블록 클리닝 비용 분석에 기초한 MODA할당 정책 설계 및 구현

(Design and Implementation of MODA Allocation Scheme based on Analysis of Block Cleaning Cost)

백 승 재 † 최 종 무 ††
(Seungjae Baek) (Jongmoo Choi)

요 약 플래시 메모리는 덮어 쓰기 제약이나, 쓰기와 삭제 연산의 단위가 다르다는 등의 특징을 가지고 있다. 따라서 플래시 메모리를 저장장치로 사용하는 파일 시스템은 블록 클리닝을 필요로 하며 이는 파일 시스템의 주된 병목으로 작용한다. 이에 따라 본 논문에서는 플래시 메모리 기반 파일 시스템의 병목 요소인 블록 클리닝에 따른 성능향상에 대해 연구한다. 우선 블록 클리닝에 영향을 끼치는 성능 인자로서 이용률, 무효율, 순수도를 정의하였다. 이 세 가지 인자를 통해 블록 클리닝 비용을 분석한 결과, 파일 시스템 수준에서 제어 가능한 인자인 순수도가 블록 클리닝 비용에 많은 영향을 끼침을 확인할 수 있었다. 따라서 순수도를 높게 유지하여 블록 클리닝 비용을 최소화함으로서 파일시스템의 성능을 향상 시킬 수 있는 MODA 할당 정책(modification-aware)을 설계하였고, 이를 내장형 보드와 YAFFS(Yet Another Flash File System)상에 구현하였다. 실험 결과 MODA는 YAFFS의 순차 할당 기법에 비해 블록 클리닝 시간을 평균 123% 단축 시켰다.

키워드 : 플래시 메모리, 파일 시스템, 블록 클리닝, 순수도, 페이지 할당

Abstract Due to the restrictions of Flash memory such as overwrite limitation and write/erase operational unit differences, block cleaning is required in Flash memory based file systems and known as a key factor on the performance of file systems. In this paper, we identify three parameters, namely utilization, invalidity and uniformity, and analyze how the parameters affect the cost of block cleaning. The analysis show that as uniformity degrades, the cost of block cleaning increases drastically. To overcome this problem, we design a new modification-aware(MODA) page allocation scheme that strives to keep uniformity high by separating frequently-updating data from infrequently-updating data. Real implementation experiments conducted on an embedded system show that the MODA scheme can actually enhance uniformity of Flash memory, which consequently leads to reduce the cost of block cleaning with an average of 123%, compared to the traditional sequential allocation scheme that is used in YAFFS.

Key words : Flash memory, File System, Block Cleaning, Uniformity, Page Allocation

1. 서 론

플래시 메모리는 기존 저장장치와는 다른 하드웨어적인 특성을 가지고 있기 때문에 효율적인 사용을 위해서는 플래시 메모리의 특성을 고려한 관리 기법에 대한 연구가 필수적이다. LFS(Log-structured File System)의 세그먼트 클리닝이 LFS의 성능에 큰 영향을 끼치는 것처럼[1-5], 블록 클리닝은 플래시 메모리 기반 파일 시스템의 성능에 큰 영향을 끼친다[6-9]. 이는 플래시 메모리의 삭제연산이 다른 연산에 비해 매우 느리기 때문이다.

저장장치의 특성은 그 장치를 관리하는 소프트웨어

† 학생회원 : 단국대학교 정보컴퓨터과학과
ibanez1383@dankook.ac.kr
†† 종신회원 : 단국대학교 정보컴퓨터과학과 교수
choijm@dankook.ac.kr
논문접수 : 2007년 1월 22일
심사완료 : 2007년 7월 13일

정보과학회논문지: 시스템 및 이론 제34권 제11호(2007.12)

설계의 핵심 고려사항으로 작용한다. 예를 들어 디스크 기반 파일시스템인 FFS(Fast File System)에선 디스크의 물리적 특성을 고려하여 성능의 최대 병목 지점인 탐색시간(seek time)을 고려한 실린더 그룹 등의 정책을 도입했다[10]. 또한 같은 이유에서 LFS는 디스크 헤드의 탐색 시간을 최소화 하고 대역폭(bandwidth)을 최대로 사용하기 위해, 되도록 data를 모아쓸 수 있는 구조로 설계되었다[1,2]. 이 외에도 디스크 헤드의 이동시간을 고려한 다양한 최적화 기법이 연구되었다[11].

따라서 본 논문에서는 추후 설명될 이용률, 무효율, 순수도를 통해 플래시 메모리 기반 파일시스템 성능의 가장 큰 병목 요소인 블록 클리닝의 비용을 예측할 수 있는 성능 분석 모델을 수립하고 이를 검증하였다.

모델의 분석결과 블록 클리닝 비용은 순수도에 따라 매우 크게 영향을 받음을 알 수 있었다. 따라서 순수도를 향상 시킬 수 있는 새로운 할당 기법인 MODA를 설계 하였다. 제안된 기법은 내장형 시스템에 실제 구현되었다. 내장형 시스템 하드웨어는 400MHz로 동작하는 XScale CPU, 64MB SDRAM, 64MB NAND 플래시 메모리, 512KB NOR 플래시 메모리, 그리고 내장형 디바이스들로 구성된다[12]. 이 하드웨어 상에 리눅스 버전 2.4.19가 동작하며, NAND 플래시 메모리는 YAFFS에 의해 관리된다. 다양한 벤치마크들을 이용한 성능 분석 결과 제안된 기법이 기존 YAFFS에 비해 최대 평균 123% 블록 클리닝 시간을 단축시킴을 알 수 있었다. 또한 이용률이 커짐에 따라 더욱 많은 성능 향상을 얻을 수 있었다.

본 논문의 구성은 다음과 같다. 2절에서는 플래시 메모리의 특성과 동작 원리를 소개 한다. 3절에서는 블록 클리닝 비용 분석을 위한 모델을 설명하며, 4절에서는 제안한 모델을 검증한다. 5절에서는 모델을 통한 블록 클리닝 비용 분석을, 6절에서는 새로운 할당기법에 대해 설명한다. 7절에서는 실제 구현 및 성능 측정에 대해 소개하며, 관련 연구는 8절에 기술되며, 마지막 9절에서 결론 및 향후 연구 방향을 소개한다.

2. 플래시 메모리의 특성과 블록 클리닝

2.1 플래시 메모리의 특성

OR나 AND등의 플래시 메모리도 존재하지만 현재 주로 쓰이고 있는 것은 NOR와 NAND 타입의 플래시 메모리 이다. 본 논문에서는 주로 NAND 플래시 메모리를 주 대상으로 하지만 연구 결과의 대부분은 NOR 플래시 메모리에도 적용가능하다.

그림 1은 NAND플래시 메모리의 논리적인 구조를 보여준다. 그림 1에서 볼 수 있듯이 플래시 메모리는 페이지가 복수 개 모여서 블록을 형성한다. 페이지와 블록의

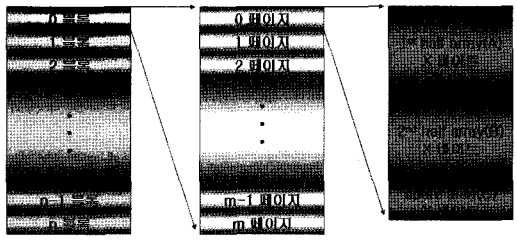

그림 1 플래시 메모리 구조

크기와 개수에 따라 NAND 플래시 메모리는 소블록(small block) NAND와 대블록(large block) NAND 두 종류로 구분된다. 소블록 NAND의 경우 528 bytes (데이타 저장을 위한 512 bytes + 스페어 영역을 위한 16 bytes) 크기의 페이지가 32개 모여 한 개 블록을 구성하고, 대블록 NAND의 경우 2112 bytes(데이타 저장을 위한 2048 bytes + 스페어 영역을 위한 64 bytes) 크기의 페이지가 64개 모여 블록을 구성한다.

플래시 메모리는 읽기, 쓰기, 삭제라는 세 가지 기본 연산을 지원하며 플래시 메모리의 종류에 따라 재기록(copy-back)등의 부가적인 연산을 지원하기도 한다[13]. 플래시 메모리의 읽기와 쓰기 연산은 페이지 단위로 수행되며, 삭제 연산은 블록 단위로 수행된다. 따라서 연산 수행 단위의 비대칭성이라는 플래시 메모리의 첫 번째 특성이 존재한다. 또한 읽기 연산의 수행시간은 20us, 쓰기연산의 수행시간은 200us, 삭제연산의 수행시간은 1.5ms로서[14] 각 연산 수행 시간의 비대칭성이라는 플래시 메모리의 두 번째 특성이 존재한다.

이외에 플래시 메모리의 특징으로는 다음과 같은 것이 있다. 셋째 플래시 메모리는 덮어쓰기가 불가능하다. 넷째 각 블록 당 제한된 숫자의 삭제 연산 횟수가 정해져 있다. 다섯째 탐색 시간이 없으며, 여섯째 블록 내에서 페이지에 대한 쓰기는 오름차순으로만 가능하다.

2.2 블록 클리닝의 필요성

기존 하드 디스크의 경우, 갱신(update) 요청이 도착하게 되면 아래 그림 2의 (a)와 같이 기존 위치에 새로운 데이타를 덮어씀으로써(overwrite) 처리해 줄 수 있다. 그러나 플래시 메모리의 경우에는 덮어 쓰는 것이 불가능하기 때문에 이를 위해 플래시 메모리를 위한 소프트웨어들은 다양한 해결책을 제시하였다. 기본적으로는 그림 2의 (b)와 같이 새로운 공간을 할당 받은 뒤 데이타를 쓰고 리매핑(re-mapping)하는, 즉 non-in-place update방식이 사용된다.

따라서 기존 저장장치인 디스크와는 다르게, 연산을 수행함에 따라 연산 수행의 최소 단위인 페이지의 상태는 유효(valid), 무효(invalid), 클린(clean) 이라는 세 가지 상태가 존재 한다. 이를 전이도를 통해 표현하면 그

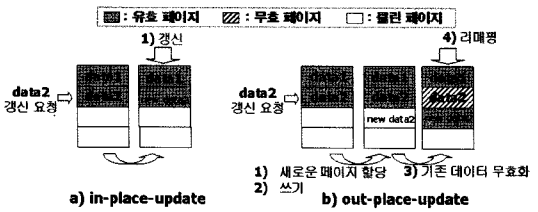

그림 2 플래시 메모리의 데이타 갱신

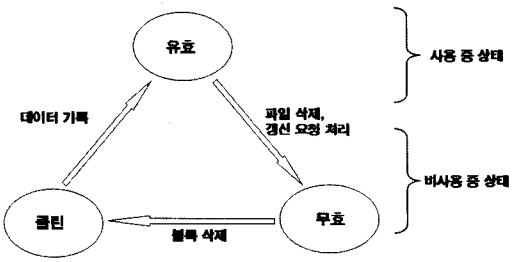

그림 3 페이지의 상태와 전이

림 3과 같다. 삭제 연산을 수행하면 해당 블록내의 페이지는 모두 클린 상태가 된다. 데이타를 기록하면 페이지는 유효 상태가 되며, 해당페이지의 데이타를 삭제 또는 갱신 하여 더 이상 유효한 데이타를 담고 있지 않으면 기존 페이지는 무효 상태가 된다. 그렇기 때문에 플래시 메모리는 유효한 정보를 담고 있지 않더라도 클린한 페이지가 존재하지 않을 가능성이 있다. 따라서 블록을 삭제하여 무효한 페이지를 클린한 상태로 바꿔놓는 작업이 필요해진다.

임의의 블록에 대해, 필요한 경우 유효한 데이타를 위한 새로운 공간을 할당 받은 뒤 그곳으로 복사하고, 해당 블록을 삭제하여 블록내의 페이지를 모두 클린상태로 바꿔놓는 작업을 블록 클리닝(block cleaning)이라고 한다. 이러한 블록 클리닝은 유효한 정보를 관리하기위한 메타데이타의 변경을 유발하게 된다. 플래시 메모리의 삭제 연산은 그 수행시간이 매우 길기 때문에 블록 클리닝이 플래시 메모리 기반 파일시스템의 성능에 큰 영향을 끼치게 된다. 이는 응용의 수행 중 요구 삭제(on-demand erase)를 최소화 시켜야만 한다는 것을 의미한다. 따라서 플래시 메모리 기반 소프트웨어는 성능 향상을 위해 상기한 여섯 가지 특성을 모두 고려하여 설계 / 구현 되어야 한다.

3. 블록 클리닝 비용과 모델

3.1 성능 인자

플래시 메모리의 블록 클리닝은 두 가지 종류가 존재한다. 첫째, 유효한 페이지와 무효한 페이지가 한 개 블록 내에 섞여있는 경우이다. 이 경우에는 해당 블록에 삭제 연산을 수행하기 전에 유효한 페이지를 다른 블록에 복사해야 한다. 따라서 이러한 경우를 "복사-삭제 클리닝(copy-erase cleaning)"이라 부르도록 한다. 둘째, 한 블록 내에 무효한 페이지만 혹은 무효한 페이지와 클린 한 페이지만 존재하는 경우이다. 이 경우에는 한 번의 삭제 연산만 필요하다. "삭제 클리닝(erase-only cleaning)"이라 부른다.

복사-삭제 클리닝 에서는 블록 내에 유효한 페이지가 몇 개나 존재하느냐가 클리닝 비용을 결정하는 큰 요소가 된다. 삭제 클리닝 에서는 무효한 페이지의 분포와 비율이 블록 클리닝 비용을 결정하는 큰 요소가 된다. 따라서 본 논문에서는 블록 클리닝 비용에 영향을 주는 성능 인자로 이용률(Utilization), 무효율(Invalidity), 순수도(Uniformity)를 추출하였다. 각 인자는 다음과 같이 정의된다.

이용률(u) : 플래시 메모리에서 유효 상태 페이지의 비율

무효율(i) : 플래시 메모리에서 무효 상태 페이지의 비율

순수도(p) : 플래시 메모리에서 무효 상태 페이지의 분포. 즉, 순수한 블록은 유효한 페이지와 무효한 페이지를 같이 가지고 있지 않은 블록의 비율을 의미한다.

그림 4는 설명을 위해 총 5개의 블록이 있고 블록 당 4개씩의 페이지를 가지고 있는 플래시 메모리를 통해, 위에서 정의한 성능인자간의 관계를 나타낸다. 이용률 u와 무효율 i는 같고 순수도 p를 변화시켰을 때의 플래시 메모리의 상태를 보여주고 있다. 총 20개의 페이지 중 유효한 페이지와 무효한 페이지는 각각 8개씩 존재하고 있기 때문에 u와 i는 0.4이다. 그러나 (a), (b), (c)는 각각 1, 3, 5개의 순수한 블록을 가지고 있기 때문에 p는 0.2, 0.6, 1이 된다.

유효율은 복사-삭제 클리닝 작업 시 복사해야 하는 유효 페이지의 평균 개수를 결정한다. 무효율은 블록 클리닝의 대상이 되는 블록의 개수를 결정한다. 순수도는

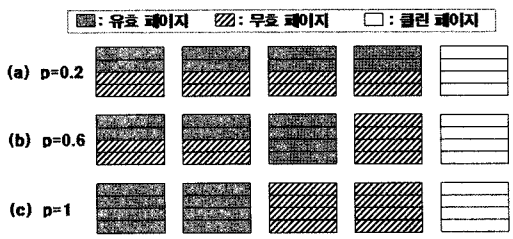

그림 4 성능 인자 설명 (u=i=0.4일 때) (a)p=0.2, (b)p=0.6 (c)p=1

전체 블록 클리닝 작업 중 부가적인 복사 오버헤드를 발생시키는 복사-삭제 클리닝 연산의 개수를 결정한다.

3.2 모델 수립

지금까지 관찰한 블록 클리닝 비용과 성능 인자 간의 관계를 수식으로 표현하면 식 (1)과 같다. 수식은 크게 2개의 항(term)으로 구성된다. 첫 번째 항은 복사-삭제 클리닝 비용이다. 이는 블록내의 유효한 페이지를 위한 복사 비용과, 해당 블록의 삭제 비용을 의미한다. 복사 대상이 되는 페이지는 읽기와 쓰기 두 가지 연산을 수행해야 하는데, 이때 읽기와 쓰기의 시간은 rt와 wt로 표현되고, 블록의 삭제 연산의 시간은 et로 표현되었다. 만약 플래시 메모리 내부에서 읽기와 쓰기를 통합한 재기록 연산을 지원하면[14] 복사 비용으로 이 연산의 시간을 사용할 수도 있다. 일반적으로 재기록 연산의 시간은 쓰기 연산의 수행시간과 유사하다. 해당 블록의 삭제 비용은 식 (2)에, 블록내의 유효한 페이지를 위한 복사 비용은 식 (3)에 나타내었다. 두 번째 항은 삭제 클리닝 비용이다. 삭제 클리닝 비용은 식 (4)에 나타내었다.

4. 모델 검증

4.1 실험 환경

본 논문에서 제안한 블록 클리닝 비용 모델을 검증하기 위해 표 1의 내장형 시스템 상에 실험환경을 구축하였다[12]. 이 시스템에는 삼성 K9F1208U0B NAND 플래시 메모리가 장착되어 있다. 또한 표 1의 하드웨어 상에 리눅스 커널이 동작하며, 본 연구에서는 커널 버전 2.4.19를 사용하였다. 그리고 YAFFS를 NAND 플래시 메모리를 관리하는 소프트웨어 컴포넌트로 사용하였다[15]. YAFFS는 VFS(Virtual File System)를 통해 open(), read(), write()등의 사용자 인터페이스를 제공하며, MTD의 readchunkfromnand(), writechunktonand(), eraseblockinnand()등의 인터페이스를 이용해 플래시 메모리를 제어한다[16].

표 1 내장형 시스템 하드웨어 명세

Hardware Component	Specification
CPU	400MHz XScale PXA255
RAM	64MB SDRAM
Flash memory	64MB NAND, 512KB NOR flash memory
Interface	RS232, USB, CS8900, JTAG
Others	LCD, Touch, LED

4.2 플래시 메모리 연산의 수행 속도 측정

식 (1)을 통해 블록 클리닝 비용을 예측하기 위해서는 플래시 메모리의 읽기(rt), 쓰기(wt), 삭제(et) 연산의 시간을 알고 있어야 한다. YAFFS는 Linux의 MTD (Memory Technology Device)가 제공하는 함수를 사용하여 플래시 메모리를 제어한다. 따라서 정확한 비용 예측을 위해서는 플래시 메모리의 데이타시트 상에 기록되어 있는 읽기, 쓰기, 삭제 연산 시간이 아닌 MTD 계층에서 제공하는 각 연산의 수행 시간을 정확하게 파악하는 것이 중요하다.

이를 위해 우선, 두 가지 조건하에서 수식에서 사용될 읽기(rt), 쓰기(wt), 삭제(et) 연산의 시간을 측정해 보았다. 첫째, YAFFS가 사용하는 MTD 계층에서 플래시 메모리의 세 가지 연산을 각각 1000회씩 수행하여 그 평균을 구하였다. 둘째, MTD 계층을 거치지 않고 NAND 플래시 메모리를 직접 제어하는 디바이스 드라이버를 제작하여 역시 각 연산을 1000회씩 수행하여 그 평균을 구하였으며, 이를 해당 플래시 메모리의 데이타시트에 나와 있는 수행시간과 비교하여 그림 5에 나타내었다.

블록 클리닝 비용
= 복사-삭제 클리닝 비용 + 삭제 클리닝 비용
= (순수하지 않은 블록 개수 * θ_t)+(순수하지 않은 블록내의 유효한 페이지 개수 * (r_t + w_t)) +
(무효 페이지만으로 구성된 블록 개수 * θ_t) (1)

r_t : 읽기 연산 수행시간
w_t: 쓰기 연산 수행시간
θ_t : 삭제 연산 수행시간

순수하지 않은 블록 개수 = (1 − p) * B (2)

순수하지 않은 블록내의 유효한 페이지 개수
= 순수하지 않은 블록 개수 * 블록내의 유효한 페이지 개수
= (1 − p) * B * (P/B) * (u/(u + i)) (3)

무효 페이지 만으로 구성된 블록 개수
= (무효한 페이지의 개수 − 순수하지 않은 블록내의 무효한 페이지 개수) / (블록당 페이지 개수)
= ((i * P) − ((1 − p) * B * (P/B) * (i/(u + i)))) / (P/B) (4)

u: 이용률(0 ≤ u ≤ 1)
i: 무효율 (0 ≤ i ≤ 1− u)
p: 순수도 (0 ≤ p ≤ 1)
P: 플래시 메모리내의 페이지 개수 (P= 용량/페이지 크기)
B: 플래시 메모리내의 블록 개수(P/B= 블록당 페이지의 개수)

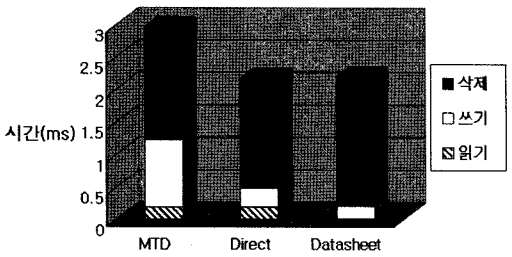

그림 5 계층별 플래시 메모리 각 연산의 수행 시간 비교

데이타시트에 나와 있는 rt, wt, et는 각각 0.01ms, 0.2ms, 2ms였고, 직접 제작한 디바이스 드라이버를 수행하여 측정된 시간은 0.19ms, 0.3ms, 1.7ms였으며, YAFFS가 사용하는 MTD 계층에서의 측정시간은 0.2ms, 1.03ms, 1.74ms였다. 추후 모델 검증을 위한 실험은 MTD 계층에서 제공하는 함수를 사용하는 YAFFS 내에서 이뤄지므로, 이후 수식에서는 MTD 계층에서 측정한 시간을 사용하도록 한다.

4.3 검증 도구

모델 검증을 위하여 이용률, 무효율 순수도를 설정할 수 있는 도구 FSST(Flash State Setting Tool)를 제작하였다. 설정을 원하는 세 가지 인자에 대해 0~100%까지 원하는 값을 입력받은 뒤, 입력된 값대로 플래시 메모리의 상태를 설정한다. 또한 실제 블록 클리닝을 수행하여 그 비용을 측정할 수 있는 도구 FBCT(Flash Block Cleaning Tool)역시 제작하였다. 이 두 가지 도구는 Proc 가상 파일시스템을 통해 YAFFS에 구현되었으며, 리눅스 커널의 모듈 형태로도 제작 가능하다. FSST와 FBCT를 사용하는 의사(pseudo) 진행과 이에 따른 플래시 메모리의 변화를 나타내면 아래 그림 6과 같다.

그림 6의 (a)는 총 5개의 블록이 있고, 각 블록에는 4개의 페이지가 있다고 가정한 플래시 메모리의 초기 상태를 나타낸다. (b)와 같이 FSST를 수행시키면서 u, i, p를 각각 20%, 20%, 60%로 설정을 하면, (c)에서 보이듯 플래시 메모리의 상태를 셋팅 한다. 그런 뒤 (d)에서

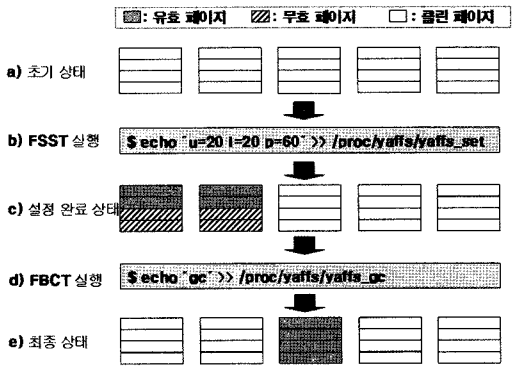

그림 6 FSST와 FBCT의 사용 예

보이는 바와 같이 FBCT를 수행 시키면 네 번의 페이지 복사 연산과 두 번의 삭제 연산을 수행하여, (e)에서 볼 수 있듯이 블록 클리닝을 완료하고, 블록 클리닝에 걸린 시간을 화면에 출력해준다.

4.4 검증 결과

FSST와 FBCT를 이용하여 모델을 검증한 결과는 그림 7과 같다. (a)그래프는 순수도와 무효율을 0.5즉 50%로 고정하고 이용률을 0.5~0.1까지 변화 시키면서 플래시 메모리의 상태를 설정 한 후, 블록 클리닝을 수행하여 그 비용을 시간 단위로 측정한 것이다. (b)그래프는 이용률과 무효율을 0.5로 고정하고 순수도를 0.5~0.9까지 변화시키면서, c)는 이용률과 순수도를 0.5로 고정 한 뒤 무효율을 0.5~0.1까지 변화 시키면서 블록 클리닝 비용을 측정한 결과이다. 각 그래프의 X축은 변화시키는 인자의 값을 의미하고 Y축은 블록 클리닝 비용을 ms(milli second)단위로 보여준다. 그림에서 검은 막대는 계산값을, 빗금 막대는 실제 측정값을 의미한다. 블록 클리닝 비용을 실제 측정해 본 결과 본 논문에서 제안한 모델은 실제 측정값과 평균 7%의 오차 범위 내에서 블록 클리닝 비용을 예측할 수 있었다.

5. 비용 분석

그림 8은 본 논문에서 제안한 모델로부터 얻어진 블

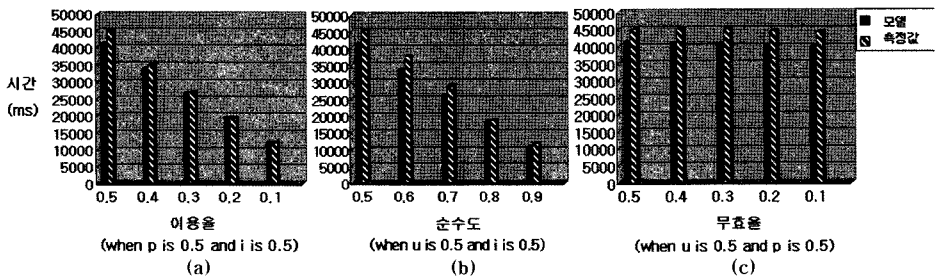

그림 7 FSST와 FBCT를 이용한 모델 검증 결과

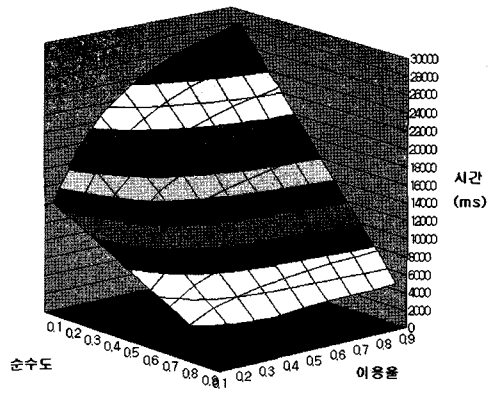

그림 8 블록 클리닝 비용과 성능 인자 관계

록 클리닝 비용에 대한 분석 결과이다. X축은 이용률을, Y축은 순수도를 의미하며, Z축은 블록 클리닝 비용을 의미한다. (a)는 무효율이 0.1일 때, (b)는 0.01일 때의 결과이다. 분석을 위한 기본 데이타는 5.1절에서 설명한 실험 환경과 동일한 자료를 사용하였다.

그림 8에서 우리는 다음과 같은 것을 파악할 수 있다. 첫째, 예상대로 이용률이 증가함에 따라 비용이 증가한다. 둘째, 순수도가 나빠짐에 따라 비용이 증가한다. 셋째, 무효율이 0.01일 때에 비해 무효율이 0.1이 되면 비용이 증가한다. 하지만 이용률과 순수도의 변화에 따른 비용의 변화는 무효율이 변하더라도 비교적 유사했다. 넷째, 이용률과 순수도 모두 나빠질 때 비용은 급격히 증가한다. 반면 이용률이나 순수도 중에서 하나를 좋은 상태로 유지할 수 있으면 비용이 급격히 증가하지 않는다. 그림 8의 분석내용은 그림 7의 검증 결과와도 같은 경향을 보여주고 있다.

다른 용량의 소블록 NAND 플래시 메모리와 대블록 NAND 플래시 메모리에 대하여 본 논문에서 제안한 수식을 이용하여 블록 클리닝 비용과 성능 인자 간의 관계를 분석한 결과는 그림 7과 8의 경향과 유사하다. 따라서 본 논문에서 제시된 수식이 다양한 용량의 플래시 메모리와 대블록 NAND 플래시 메모리의 분석에도 사용될 수 있음을 알 수 있다.

6. 순수도 기반 페이지 할당 기법

블록 클리닝 비용은 이용률, 무효율 그리고 순수도에 의해 영향을 받는다. 이때 무효율과 순수도는 파일 시스템 수준에서 제어 가능하다. 무효율의 경우 파일 시스템이 관리하는 버퍼 캐시에서 지연 쓰기(delayed write)를 이용하면 감소시킬 수 있다. 또한, 파일 시스템의 페이지 할당 기법과 블록 클리닝 기법을 이용하여 순수도를 제어할 수 있다. 그런데, 플래시 메모리를 주로 사용하는 휴대폰, MP3 플레이어, USB 저장장치 등 이동형 기기들은 갑작스러운 전원 결함을 감내할 수 있어야 한다. 따라서 지연 쓰기는 결함 복구 방법이 있는 제한된 범위에서 적용할 수 있다. 본 논문에서는 순수도를 향상시킬 수 있는 새로운 페이지 할당 기법을 제안한다.

기존의 플래시 메모리 파일 시스템은 페이지 할당을 위해 순차 할당(sequential allocation) 기법을 사용한다. 즉 새로운 페이지에 대한 요청이 발생하면 가용 페이지 공간에서 순서대로 페이지를 할당하는 것이다. 반면 본 논문에서 제안하는 기법은 데이타의 변경 특성을 고려한 할당(modification aware allocation) 기법이다. 이 기법의 기본 아이디어는 빈번하게 변경되는 데이타(hot data)와 그렇지 않은 데이타(cold data)를 구분하고, 이들을 서로 다른 블록에 할당하여 순수도를 향상시키겠다는 것이다. 빈번하게 변경되는 데이타(hot data)와 그렇지 않은 데이타(cold data)의 구분은 데이타 변경의 치우침 경향(skewness)을 이용하면 가능하다[5,6].

MODA에서는 데이타의 정적 속성(static property)과 동적 속성(dynamic property)을 이용하여 분류한다. 그림 9는 변경 특성을 고려한 데이타의 분류 방법을 보여준다. 일반적으로 파일 시스템의 수퍼 블록이나 inode 같은 메타데이타는 비교적 자주 접근되며, 이에 비해 사용자 데이타는 덜 자주 접근된다. 따라서 MODA의 정적 속성 분류에서는 데이타가 메타 데이타인지 아니면 사용자 데이타인지에 따라 분류를 한다. 파일 시스템이 아닌 데이타베이스의 경우에는 인덱스인지 아니면 데이타인지를 정적 속성으로 사용할 수도 있다.

다음으로 사용자 데이타는 동적 분류를 하게 된다. 동적 분류를 위해 MQ(Multi Queue)알고리즘[17]을 도입하였다. MQ는 복수개의 LRU(Least-Recently Used) 큐(Queue)를 사용하는 구조로써 각각은 Q0, Q1, ..., Qm-1처럼 번호로써 구분된다. 사용자 데이타는 각 큐에 주어진 생존시간(lifetime) 동안 머무르게 된다.

동적 속성 분류의 구체적인 내용은 다음과 같다. 처음

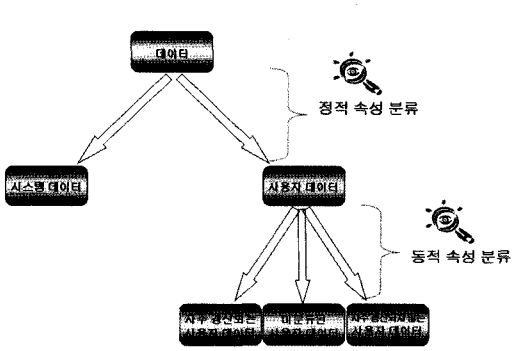

그림 9 변경 특성을 고려한 데이터 분류

데이타가 생성되면 우선 Q0에 삽입된다. 생존 시간 내에 수정되면 높은 번호의 큐로 이동되며, 만약 생존 시간 내에 수정되지 않는다면 하위 큐로 내려오게 된다. 따라서 빈번하게 접근되는 데이타는 Qm-1 큐에 존재하게 되며, 반대의 경우 Q0큐에 위치하게 된다. 실험에서 m은 2로, 생존 시간은 100으로 설정하였다. 따라서 2회 이상 접근 된 데이타는 빈번하게 접근되는 데이타로 분류되며, 생존 시간 100 이내에 접근 되지 않은 데이타는 빈번하게 접근 되지 않는 데이타로 분류됨을 의미한다.

결국 정적 분류는 파일시스템 수준에서 파일의 메타 데이타와 사용자 데이타를 분류하는 것이며, 사용자 데이타의 경우 동적 분류를 하는데 이때 사용자 데이타는 파일시스템 수준에서 블록 단위 정보와 해당 파일의 메타 데이타 정보를 얻을 수 있으므로 두 기법 모두 어떠한 파일 시스템에서도 구현가능 하다.

7. 성능 측정

7.1 벤치마크 프로그램

제안된 기법의 성능을 평가하기 위해 기존 플래시 메모리 관련 논문을 조사하였고, 그 결과 다음 7개의 벤치마크를 선정하였다.

- Camera benchmark : JPEG 파일의 일반적인 크기인 200~1000KB 사이의 파일을 생성하고, 생성한 파일에 대해 읽기 및 삭제 연산을 수행하는 벤치마크이다. 이러한 연산은 사진을 찍은 뒤 확인하고 지우는 등 일반적인 디지털 카메라의 동작특성을 반영한 것이다[18,19].
- Movie benchmark : PMP(Portable Media Players)의 동작 특성을 반영한 벤치마크로써, 15~30MB 사이의 파일을 생성하고 읽기/제거 연산을 수행한다[19].
- Phone benchmark : 휴대전화의 동작 특성을 반영한 벤치마크로써 1일 분량의 작업은 다음과 같다. 각각 3번의 SMS 수신/발신 기록, 각각 10번의 전화 송/수신 기록, 5개의 부재중 전화 기록, 각각 5개의 약속 정보 추가/제거 기록[20].
- Recorder benchmark : 블랙박스와 같은 이벤트 레코더(Event Recorder)의 동작 특성을 반영한 벤치마크 프로그램. 두 개의 파일에 대해 다른 주기로 반복하여 32byte의 데이타를 추가 한다[20].
- Fax machine benchmark : 51300bytes 크기의 파일 4개로 구성된 팩스 송/수신 데이타를 저장하는 벤치마크로써 각 송수신 동작 마다 로그를 남기는 작업을 수반한다[20].
- Postmark : Postmark는 작은 크기의 파일들을 생성하고, 읽기, 변경, 삭제 등의 트랜잭션을 수행한다[21]. 파일의 개수와 수행할 트랜잭션 개수는 사용자에 의해 설정될 수 있다(기본 설정은 500 트랜잭션).
- Andrew : 이 벤치마크는 디렉터리 생성, 파일 복사, 파일 상태 검사, 파일 내용 검사, 컴파일이라는 5단계로 구성된다[22]. 본 논문에서는 디렉터리 생성, 파일 복사라는 2단계만 수행하였으며, 벤치마크의 부하를 증가시키기 위해 생성 및 복사 횟수를 디폴트 설정보다 크게 하였다.

위에서 언급한 벤치마크 프로그램은 크게 세 가지로 구분 될 수 있다. 첫째, 순차 읽기/쓰기 위주의 부하(Workload), 둘째, 갱신 위주의 부하, 셋째, 복합적인 부하이다. Camera와 Movie 벤치마크는 대용량 파일을 다루는 순차 읽기/쓰기 위주의 부하이고, Phone, Recorder는 작은 개수의 파일을 생성한 뒤 반복적으로 갱신하는 갱신 위주의 부하이다. Fax, Postmark, Andrew 벤치마크는 복합적인 부하를 생성하는 벤치마크이다.

7.2 성능 측정 결과

표 2는 벤치마크를 수행하여 얻은 수행시간 및 블록 클리닝 결과를 보여준다. 블록 클리닝 시간은 실제 실험 결과 및 모델을 통해 예측한 계산 값 둘 다를 보여주고 있다.

구체적인 실험의 내용은 다음과 같다. 각각의 벤치마크를 수행하기 전, 플래시 메모리의 전체 블록에 대해 모두 삭제 연산을 수행함으로써 이용률을 0%로 설정하였다. 그런 뒤, 기존 YAFFS와 본 논문에서 제안한 MODA기법이 구현된 YAFFS(MODA-YAFFS) 두 개의 파일 시스템 상에서 벤치마크를 수행하였고, 소요된 수행시간을 'Benchmark Running Time'칸에 기록하였다. 벤치마크를 수행한 뒤엔 이용률, 무효율, 순수도를 측정하여 표의 U, I, P칸에 기록하였다. 이때 U, I, P 값은 파일 시스템을 실제 사용하기 전에는 예측 할 수 없는 값이며, 파일시스템을 일정 시간 사용하여 에이징(aging)시킨 이후에 얻을 수 있다. 따라서 측정된 세 가지 인자를 통해 본 논문에서 제안한 모델을 이용하여 블록 클리닝에 필요한 시간과 삭제, 복사 연산의 횟수를 예측하였고, 이를 '모델에 의한 계산값' 칸에 기록하였다. 그런 뒤 4.3에서 소개한 FBCT를 이용하여 실제 블록 클리닝을 수행하였고, 소요된 시간과 삭제, 복사 연산의 횟수를 측정하여 '측정값' 칸에 기록하였다.

표 2에서 볼 수 있듯이 본 논문에서 제안한 모델은 무효율이 낮은 경우 복사 연산의 횟수를 약간 크게 예측하는 경향이 있지만, 블록 클리닝을 수행하는데 드는 삭제와 복사 횟수를 비교적 정확하게 예측할 수 있었다. 그러나 블록 클리닝 시간의 경우 모델에 의한 계산 값과 실제 측정값 사이에 차이가 큰데 그 이유는 다음과 같다. YAFFS 파일시스템은 RAM상의 점유율을 최소화하기 위해 블록과 페이지의 현재 상태를 별도로 유지

표 2 기존 YAFFS와 MODA간의 성능 비교

벤치마크 종류	정책	벤치마크 수행시간	성능 인자			모델에 의한 계산값			측정값		
			U	I	P	삭제 횟수	복사 횟수	블록 클리닝 시간	삭제 횟수	복사 횟수	블록 클리닝 시간
Camera	YAFFS	37	0.3	0.0006	0.98	62	1916	2.46	60	1504	9.97
	MODA	37	0.3	0.0006	0.99	7	159	0.20	5	62	7.58
Movie	YAFFS	564	0.99	0.0001	0.99	10	319	0.41	10	17	24.64
	MODA	564	0.99	0.0001	0.99	1	31	0.04	1	3	24.54
Phone	YAFFS	88	0.05	0.32	0.62	1398	6047	9.87	1398	6047	13.40
	MODA	88	0.05	0.22	0.72	1010	6052	9.20	1011	6063	12.08
Recorder	YAFFS	31	0.005	0.16	0.83	626	692	1.94	626	699	2.17
	MODA	31	0.005	0.08	0.90	233	692	1.44	344	690	1.91
Fax machine	YAFFS	97	0.86	0.0087	0.73	1023	31710	40.78	1001	30996	67.50
	MODA	97	0.86	0.0087	0.97	107	2407	3.14	76	1383	23.47
Postmark	YAFFS	16	0.08	0.0107	0.90	357	10158	13.11	357	10147	18.39
	MODA	16	0.08	0.0107	0.93	260	7057	9.13	248	6652	12.84
Andrew	YAFFS	33	0.09	0.0008	0.90	348	10174	13.12	345	10060	18.51
	MODA	32	0.09	0.0008	0.98	87	1828	2.40	62	1004	3.86

하지 않는다. 따라서 블록 클리닝이 수행되는 시점에 삭제연산이 필요한 블록을 탐색하기 위한 시간과, 블록내의 유효한 페이지를 찾기 위한 오버헤드가 발생된다. 그러므로 이용률이 커질수록, 또한 무효율이 작아질수록 검색을 위한 오버헤드 역시 커지게 된다. 이러한 경향은 표 2에 잘 나타나 있다. 그러나 모델에 의한 계산 값의 변화 경향과 실제 측정값의 변화 경향은 매우 유사하므로, 본 논문에서 제안한 모델이 플래시 메모리 기반 소프트웨어의 성능을 정성적으로 예측할 수 있는 도구임을 알 수 있다.

7.3 이용률의 영향 측정

실제 플래시 메모리를 사용하는 기기의 경우 이용률이 0이 되는 경우는 매우 드물다. 따라서 표 2에서 수행한 실험을 다양한 이용률 하에서 진행하여 이용률이 플래시 메모리 기반 소프트웨어의 성능에 미치는 영향을 확인 해 보았다.

그림 10은 Andrew 벤치마크를 이용하여 다양한 이용률을 생성해 놓은 뒤, Postmark 벤치마크를 수행한 결과를 보이고 있다. 그림에서 우리는 다음 2가지를 발견할 수 있다. 첫째, 이용률이 증가함에 따라 블록 클리닝의 비용이 증가한다. 둘째, 이용률이 증가함에 따라 새로운 YAFFS와 기존 YAFFS에서 블록 클리닝의 성능 차가 점점 커진다. 이는 이용률이 증가함에도 제안된 페이지 할당 기법이 순수도를 좋은 상태로 유지하여 블록 클리닝 비용의 급격한 증가를 효과적으로 방지하고 있음을 보여준다.

한 가지 흥미로운 것은 이용률이 거의 100%가 되도록 벤치마크를 수행 시켰을 때 성능결과인데 이는 그림 10에서 점선 사각형으로 둘러싸여 있는 부분이다. 이 부분은 이용률 90%의 상태에서 Postmark(트랜잭션 500개)를 수행한 결과이며, 이때에는 벤치마크 수행 중에

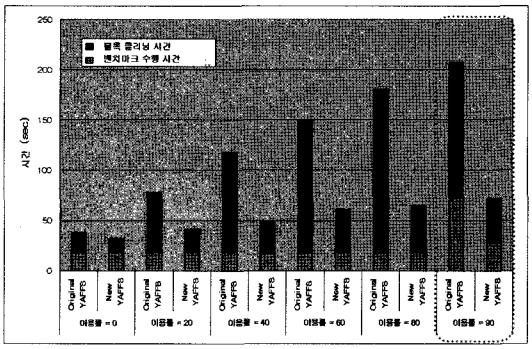

그림 10 다양한 이용률에서의 벤치마크 수행 결과

블록내의 페이지가 클린한 상태인 블록이 부족하게 되어 YAFFS의 디폴트 블록 클리닝이 aggressive mode로 동작한다. 따라서 응용 수행 중에 블록 클리닝이 수행되는 요구 블록 클리닝(on-demand block cleaning)이 많이 발생한다. 그 결과 벤치마크의 수행 시간이 크게 증가한다. 구체적으로 기존 YAFFS의 경우 80%의 이용률에서는 수행 시간이 18초인데, 90%의 이용률에서는 수행 시간이 72초로 증가하였다. 반면 새로운 YAFFS의 경우 수행 시간이 27초로 증가하였다. 요구 블록 클리닝 횟수를 측정해본 결과 기존 YAFFS에서는 1257회, 새로운 YAFFS에서는 349회 발생한 것으로 파악되었다. 결국 이 결과는 제안된 블록 클리닝 기법이 순수도를 증가시켜 블록 클리닝 비용 자체를 감소시킬 수 있을 뿐만 아니라 요구 블록 클리닝 횟수를 감소시켜 응용의 수행 시간도 단축시킬 수 있음을 보여준다.

7.4 MODA vs DAC

본 논문에서 제안한 MODA 기법과 기존 관련 연구와의 성능 비교를 위해 Chiang et al.[6]의 DAC(Dynamic dAta Clustering)기법을 YAFFS에 구현하였다.

DAC기법은 플래시 메모리를 복수개의 구역(region)으로 구분하였고, 갱신 요청이 올 때 페이지를 상위 구역으로 이동시키며 접근이 없는 경우 하위 구역으로 이동시킨다. 구역의 개수는 가장 성능이 좋다고 보고된 4개로 설정하였다[6].

그림 11은 기존 YAFFS와 DAC, MODA간의 성능 비교 결과를 보여준다. 표 2에 기술된 벤치마크를 각각의 파일시스템 상에서 수행하였다. 실험 결과 MODA 기법은 DAC 기법보다 좋은 성능을 보이고 있었다. MODA 기법이 DAC 기법보다 좋은 성능을 보이는 이유는 다음과 같다. 첫째, DAC 기법에서는 접근 빈도가 낮은 데이타를 하위 구역에 위치시킨다. 따라서 새로 생성된 데이타와 빈번하게 접근되지 않는 데이타가 같은 블록에 존재할 수 있게 되고, 그러므로 순수도는 나빠지게 된다. MODA에서는 새로 생성된 데이타를 unclassified manager를 통해 관리함으로써 순수도를 높게 유지시키고 있다. 둘째, MODA는 DAC와 다르게 데이타를 분류하기 위해 2단계 분류기법을 사용하고 있다. 정적 분류와 동적 분류로 구성된 MODA의 기법은 DAC보다 순수도를 높게 유지시킬 수 있다. 따라서 DAC기법에 데이타의 정적 분류 기능을 추가 구현하여 실험하면 보다 좋은 성능을 얻을 수 있었다. 이는 데이타의 정적 분류가 순수도를 좋게 유지시키는데 매우 효율적임을 의미한다.

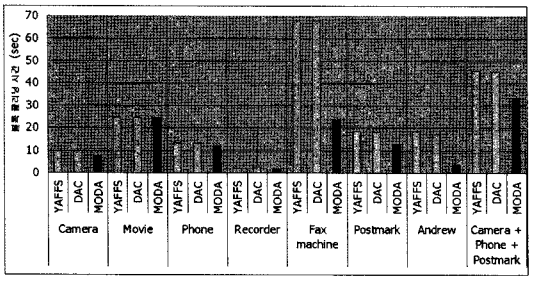

그림 11 YAFFS, DAC, MODA간의 성능 비교

8. 관련 연구

로그 구조 파일 시스템은 디스크를 추가 전용(append only) 저장 장치로 간주한다. 따라서 수정 요청을 덮어쓰기가 아닌 새로운 위치에 쓰기와 기존 데이타 무효화로 서비스하며, 무효화된 데이타는 세그먼트 클리닝(Segment Cleaning)을 통해 재생한다[1]. 이러한 기법은 플래시 메모리의 특성에 부합되기 때문에 많은 플래시 메모리 기반 소프트웨어에서 사용되고 있다. 세그먼트 클리닝 성능이 로그 구조 파일 시스템의 성능에 중요한 영향을 끼침에 따라 이를 개선하려는 많은 기법이 제안되었다[2-5]. 참고문헌 [3]에서는 요구 클리닝과 백그라운드 클리닝을 구분하고, 휴리스틱을 이용한 유휴 시간 발견 및 백그라운드 클리닝을 통해 사용자가 느끼는 클리닝 비용의 감소를 시도하였다. 참고문헌 [2]에서는 디스크의 이용률에 따라 클리닝과 홀 플러깅(hole plugging)을 적응력 있게 적용하는 방법과 캐시를 이용한 클리닝 성능 향상 기법을 제안하였다. 참고문헌 [4]와 [5]는 본 연구와 유사하게 변경 횟수를 고려하여 클리닝의 성능 향상을 시도한 연구들이다. 참고문헌 [4]에서는 데이타의 변경 빈도가 높은 데이타들과 그렇지 않은 데이타를 모아 각각 서로 다른 세그먼트에 한 번에 기록함으로써 클리닝 비용을 줄였고, 참고문헌 [5]에서는 변경 빈도가 높은 데이타는 non in-place update를 수행하고 그렇지 않은 데이타는 in-place update를 수행하여 클리닝 비용을 줄였다. 하지만 참고문헌 [4]와 참고문헌 [5]는 디스크를 대상으로 하기 때문에 플래시 메모리를 대상으로 하는 본 연구와는 다르다. 우선 참고문헌 [5]는 덮어 쓰기 제한이 있는 플래시 메모리에서는 적용할 수 없다. 그리고 참고문헌 [4]는 디스크의 탐색 거리를 고려하기 위해 2개의 세그먼트를 선택하여 한 곳에는 변경 빈도가 낮은 데이타를 다른 곳에는 변경 빈도가 높은 데이타를 저장하였으며, 저장할 때에는 세그먼트 크기만큼 데이타를 모아 한 번에 기록하였다. 하지만 플래시 메모리의 경우에는 탐색 거리를 고려할 필요가 없어 여러 블록들을 동시에 사용할 수 있으며, 특정 블록에 기록할 때에도 블록 크기만큼 데이타를 모아 한 번에 기록하는 것이 아닌 일부분만 점차적으로 기록할 수 있다.

플래시 메모리를 고려한 효과적인 블록 클리닝에 대한 연구도 많이 수행되었다[6-8,23]. 참고문헌 [23]에서는 소거 평준화에 대한 고려는 없었지만, 새로 쓰이는 데이타와 클리닝에 의해 복사되는 데이타를 서로 다른 블록에 쓰는 기법(separate segment for cleaning)을 제안하였다. 참고문헌 [7]에서는 삭제 색인을 유지하여 쓰기 분포가 균등할 때 FIFO 알고리즘과 치우쳐 있을 때 locality gathering 알고리즘을 함께 사용하는 혼성 클리닝(hybrid cleaning) 기법을 제안하였다. 이때 locality gathering은 클리닝할 때 유효한 데이타를 높은 번호의 블록으로 이동시키고, 새로 쓰여진 데이타를 낮은 번호의 블록으로 이동시킴으로써 궁극적으로 자주 변경되는 데이타와 그렇지 않는 데이타를 다른 블록으로 모으는 방법이다. 이 두 연구는 블록 클리닝 시점에서만 데이타를 분류하고 관리한다는 측면과 실제 분류하는 방법에서 본 연구와 차이가 있다. [24]에서는 교대영역, 예비영역 등 플래시 메모리를 여러 개의 영역으로 구분한 뒤 블록의 소거 횟수가 특정 횟수에 다다른 경우 영

역 이동을 수행하여 전체적인 소거 횟수를 평준화 시킬 수 있는 방법을 제안하였다. 참고문헌 [6,25]에서는 CAT (Cost Age Time)과 DAC(Dynamic dAta Clustering) 기법을 제안하였다. CAT는 블록 클리닝 비용, 데이타가 쓰여진 이후 지난 시간, 마지막으로 삭제를 수행한 시간을 모두 고려하여 클리닝할 블록을 선택한다. 한편, DAC는 CAT를 개선한 것으로, 플래시 메모리를 몇 개의 구역(region)으로 구분하고 변경 빈도에 따라 데이타를 특정 구역에 할당한다. 결과적으로 이 방법은 데이타를 쓸 때 변경 빈도에 따라 서로 다른 블록에 할당하는 것으로 본 논문의 접근 방법과 유사하다. 하지만 본 연구는 실제 플래시 메모리 상에서 구현하고 성능을 평가하였으며, 순수도(Uniformity)라는 측정 가능한 성능 인자를 제안하고 이를 기반으로 블록 클리닝 비용을 예측하였다는 점에서 참고문헌 [6,25] 연구와 차이가 있다.

사실 본 연구는 기존 연구[26]를 확장 발전한 것이다. [26]과 본 연구의 차이점은 첫째, 모델의 수정을 통해 보다 높은 정밀도를 가지게 되었고, 둘째, 실제 구현을 통해 모델을 검증하였다. 셋째, [26]에서는 성능측정을 위해 Postmark이나 Andrew등 하드디스크 기반의 벤치마크를 주로 사용하였으나 본 연구에서는 플래시 메모리를 위한 벤치마크가 대폭 추가 되었다. 넷째, 관련 선행 연구인 [6]을 YAFFS에 구현하여 본 논문에서 제안한 MODA기법과 비교 실험함으로써 제안된 기법이 별도의 오버헤드를 초래하지 않음을 보였다.

9. 결론 및 향후 연구 방향

본 논문에서는 플래시 메모리의 특징을 기술하고, 플래시 메모리 기반 파일시스템의 성능에 영향을 주는 최대 병목 요소로써 블록 클리닝을 정의하였다. 또한 플래시 메모리의 특징에 기반 한 관찰을 통해 블록 클리닝 비용에 영향을 끼치는 성능인자 세 가지-이용률, 무효율, 순수도-를 도출하였다. 이 세 가지 인자를 통해 블록 클리닝 비용을 예측 할 수 있는 모델을 제안 하였다. 또한 제안한 모델을 검증하기 위해 플래시 메모리의 상태를 설정 할 수 있는 도구(FSST)와, 실제 블록 클리닝을 수행하여 그 비용을 시간 단위로 측정 할 수 있는 도구(FBCT)를 제작 하였다. 제작한 도구를 통해 64MB의 플래시 메모리와 이를 위한 YAFFS가 구동되고 있는 내장형 보드 상에서 블록 클리닝 비용을 실제로 측정하여, 제안한 모델의 계산 값과 비교하였다. 실험 결과 평균 7%의 오차 범위 내에서 블록 클리닝 비용을 예측할 수 있었다. 이는 블록 클리닝 수행 시 필요한 비용이 본 논문에서 제안한 모델을 통해 계산 가능함을 의미한다. 따라서 플래시 메모리 기반 소프트웨어의 성능을 정성적으로 측정 할 수 있으며, 또한 플래시 메모리를 위한 효율적인 블록 클리닝 정책의 설계가 가능해진다.

또한 순수도를 향상시키는 새로운 페이지 할당 정책을 제안하였다. 실험 결과 기존 YAFFS나 관련 연구보다 블록 클리닝 시간을 평균 123% 단축 시켰으며, 이용률이 100%가 되도록 벤치마크를 수행시키면 제안된 기법이 순수도를 증가시켜 블록 클리닝 비용 자체를 감소시킬 수 있을 뿐만 아니라 요구 블록 클리닝 횟수를 감소시켜 응용의 수행 시간도 단축시킬 수 있음을 보여준다.

향후 본 연구는 다음 두 가지 방향으로 확장될 것이다. 첫째, 본 논문에서 제안된 수식을 이용하여 효율적인 블록 클리닝 정책을 설계하는 것이다. 블록 클리닝이 수행된 뒤에는 플래시 메모리의 순수도가 변경된다. 따라서 블록 클리닝이 수행되는 시점에 순수도를 고려하여, 주어진 시간 내에 가장 높은 이득을 낼 수 있는 블록 클리닝 정책을 설계할 수 있을 것이다. 둘째, 마모도 평준화를 고려하는 할당 정책의 설계이다. 빈번하게 접근 되는 데이타를 위한 블록을 할당하는 경우와, 빈번하게 접근되지 않는 데이타가 존재하고 있는 블록을 블록 클리닝 할 때 둘 간의 상관관계를 이용하면 별도의 오버헤드 없이도 효율적으로 마모도 평준화를 얻을 수 있을 것이다.

참 고 문 헌

[1] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," ACM Transactions on Computer Systems, Vol.10, No.1, pp. 26-52, 1992.

[2] J. Matthews, D. Roselli, A. Costello, R. Wang, and T. anderson, "Improving the performance of log-structured file system with adaptive methods," ACM Symposiums on Operating System Principles (SOSP), pp. 238-251, 1997.

[3] T. Blackwell, J. Harris, and M. Selzer, "Heuristic cleaning algorithms in log-structured file systems," Proceedings of the 1995 Annual Technical Conference, pp. 277-288, 1993.

[4] J. Wang and Y. Hu, "WOLF - a novel reordering write buffer to boost the performance of log-structured file system," Proceedings of the USENIX Conference on File and Storage Technologies (FAST), pp. 46-60, 2002.

[5] W. Wang, Y. Zhao, and R. Bunt, "HyLog: A High Performance Approach to Managing Disk Layout," Proceedings of the USENIX Conference on File and Storage Technologies (FAST), pp. 145-158, 2004.

[6] M-L. Chiang, P. C. H. Lee, and R-C. Chang, "Using data clustering to improve cleaning performance for flash memory," Software: Practice

and Experience, Vol.29, No.3, pp. 267-290, 1999.
[7] M. Wu and W. Zwaenepoel, "eNVy: a non-volatile, main memory storage system," Proceeding of the 6th international conference on Architectural Support for Programming languages and operation systems, pp. 86-97, 1994.
[8] L. P. Chang, T. W. Kuo and S. W. Lo, "Real-time garbage collection for flash memory storage systems of real time embedded systems," ACM Transactions on Embedded Computing Systems, Vol.3, No.4, pp. 837-863, 2004.
[9] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories," ACM Computing Surveys, Vol.37, No.2, pp. 138-163, 2005.
[10] M. Mckusick, W. Joy, S. Leffler, and R. Fabry, "A Fast File System for UNIX," ACM Transactions on Computer Systems, 2(3), pp. 181-197, Aug., 1984.
[11] William Stalling, "Operating Systems: Internals and Design Principles," 5th Edition, Pearson Prentice Hall, 2004.
[12] EZ-X5, FALINUX Inc., http://www.falinux.com/zproducts/ez-x5.php
[13] Samsung Inc., http://www.samsung.com/Products/Semoconductor/NANDFlash
[14] Samsung Inc., http://www.samsung.com/Products/Semiconductor/NANDFlash/SLC_LargeBlock/32Gbit/K9NBG08U5M/ds_k9xxg08uxm_rev10.pdf
[15] Aleph One, "YAFFS: Yet Another Flash File System," http://www.aleph1.co.uk/yaffs/
[16] MTD subsystem for Linux, http://www.linux-mtd.infradead.org/archive/index.html
[17] Y. Zhou, P. M. Chen, and K. Li, "The Multi-Queue Replacement Algorithm for Second-Level Buffer Cache," Proceeding of the 2001 USENIX Annual Technical Conference, June, 2001.
[18] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, Y. Cho, "A space-efficient Flash translation layer for CompactFlash systems," IEEE Transactions on Consumer Electronics, Vol.48, No.2, pp. 366-375, 2002.
[19] H. Jo, J. Kang, S. Park, J. Kim, and J. Lee, "FAB: Flash-Aware Buffer Management Poicy for Portable Media Players," IEEE Transactions on Consumer Electronics, Vol.52, No.2, May, 2006.
[20] E. Gal and S. Toledo, "A Transactions Flash File System for Microcontrollers," Proceedings of the 2005 USENIX Annual Technical Conference, pp. 89-104, 2005.
[21] J. Katcher, "PostMark: A New File System Benchmark," Technical Report TR3002, Network Appliance Inc., 1997.
[22] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, M. J. West, "Scale and Performance in a distributed file system," ACM Transactions on Computer Systems, 6(1), Feb., 1988.
[23] Kawaguchi, S. Nishioka and H. Motoda, "A flash-memory based file system," Proceedings of the 1995 USENIX Annual Technical Conference, pp. 155-164, 1995.
[24] 배영현, 최종무, 이동희, 노삼혁, 민상렬, "플래시 메모리 파일 시스템을 위한 효율적인 소거 평준화 기법", 한국정보과학회 가을 학술발표논문집, pp. 580-582, 2004.
[25] M. L. Chiang and R. C. Chang, "Cleaning Policies in Mobile Computers using Flash Memory," Journal of systems and software, pp. 213-231, Vol. 48, No, 3, 1999.
[26] 백승재, 최종무, "플래시 메모리 파일 시스템을 위한 순수도 기반 페이지 할당 기법에 대한 연구," 정보처리학회논문지 A 제 13-A권 제 5호, pp. 387-398, 2006.

백 승 재

2005년 단국대학교 컴퓨터 공학과 졸업(공학사). 2004년~현재 비트 컴퓨터 강사. 2007년 단국대학교 대학원 정보컴퓨터 과학과(이학석사). 2007년~현재 단국대학교 대학원 정보컴퓨터 과학과 박사과정. 관심분야는 운영체제, 내장형 시스템

최 종 무

1993년 서울대학교 해양학과 졸업(이학사). 1995년 서울대학교 대학원 컴퓨터 공학과(공학석사). 2001년 서울대학교 대학원 컴퓨터 공학과(공학박사). 2001년~2003년 유비쿼스 주식회사 책임 연구원, 2003년~현재 단국대학교 정보컴퓨터학부 컴퓨터과학 전공 조교수. 2005년~2006년 UC Santa Cruz 방문 교수. 관심분야는 운영체제, 내장형 시스템, 고성능 저장 장치 등