프로그램을 바로 실행시킬 수 있는 제자리 실행(execute in-place)도 가능하여 메모리의 소비와 메모리에 옮기는 시간을 절약할 수 있다. 비휘발성은 데이터의 보존과 전력소모가 적다는 점에서 이동 컴퓨터에 적합하다. 이런 장점으로 앞으로 이동컴퓨터 분야에 적용될 가능성이 크다. 그러나 플래시메모리는 데이터의 덮어쓰기가 불가능하다. 즉 같은 주소에 수정된 내용을 바로 다시 쓰는 것이 불가능하다. 만약 같은 자리에 다시 쓰고자 한다면 그 블록이 포함된 세그먼트에 소거작업(erase operation)을 수행한 후 다시 쓸 수 있다. 따라서 수정된 블록은 쓰기 가능한 블록이 있는 세그먼트에 쓰여져야 한다. 여기서 세그먼트는 여러 개의 블록들이 모여 하나의 세그먼트를 이루게 되는데 플래시메모리의 종류에 따라 그 크기가 8KB, 16KB, 64KB, 128KB등으로 다양하다. 그리고 칩의 종류에 따라 블록킹(blocking)기술을 이용하여 대칭 블록킹(symmetric blocking), 비대칭 블록킹(asymmetric blocking)으로 나뉘어져 소거 세그먼트의 크기를 균등하게 하는 경우와 비균등하게 나누는 경우가 있다[8]. 소거작업은 세그먼트 당 약 2ms정도의 시간이 소요되는 긴 작업(long operation)이다[11]. 그리고 각 세그먼트 당 소거작업의 수의 한계가 있어서 무한정 소거작업을 할 수 없도록 되어 있다. 제조 업체에서는 한 세그먼트 당 소거작업의 횟수의 한계를 보증한다[11]. 일반적으로 100,000 - 1,000,000회 정도의 소거작업을 보증한다. 보증하는 회수 이상 소거작업을 하게되면 오동작을 일으킬 수 있게 되어 그 세그먼트를 다시 소

www.dbpia.co.kr

거할 수 없어 쓰기 작업을 수행할 수 없게된다. 반면에 디스크 같은 경우 수명이 다하면 읽기와 쓰기 모두 불가능해 지지만 플래시메모리의 경우는 쓰기는 불가능하여도 읽기는 가능하다.

2. 플래시메모리 클리닝 작업

앞서 언급한대로 플래시메모리는 같은 주소에 덮어쓰기가 불가능하다. 따라서 어떤 블록을 수정하려고 한다면 먼저 있던 블록을 무효화시키고 쓰기가 가능한 세그먼트를 찾아 수정된 내용을 다시 기록하여야 한다. 쓰기와 수정이 계속된다면 플래시메모리에는 더 이상 데이터를 기록할 공간이 없어지게 되는데 이는 일반적인 디스크나 DRAM이 다 차는 경우와는 조금 틀리다. 디스크나 DRAM같은 경우 일반적으로 다 찼다는 것은 유효한 데이터들이 다 차는 경우인데 반해 플래시메모리의 경우는 무효한 데이터와 유효한 데이터들로 가득 차 더 이상 기록할 공간이 없는 경우를 말한다. 이러한 상태가 오기 전에 세그먼트 클리닝작업을 통해서 쓸 수 있는 플래시메모리의 공간을 항상 확보하고 있어야 한다.

세그먼트 클리닝 작업은 먼저 그림 1과 같이 유효한 블록들을 쓰기가 가능한 다른 블록으로 옮겨서 세그먼트 전체를 무효한 블록만으로 구성되도록 만든다. 이후에 무효한 블록들로만 구성되어 있는 세그먼트를 소거작업(erase operation)을 수행한다. 이렇게 클리닝 된 세그먼트 들은 쓸 수 있는 블록만으로 구성되게 된다.

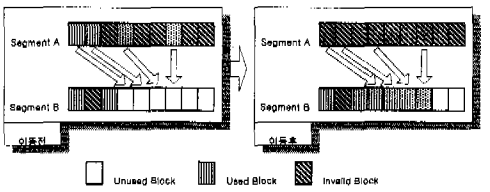


그림 1. 클리닝 작업

일반적인 디스크나 메모리에서는 세그먼트 클리닝 작업이 필요하지 않지만 플래시메모리 구조에서는 세그먼트 클리닝 작업이 필요하게 된다. 세그먼트 클리닝작업을 하는데 소모되는 작업을 측정하기 위해서는 세그먼트 클리닝 비용을 계산해야 한다. 클리닝 비용은 세그먼트 안에 옮겨야 할 블록의 수로 결정된다. 클리닝 작업이 시작되면 처음에 클리닝을 할 세그먼트를 정하고 다음으로 옮겨야할 블록을 검색한 후 빈 세그먼트를 찾아 옮기게 되는데 블록마다 읽기 작업과 쓰기 작업을 한번씩 해야 하고 블록이 다 옮겨진 후 소거작업을 하게된다. 쓰기 작업이 읽기 작업보다 시간이 많이 걸리게 된다. 여기서 블록의 수가 많으면 이동하는데 드는 비용이 늘어나게 되므로 클리닝 비용은 증가하게 되고 반대로 옮겨야할 블록의 수가 작으면 클리닝 비용은 감소한다. 그리고 전체 메모리 중에 유효한 데이터들이 많아 그 비율이 높아지게 되면 무효한 블록의 수가 적어 클리닝 작업을 많이 수행해도 새로 생성되는 쓰기 가능한 블록의 수는 적다. 따라서 계속하여 세그먼트 클리닝을 요구하게 되므로 또한 클리닝 비용을 증가시키는 요인이 된다.

Ⅲ. 방향성 클리닝 정책 및 구성

1. 플래시메모리 관리자

방향성 클리닝 정책을 적용하는데는 세그먼트 테이블(segment table), 메모리 사상 테이블(memory mapping table), 플래시메모리 사상 테이블(flash memory table), 플래시메모리 관리자의 구성요소를 가지고 있어야 한다.

Flag	Block Status	Unused Block Pointer	# of Invalid Blocks	# of Erase Operations

그림 2. 세그먼트 테이블

그림 2의 세그먼트 테이블은 세그먼트를 관리하는데 필요한 요소들을 포함하고 있다. 플래그(flag)는 이 세그먼트가 현재 쓰기 가능한 블록의 유무를 나타낸다. 클리닝 작업이나 쓰기 작업에서 쓰기 가능한 블록을 찾을 때 참고된다. 블록 상태(block status)는 세그먼트 내의 각각의 블록들이 현재 어떤 상태인지를 표시하도록 한다. 유효한 블록(valid block), 무효한 블록(invalid block), 미사용 블록(unused block)의 세 가지 상태가 있을 수 있다. 세그먼트 안의 블록의 수에 따라 이 요소의 크기가 결정된다. 미사용 블록의 위치(unused block pointer)는 세그먼트에 미사용 블록의 최초 위치를 나타낸다. 세그먼트에 블록을 할당할 때 논리적 주소(logical address)를 플래시메모리의 물리적 주소(physical address)로 변환하는 사상 테이블을 이용하므로, 플래시메모리 블록의 주소는 플래시메모리 관리자를 제외하고는 아무런 의미가 없다. 그러므로 쓰기 작업이 수행되는 블록들은 세그먼트 안에서 주소가 낮은 곳부터 채워진다. 무효블록(invalid block)의 수는 현재 무효한 블록의 수를 나타낸다.

www.dbpia.co.kr

무효블록의 수는 플래시메모리 관리자가 클리닝 작업을 수행할 때 세그먼트를 선택하기 위해 사용된다. 세그먼트마다 무효한 블록들을 찾아 몇 개의 블록이 무효한지를 찾기에는 시간이 많이 소요되므로 플래시메모리 관리자가 쓰기 작업에서 수정 작업이 발생할 때 무효한 블록의 수를 증가시켜 현재 무효한 블록이 몇 개인지를 알 수 있도록 한다. 소거작업(erase operation)의 횟수는 이 세그먼트가 몇 번 소거작업을 수행했는지를 나타낸다. 세그먼트 평준화를 위해 플래시메모리 관리자가 참고할 수 있도록 한다.

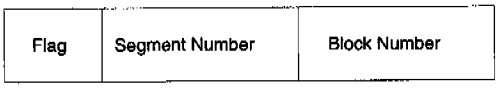


그림 3. 메모리 사상 테이블

메모리 사상 테이블(memory mapping table)은 읽기 작업이나 쓰기 작업이 수행될 때 논리적 주소(logical)를 플래시메모리의 물리적 주소(physical address)로 변환할 수 있도록 한다. 플래그(flag)는 논리적 주소에 플래시메모리가 할당 여부를 표시한다. 만약 플래시메모리에 할당되어 있지 않았을 때 쓰기 작업을 수행하면 새로운 블록을 할당받아 새로운 세그먼트 번호와 블록의 번호로 바꿔준다.

또 필요한 테이블로 플래시메모리 사상 테이블(flash memory table)이 있다. 플래시메모리 사상 테이블은 클리닝 작업에서 사용된다. 클리닝 작업에서 유효한 블록들을 미사용 블록을 포함하는 세그먼트로 이동시킨다. 이 유효한 블록들의 논리적 주소의 정보를 가지고 있어야 블록의 이동이 일어난 후 메모리 사상 테이블(memory mapping table)의 세그먼트 번호와 블록 번호를 수정할 수 있다.

세그먼트 테이블, 메모리 사상 테이블, 플래시메모리 사상 테이블은 수정이 빈번하게 일어나므로 쓰기 작업의 시간이 읽기에 비해 오래 걸리는 플래시메모리에 저장하는 것은 많은 성능의 저하를 가져온다. 따라서 플래시메모리보다는 DRAM이나 SRAM에 저장하도록 한다. 또한 이동 컴퓨터의 전원을 별도로 공급하여 전원이 차단되어도 데이터가 보존 되도록 한다. 세그먼트 테이블은 전체 플래시메모리에 비해 상대적으로 작기 때문에 작은 크기의 DRAM이나 SRAM이라도 충분히 저장 가능하다. 테이블들은 플래시메모리 관리자에게 매우 중요하여 만약 이 테이블들이 손실되는 경우에는 플래시메모리를 관리할 수 없게 된다. 따라서 이 테이블들은 운영체제에서 따로 저장(backup)해두어야 한다.

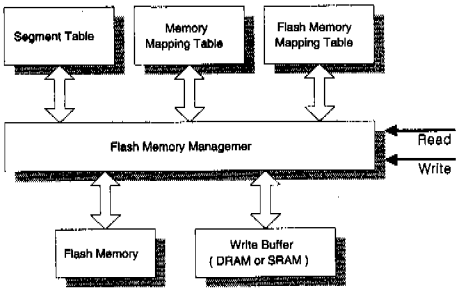


그림 4. 플래시메모리 관리자

그림 4는 플래시메모리 관리자를 보여준다. 플래시메모리 관리자는 다음의 몇 가지 작업을 수행한다.

* 읽기 작업, 쓰기 작업에 대해 논리적 주소를 물리적 주소로 변환
* 쓰기 작업에서 블록의 이동과 할당
* 클리닝 작업

읽기 작업을 수행하는 경우 논리적 주소를 받아 메모리 사상 테이블(memory mapping table)을 이용하여 물리적 주소로 전환하여 플래시메모리 블록을 읽을 수 있도록 만들어 준다. 쓰기 작업을 수행하면 쓰기 버퍼(write buffer)에 저장하고 메모리 사상 테이블의 세그먼트 번호와 블록 번호가 쓰기 버퍼의 블록을 나타내도록 수정하여 준다. 만약 쓰기 버퍼가 다 차는 경우는 LRU(least recently used)방법을 이용하여 플래시메모리 블록에 이동하도록 하고 메모리 사상 테이블의 내용을 수정한다. 쓰기 버퍼의 사용은 쓰기 작업이 많을수록 플래시메모리의 수명과 성능을 개선시킨다. 같은 블록이 계속하여 쓰기 작업이 수행되는 경우 플래시메모리에 쓰기 작업을 수행하지 않았다가 후에 한 번만 쓰기 작업을 수행하면 되므로 플래시메모리의 수명을 연장시킬 수 있고, 쓰기 작업의 시간이 느린 플래시메모리에 쓰기 전에 DRAM(또는 SRAM)에 쓰므로 쓰기 작업의 성능도 DRAM만 큼의 성능을 얻을 수 있다. 쓰기 작업을 수행할 때 미사용 블록의 수가 일정 수준 이하로 내려가게 되면 플래시메모리 관리자는 클리닝 작업을 수행하게 된다. 이 클리닝 작업 역시 사용자가 작업을 하지 않는 쉬는(idle) 상태일 때 수행되도록 하고 계속하여 미사용 블록의 수가 더 줄어드는 상태가 되면 쓰기 작업을 잠시 멈추도록 하고 클리닝 작업을 하도록 한다. 여기서 미사용 블

www.dbpia.co.kr

록의 수는 클리닝 작업을 하는데 필요한 최소한의 수가 확보되어 있어야 한다. 일반적으로 하나의 세그먼트에 포함되어 있는 블록의 수만큼은 미사용 블록의 수가 확보되어 있어야 한다.

2. 방향성 클리닝 정책

세그먼트 클리닝 비용은 시스템의 성능에 매우 큰 영향을 미치므로 클리닝 정책을 적용할 때 중요한 참고가 된다. 비용을 낮추기 위해서 취할 수 있는 일반적인 방법은 greedy 방법이 있는데, 먼저 전체 세그먼트 중에서 무효한 블록이 많은 것을 찾아 선택하여 세그먼트 클리닝 작업을 하고 이 세그먼트는 다음 쓰기 작업이 수행 될 때 가장 먼저 선택되어 채워진다[10]. 이 방법은 먼저 쓰여진 것이 먼저 세그먼트 클리닝 될 확률이 높아 FIFO(first-in-first-out)의 성격을 가지고 있다. 이 방법은 구현이 간단하지만 클리닝 비용에서는 그다지 이득을 보지 못한다.

일반적으로 데이터들은 균일한 수정 빈도를 가지고 있지 않다. 어떤 데이터 블록들은 한번 쓰여지면 읽기만 계속해서 수행하고 반대로 어떤 데이터 블록들은 수정작업이 빈번하게 일어난다. 수정이 빈번하게 일어나는 데이터 블록을 hot 블록, 수정작업이 거의 없는 블록을 cold 블록이라고 할 때, greedy 방법에서 hot 블록과 cold 블록이 결과적으로 균일하게 섞이게 되므로 블록의 이동이 일어나지 않아야 좋은 cold 블록들까지 이동이 함께 일어나 결국 세그먼트 클리닝 비용이 많이 들게 된다. 뿐만 아니라 필요이상으로 블록의 이동이 일어나 결국 세그먼트 클리닝의 횟수가 늘어나게 되어 수명에도 좋지 않은 영향을 준다. 이 같은 점을 해결하기 위해서 hot 블록과 cold 블록을 분리해내는 작업이 필요하다.

실제로 유닉스의 예를 들면 읽기 작업보다 쓰기 작업이 조금 더 많고 데이터의 성격에 따라 어떤 데이터는 자주 쓰여지고 어떤 데이터는 덜 쓰여진다. 여기서 쓰기 작업이 읽기 작업보다 많은 이유는 실제적인 데이터의 쓰기 작업에 메타데이터를 포함되기 때문이다. 디스크 드라이브에서는 약 90%의 액세스가 10%의 블록에 주로 집중된다[6]. 이동 컴퓨터 환경에서도 마찬가지로 사용자는 특정 파일을 계속하여 일정시간동안 사용한다[5]. 쓰기 작업이 빈번할수록 hot 블록과 cold 블록을 구분하여 세그먼트 클리닝 비용을 줄이는 것이 중요하다.

방향성 클리닝 정책에서는 hot 블록과 cold 블록을 구분해내고 또 그 작업을 간단히 하는 구조로 컴퓨터 연산 처리 능력이 약한 이동 컴퓨터에 적합하도록 하였다. 실제로 hot 블록과 cold 블록을 구분하는 것은 앞으로 쓰기 작업을 예측하여 구분하는 것이 이상적이지만 현실적으로 불가능하기 때문에 여기서는 수정작업이 일어나는 빈도를 이용해 구분하도록 하였다. 수정이 되는 횟수를 기록하여 나중에 참고하는 것이 일반적인 방법이지만 이런 방법은 블록마다 저장공간을 따로 필요로 한다. 그리고 예전에 많이 수정되었던 블록이더라도 최근 얼마간은 수정되지 않은 경우 앞으로 수정이 빈번하지 않을 것으로 예상됨에도 불구하고 hot 블록으로 간주될 수 있다. 따라서 이 방법은 불합리하다. 방향성 클리닝 정책에서는 직접적으로 횟수를 기록하기보다는 세그먼트 번호를 이용하여 hot 블록과 cold 블록을 구분하도록 한다.

그림 5와 그림 6은 세그먼트 B에 있는 hot 블록과 cold 블록이 분리되는 과정을 보여 주고 있다. 세그먼트 번호는 낮은 순서대로 A, B, C이다. 블록의 수정이 필요하여 데이터가 이동한다면 이것은 hot 데이터로 간주하고 쓰기 가능한 세그먼트 중에서 자신이 속한 세그먼트 번호 보다 낮은 세그먼트로 이동하는데 이때 자신이 속한 세그먼트와 가장 가까운 세그먼트를 선택하도록 한다. 세그먼트 클리닝 때에는 수정작업 때와는 반대로 쓰기 가능한 세그먼트 중에서 자신이 속한 세그먼트 번호 보다 높은 세그먼트로 이동하는데 이때 자신이 속한 세그먼트와 가장 가까운 세그먼트를 선택한다. 자신이 속한 세그먼트 번호가 최소이거나 최대일 경우에는 자신과 가장 가까운 쓰기 가능한 세그먼트를 선택한다. 그리고 수정의 경우 자신의 세그먼트 번호보다 낮은 세그먼트 중에서 쓰기 가능한 블록이 없는 경우와 반대로 클리닝 작업 때에는 클리닝되는 세그먼트 번호보다 높은 세그먼트 중에서 쓰기 가능한 블록이 없는 경우 역시 자신과 가장 가까운 쓰기 가능한 세그먼트를 선택하도록 한다.

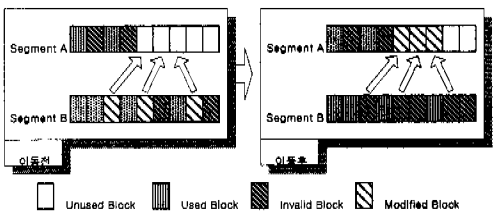


그림 5. 수정으로 인한 블록의 이동

www.dbpia.co.kr

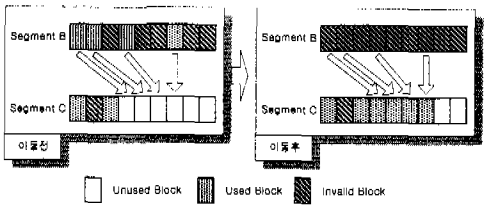


그림 6. 클리닝 작업으로 인한 블록의 이동

블록의 수정과 세그먼트 클리닝 작업이 계속해서 이루어지게 되면 hot 블록들은 세그먼트 번호가 낮은 곳에 집중되게 되고 반대로 cold 블록들은 세그먼트 번호가 높은 곳에 집중되게 된다. 방향성 클리닝 정책에서는 hot 블록과 cold 블록을 구분해내기 위해서 별도의 연산이 필요 없어 작업에 부하가 적게 걸리고 hot 블록이었다가 cold 블록으로 또는 cold 블록이었다가 hot 블록으로 변하는 경우에도 유동적으로 변화에 대처할 수 있다. 예를 들면 처음에는 hot 블록이어서 세그먼트 번호가 낮은 곳에 있던 블록이 수정 작업을 한동안 거치지 않게 되면 세그먼트 클리닝 작업을 계속해서 거치게 되어 cold 블록이 모이는 곳으로 이동하게

그림 7은 쓰기, 지우기, 세그먼트 클리닝작업이 어느 정도 이루어졌을 때 블록의 분포를 나타내고 있다. (a)는 greedy 방법 (b)는 본 연구에서 제안한 방향성 정책에 의한 방법으로 Direction이라 표시했다. (b)에서는 세그먼트 번호가 낮을수록 데이터 블록의 수정이 빈번하게 일어나 무효한 블록의 수가 많이 분포하고 있고 세그먼트의 번호가 높을수록 유효한 블록의 수가 많아지고 있다. 이것은 hot 블록이 모여있는 세그먼트는 클리닝 비용을 낮추는 작용을 하게되고 반대로 cold 블록에서는 불필요한 이동을 억제하도록 하고 있다.

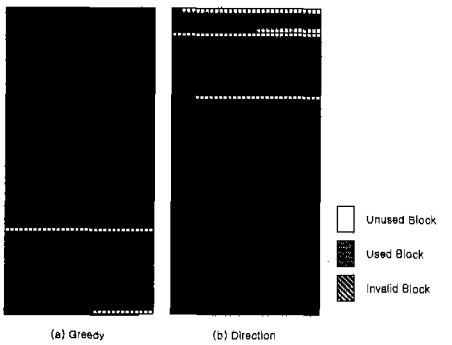


그림 7. 블록분포

3. 수정된 방향성 클리닝 정책

전의 방향성 클리닝 정책에서의 기본은 hot 블록과 cold 블록의 분리에 있다. 그러나 만약 수정이나 클리닝 작업에서 이동 할 수 있는 세그먼트에 제한이 있다면 hot 블록과 cold블록의 분리는 제대로 이루어지지 않는다. 그림 8에서 (a)는 방향성 클리닝 정책이고 (b)는 수정된 방향성 클리닝 정책이다. (a)는 방향성 클리닝 정책에서 클리닝이 잘못 되어진 경우를 보여주고 있다. 이 경우 쓰기 가능한 블록이 있는 세그먼트가 특정부분에 몰려있음을 볼 수 있다. 방향성 클리닝 정책에서 자주 수정되는 블록이 세그먼트 번호가 낮은 부분에 몰리는 경우에 이 부분에 무효한 블록이 많아지게 되고 따라서 이 부분에 클리닝 작업이 이루어질 확률이 많다. 그러나 이렇게 클리닝 작업이 집중되어 일어나게 되면 데이터의 분리가 잘 이루어지지 않는다. 그 이유는 수정작업이나 클리닝 작업에서 cold 블록이 모여있는 부분이나 hot 블록이 모여있는 부분 모두 빈 세그먼트가 집중되어 있는 곳으로 옮겨져 hot 블록과 cold 블록이 섞이는 결과가 나타나게 된다. 이는 방향성 클리닝의 효율을 떨어뜨리는 결과를 가져온다.

위와 같은 문제를 해결하기 위해서 수정된 방향성 클리닝 정책에서는 쓰기 가능한 블록이 있는 세그먼트를 플래시메모리 전체에 골고루 퍼지게 배치하여 블록들이 옮겨 질 때 옮겨질 블록이 속한 세그먼트의 주위에 있는 세그먼트로 옮겨지도록 한다. 클리닝 작업을 수행할 때 세그먼트 하나만 클리닝 작업을 하는 것이 아니라 여러 개의 세그먼트를 클리닝을 하게 되는데 이때 연속된 몇 개의 세그먼트 안에는 반드시 미사용 블록이 있는 세그먼트를 포함하도록 클리닝 작업을 한다.

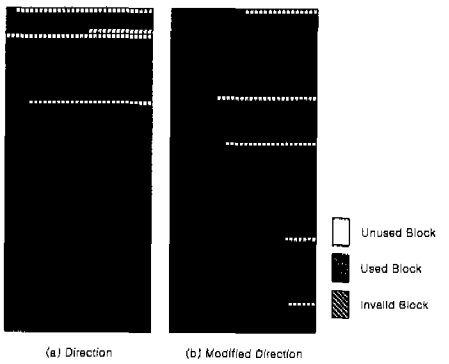


그림 8. 방향성 클리닝 정책과 수정된 방향성 클리닝 정책의 비교

www.dbpia.co.kr

그림 8의 (b)수정된 방향성 클리닝 정책을 보면 쓰기 가능한 블록이 있는 세그먼트가 전체 세그먼트에 골고루 퍼져 있음을 볼 수 있다. hot 블록과 cold 블록은 수정 혹은 클리닝으로 인한 이동이 일어날 때 각각 자신이 속한 세그먼트에 근접한 세그먼트로 옮겨져 cold 블록과 hot 블록이 뒤섞일 확률을 줄였다. 미사용 블록(unused block)의 수를 항상 그림 8의 (b)처럼 유지시킬 수는 없다. 미사용 블록이 많게 유지를 하려면 미사용 블록이 적을 때 보다 클리닝 작업이 빈번하게 일어나게 되므로 결국 미사용 블록이 세그먼트 하나 정도의 수만 남는 경우가 발생하는데 이때에는 역시 hot 블록이나 cold 블록이나 같은 세그먼트로 이동할 수밖에 없어 방향성 클리닝 정책과 같은 효과가 나타난다. 그러나 방향성 클리닝 정책보다는 블록들이 뒤섞이는 확률을 줄여 더 적은 클리닝 작업을 수행하도록 하였다.

4. 세그먼트 평준화

클리닝 정책에서 고려해야할 사항으로 세그먼트 평준화(wear leveling, cycle leveling)가 있다[8]. 플래시메모리는 세그먼트마다 수명이 있기 때문에 만약 어느 세그먼트만 집중적으로 소거작업이 일어나게 되면 전체 플래시메모리가 고르게 한계 수명까지 쓰는 것이 아니라 빈번하게 사용된 세그먼트는 더 이상 쓰기가 불가능하게 되어 전체 메모리의 크기가 줄어드는 결과가 나오고 더불어서 전체 메모리내의 유효한 블록의 수가 증가되어 세그먼트 클리닝 작업이 더욱 빈번하게 발생하게 된다. 이는 또 세그먼트의 수명에 영향을 주어 메모리 크기가 줄어드는 악순환을 거듭하게 된다. 방향성 클리닝 정책 및 수정된 방향성 클리닝 정책에서도 hot 블록이 집중되는 세그먼트가 다른 세그먼트에 비해 여러 번 소거작업이 일어나는 구조를 가지고 있다. 그래서 이러한 문제점을 해결하기 위해 방향성 클리닝 정책에서는 일정 수의 작업이 일어난 후에 블록의 이동방향을 바꾸어 이 문제를 해결하도록 하였다. 즉 블록의 수정으로 인한 이동의 겨우 세그먼트 번호가 낮은 방향으로 이동하던 것을 주기적으로 바꾸어주어 세그먼트의 소거작업이 고르게 일어나도록 한다. 물론 세그먼트 클리닝 작업의 경우도 같이 바꾸어 주도록 하였다.

Ⅳ. 시뮬레이션 (Simulation)

각 클리닝 정책에 따른 성능을 비교하기 위하여 시뮬레시션을 수행하였다. 시뮬레이션 수행에서는 쓰기 작업의 집중하는 정도에 따라 greedy 방법, 방향성 클리닝 정책, 수정된 방향성 클리닝 정책을 비교하였다. 시뮬레이션 조건은 버퍼가 없는 상태에서 읽기 작업을 제외하고 쓰기 작업만을 블록단위로 1,000,000회 수행하였다. 읽기 작업은 클리닝 정책에 영향을 미치지 않으므로 제외하였다. 1,000,000회를 임의의 블록에 쓰기 작업을 하여 수정작업과 쓰기가 이루어졌으며 메모리가 모두 차는 것을 방지하기 위하여 전체 메모리 내에 유효블록의 비율이 일정 수준을 넘으면 쓰기 작업은 지우기 작업으로 대치하여 유효 블록의 비율을 유지하도록 하였다. 전체 세그먼트의 수는 64개로 하였고 한 세그먼트 안의 블록의 수는 32개로 하였다. 쓰기 작업을 통해 얻은 결과는 각 세그먼트의 소거 작업의 수, 클리닝 작업으로 인해 이동된 블록의 수를 얻었다. 초기 상태에서의 작업은 평형상태(steady state)와 거리가 있으므로 유효한 블록이 일정 수준에 이르는 시점에서 1,000,000회를 수행하고 소거작업의 수와 이동된 블록의 수를 기록하였다. 시뮬레이션 조건은 쓰기 블록이 집중되는 정도의 변화, 유효한 블록의 비율의 변화, 클리닝 작업에서 세그먼트의 수 그리고 방향을 전환하는 주기의 변화에 대하여 시뮬레이션을 수행하였다.

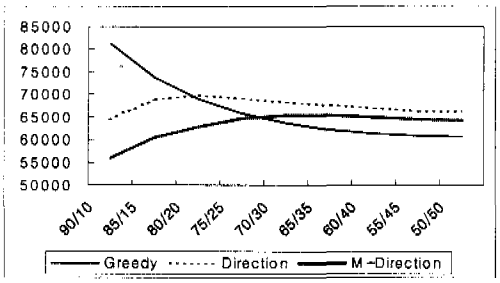


그림 9. 쓰기 작업의 집중도에 따른 클리닝 작업의 수

그림 9는 데이터의 집중도에 따른 각 정책의 클리닝 작업 횟수에 대한 결과를 그래프로 나타낸 것이다. 가로축은 데이터의 집중도를 나타내는데 90/10은 90%의 쓰기 작업이 10%의 블록에 수행되었다는 것을 나타낸다. 쓰기 작업을 100만회 수행하였고 유효한 블록의 비율은 전체 플래시메모리의 80%로 실험하였다. 클리닝 작업의 횟수는 초기에 플래시메모리가 완전히 비어 있는 상태에서 실행한 것이 아니라 유효한 블록의 비율이 80%가 되는 순간부터 클리닝 작업의 수를 계산하였다. 방향성 클

www.dbpia.co.kr

리닝 정책과 수정된 방향성 클리닝 정책에서 클리닝 작업을 수행할 때 클리닝 세그먼트의 수는 각각 9개 7개이다. 이러한 조건에서 hot 블록과 cold 블록이 많은 차이를 보이는 80/20에서부터 방향성 클리닝 정책과 수정된 방향성 클리닝 정책이 적은 클리닝 작업 수를 나타내기 시작했다. 일반적으로 디스크의 경우 90/10의 경우가 일반적이므로 90/10에서 클리닝 작업의 수를 살펴보면 수정된 방향성 클리닝 정책이 greedy 정책에 비해 25000여 회 더 적은 클리닝 작업을 하였다. 이것을 쓰기 작업 당 클리닝 작업이 일어난 수로 바꾸면 greedy 정책은 쓰기 작업 1회당 0.081회, 수정된 방향성 클리닝 정책은 0.056회로 나타나 응답시간도 평균적으로 빨라짐을 보였다.

그림 10은 위 그림 9의 실험에서 유효한 블록의 비율을 변화시켜가면서 실험하였다. 쓰기 작업의 집중도는 90/10으로 하고 다른 조건들은 그림 9의 실험과 모두 같다. 그래프에서 보듯이 클리닝 작업은 유효한 블록의 비율이 높을수록 높아지고 있다. 유효한 블록의 비율이 80%정도부터에서 급격히 클리닝 작업의 수가 높아져 유효한 블록의 비율을 80% 정도에서 유지하도록 해야함을 나타내고 있다. 유효한 블록의 비율이 높거나 낮음에 관계없이 greedy 정책보다는 수정된 방향성 클리닝 정책이 클리닝 작업을 적게 수행함을 보였다. 특히 유효한 블록의 비율이 70%이상인 경우에 더욱 적은 클리닝 작업을 수행하였다.

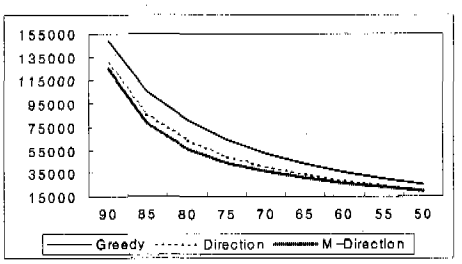


그림 10. 유효 블록의 비율에 따른 클리닝 작업의 수

그림 11은 클리닝 작업에서 일단 클리닝이 시작될 때 몇 개의 세그먼트를 동시에 클리닝하는가에 대한 시뮬레이션 결과이다. 클리닝 세그먼트의 수가 늘어날수록 hot 블록과 cold 블록이 잘 나누어져 클리닝 작업의 수가 줄어들었다. 방향성 클리닝 정책은 클리닝 세그먼트가 몰리는 경우에 블록의 분리가 잘 이루어지지 않는 경우가 많아 클리닝 세그먼트의 수를 늘려도 별 이득이 없다. 그러나 수정된 방향성 클리닝 정책에서는 클리닝 세그먼트의 수가 많을수록 분리가 잘 이루어져 클리닝 작업 횟수가 줄어들었다. 그러나 7개 이상이 되면 클리닝 세그먼트를 만드는데 무리가 생겨 더 많은 클리닝 횟수를 나타냈다.

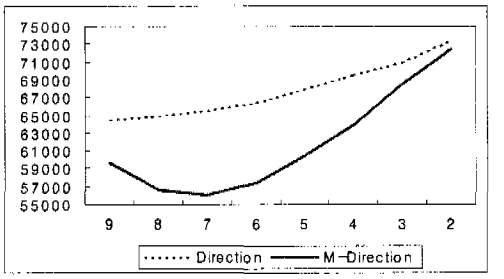


그림 11. 동시수행 클리닝 세그먼트의 수에 따른 클리닝 작업 수

그림 12는 방향성 클리닝 정책 및 수정된 방향성 클리닝 정책에서 방향을 반대로 바꾸어줄 때 어느 정도의 쓰기 작업이 수행된 후에 방향을 바꾸어 주어야 하는지를 나타낸다. 방향성 클리닝 정책에서는 블록의 분리가 잘 이루어지지 않았기 때문에 많은 변화가 없지만 수정된 방향성 클리닝 정책에서는 약 20,000여 회 이상에서 좋은 결과를 보여주었다. 따라서 세그먼트 평준화를 위해서는 20,000여 회 이상 쓰기 작업을 한 후에 방향을 바꾸어주면 됨을 알 수 있다.

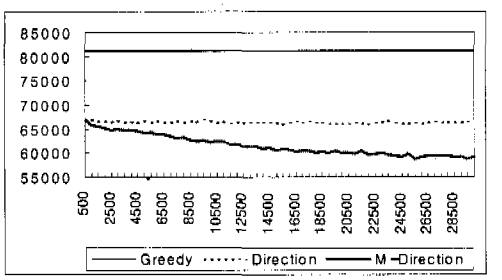


그림 12. 방향전환 주기에 따른 클리닝 작업의 수

그림 13은 그림 12의 결과로 20,000회 쓰기 작업 수행 후 방향전환을 해 주었을 경우(DC)와 방향 전환을 해주지 않은 경우(NDC)를 비교한 것이다. 방향전환을 주기적으로 해준 결과 각 세그먼트에서 일어난 소거작업의 수가 비슷하였다. 각 세그먼트에서 소거작업의 수는 방향을 전환하지 않았을 때

www.dbpia.co.kr

2400회 정도 차이가 났지만 방향전환을 해준 경우 약 1,000회 정도에서 평준화되었다.

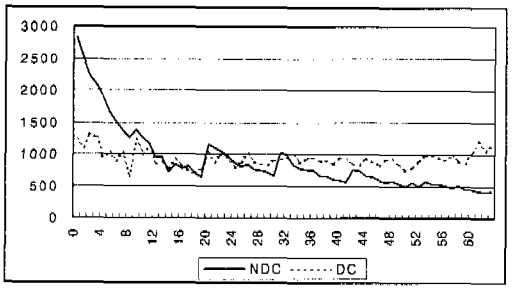


그림 13. 각 세그먼트의 소거작업의 수

V. 결 론

플래시메모리는 이동컴퓨터에 적합하여 앞으로 널리 사용될 것이다. 그러나 플래시메모리의 단점 때문에 쓰기 작업에서 성능의 저하가 있게 된다. 본 논문에서는 기존 플래시메모리 저장기법을 이동컴퓨터에 적합하도록 하고 클리닝 회수를 줄임으로써 쓰기 성능을 개선하였다. 시뮬레이션 결과 일반적인 디스크 사용 유형인 쓰기 작업이 집중되는 환경에서 클리닝회수가 현저히 줄어듦을 보였다.

전체 메모리 중에서 유효블록이 80%이상이고 쓰기 작업의 90%가 10%의 블록에 집중된다면 기존의 정책보다 약 25%정도의 성능 개선을 보였다. 쓰기 작업 1회당 greedy 정책은 0.081회 방향성 클리닝 정책은 0.064회 수정된 방향성 클리닝 정책은 0.056회로 수정된 방향성 클리닝 더 적은 클리닝 작업을 수행하는 것으로 나타났다. 이는 쓰기 버퍼를 거치지 않았을 때이고 쓰기 버퍼를 거친다면 더 적게 클리닝 작업이 발생할 것으로 예상된다. 또한 수정된 방향성 클리닝 정책에서 적절한 클리닝 세그먼트의 수를 결정할 수 있었으며 어떤 크기의 주기에 가장 효율적인 클리닝이 가능한지를 시뮬레이션을 통해 결과를 얻었다.

참 고 문 헌

[1] Fred Douglis, Ramon Caceres, Brian Marsh, Frans Kaashoek, Kai Li, and Joshua Tauber, "Storage Alternatives for Mobile Computers, Proceedings of the First Symposium on Operating Systems Design and Implementation," USENIX Association, pp. 25-37, November 1994.

[2] George H. Forman and John Zahorjan, "The Challenges of Mobile Computing," Technical Report 93-11-03, University of Washington, March 1994.

[3] B. Marsh, F. Douglis, and P. Krishnan, "Flash Memory File Caching for Mobile Computers," in Proceedings of the 27th Hawaii International Conference on System Sciences,pages 279-291, IEEE, January 1994.

[4] Tomaz Imielinkski, Henry F. Korth, "Intoduction to Mobile Computing," Mobile Computing, pp 1-43, Imielinski and Korth eds., Kluwer, 1995.

[5] Geoffrey H. Kuenning, Gerald J. Popek, Peter L. Reiher, "An Analysis of Trace Data for Predictive File Caching in Mobile Computing," Proc. USENIX conf., 291-303, Bostone, June 1994

[6] C. Reummer and J Wilkes, "UNIX disk access patterns," In Proceedings of the 1993 Winter USENIX Conference, pp. 405-420.

[7] Mendel Rosenblum, and J Wilkes, "The Design and Implementation of a Log-Structured File System." ACM Transactions on Computer System, vol. 10, no. 1, pp. 26-52, February 1992.

[8] Deborah See and Clark Thurlo, "Managing Data in an Embedded System Utilizing Flash Memory," Intel, 1995.

[9] Roy Wamt, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, Mark Weiser, "The PARCTAB Ubiquitous Computing Experiment," Mobile Computing, ,Imielinski and Korth eds., Kluwer, 1995.

[10] Michael Wu and Willy Zwaenepoel, "eNVy : A Non-Volatile, Main Memory Storage System," Proceedings 6th International Conference on Architectural Support for Programming languages and Operating System, 1994.

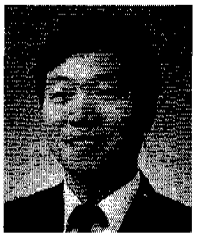
[11] Flash Memory Data Book, samsung, 1997.

www.dbpia.co.kr

민 용 기(Yong-ki Min) 정회원

1972년 8월 28일 출생
1997년 2월 : 아주대학교 화학공학과(공학사)
1999년 2월 : 아주대학교 컴퓨터공학과(공학석사)
<주관심 분야> 이동컴퓨팅, 분산파일시스템

박 승 규(Seung-kyu Park) 정회원



1950년 6월 17일 출생
1974년 2월 : 서울대학교 공응용수학과(공학사)
1976년 2월 : 한국과학원(KAIST) 전산학(공학석사)
1982년 3월 : 프랑스 INP de Grenoble 전산학 (공학박사)
1976년 3월~1977년 10월 : 한국과학기술연구소(KIST) 연구원
1977년 11월~1992년 2월 : 한국전자통신연구소(ETRI) 부장
1984년 3월~1985년 9월 : 미국 IBM 왓슨 연구소 연구원
1992년 3월~현재 : 아주대학교 정보 및 컴퓨터공학과 정교수
<주관심 분야> 멀티미디어 처리구조, 다중처리 및 MPP 시스템, 실시간시스템, 이동컴퓨팅 시스템 등

www.dbpia.co.kr