신뢰성 있는 플래시메모리 저장시스템 구축을 위한 플래시메모리 저장 공간 관리 방법

(New Flash Memory Management Method for Reliable Flash Storage Systems)

김 한 준 † 이 상 구 ††
(Han-joon Kim) (Sang-goo Lee)

요 약 본 논문은 로그파일시스템의 원리를 바탕으로 플래시메모리 저장시스템에 적합한 플래시메모리 공간 관리 방법을 제안한다. 플래시메모리는 비휘발성, 빠른 입출력 속도 등의 장점을 지니고 있지만, 제자리 덮어쓰기(in-place update)가 불가능하고 메모리 셀에 대한 쓰기(write) 횟수가 제한되는 단점이 있다. 이러한 특성은 플래시메모리를 저장 매체로 사용할 때 기존의 저장 매체 관리 방법과는 다른 방법을 요구하게 된다. 본 논문은 자유 공간유지를 위해 필요한 클리닝 메커니즘의 연산 비용을 낮추면서 동시에 저장공간이 전체적으로 균등하게 사용될 수 있도록 하는 사이클 평준화 기법을 제안한다. 제안된 방법은 특히 메모리 활용도와 로컬리티가 높을 때 좋은 성능을 보인다. 클리닝 메커니즘은 자주 접근되지 않는 COLD데이타를 그렇지 않은 데이타와 격리시킴으로써 그 효율이 향상되었으며, 사이클 평준화는 사용 횟수간의 최대 차이가 하드웨어적인 오차를 벗어나지 않는 수준까지 이루었다. 실험을 통해 제안된 방법은 비교 기준으로 삼은 직관적 방법(greedy policy)에 비해 사이클 평준화가 잘 이루어진 상태에서 최대 35%정도의 클리닝 비용 절감 효과를 보였다.

Abstract We propose a new way of managing flash memory space for flash memory-specific file system based on log-structured file system. Flash memory has attractive features such as non-volatility, and fast I/O speed, but it also suffers from inability to update in place and limited usage cycles. These drawbacks require many changes to conventional storage (file) management techniques. Our focus is on lowering cleaning cost and evenly utilizing flash memory cells while maintaining a balance between the two often-conflicting goals. The proposed cleaning method performs well especially when storage utilization and the degree of locality are high. The cleaning efficiency is enhanced by dynamically separating cold data and non-cold data. The second goal, cycle-leveling is achieved to the degree where the maximum difference between erase cycles is below the error range of the hardware. Simulation results show that the proposed method has significant benefit over naxve methods: maximum of 35% reduction in cleaning cost with even spreading writes across segments.

1. 서 론

플래시메모리는 기존의 기억장치인 하드디스크를 대체할 것으로 주목받고 있는 차세대 기억장치로서 집적회로로 구성된 비휘발성 메모리이다. 최근 플래시메모리의 집적도와 입출력 성능이 계속 개선되고 있어 이동 컴퓨터의 보조 기억 장치로 뿐만 아니라 일반 컴퓨팅 시스템의 저장 매체로까지 사용할 수 있게 되었다[4,13]. 플래시메모리는 전기적으로 입출력을 수행하기 때문에 읽기 성능은 D램과 거의 같은 수준이며 쓰기 속도도 하드디스크에 비해 수백 배 이상 빠르다[1][14]. 그리고 이 저장 매체는 같은 크기의 D램보다 그 크기가 약 30%

† 비 회 원 : 서울대학교 컴퓨터공학부
hjkim@europa.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
sglee@mars.snu.ac.kr
논문접수 : 1999년 3월 25일
심사완료 : 2000년 4월 21일

1) 플래시메모리의 읽기, 쓰기 연산 시간은 각각 바이트당 약 85-100 ns, 4-10 μs가 소요된다.

작으면서도 내충격성과 고장률 측면에서 훨씬 우수하기 때문에 대용량화가 가능하다. 또한 플래시메모리에 존재하는 데이타는 주기억장치로 로드될 필요 없이 직접 플래시메모리 공간상에서 입출력이 가능하다. 이러한 특성 때문에 플래시메모리가 기존의 디스크 기반 저장시스템의 저장 매체를 대체하는 경우에 시스템의 입출력 성능을 획기적으로 향상시킬 수 있다.

그러나 플래시메모리를 저장시스템의 매체로 사용하기 위해서 극복해야 할 두 가지 단점이 있다. 첫째, 플래시메모리는 디스크 매체와는 달리 저장 공간상에 존재하는 데이타를 수정할 때 본래 주소에서 바로 덮어쓰는 것(in-place update)이 불가능하다. 이는 플래시메모리가 쓰기(갱신) 연산을 수행할 때 각 비트(또는 바이트)가 한쪽 방향으로만 토글링(toggling)을 허락하기 때문이다. 그래서 플래시메모리는 메모리 셀 상에 데이타가 쓰여지기 전에 미리 '0' 또는 '1'로 초기화(erase)[2]하는 연산이 필요하다. 결국 쓰기(갱신)연산을 수행하기 위해서는 초기화된 저장 공간을 미리 확보하고 있어야 한다. 둘째, 플래시메모리는 각 메모리 블록에 대해 수행되는 초기화 연산의 횟수가 제한되어 있다[3]. 허용되는 초기화 횟수를 초과한 플래시메모리 블록에는 쓰기 오류(write failure)가 발생한다. 그래서 전체 플래시메모리 공간이 균등하게 활용되지 못하는 경우에는 사용 가능한 저장 공간이 급격히 줄어드는 현상이 발생하게 된다. 결국 이 '초기화 횟수 제한' 문제는 플래시메모리가 신뢰성 있는 저장시스템을 위한 저장 매체로 사용되는데 큰 장애 요소가 된다.

플래시메모리에 기반한 저장시스템(이하 플래시 저장시스템)을 구축하기 위해서는 전술한 두 가지 단점을 해결할 수 있는 메커니즘이 필요하다. 첫 번째 단점(제자리 덮어쓰기의 불가능)을 해결하기 위해서는 데이타 갱신 방법과 자유 공간 유지 방법이 관건이 된다. 데이타를 갱신하기 위해서는 우선 다른 주소 공간에 새로운 데이타를 쓰게 하고 원래 데이타는 무효화(invalidation) 시키는 방법을 생각할 수 있다. 그리고 사용되는 전체 메모리 크기는 한정되어 있기 때문에 지속적으로 자유 공간을 확보하기 위해서는 무효화된 데이타가 있는 블록들에 대해 '가베지 컬렉션(garbage collection)' 또는 '클리닝 연산(cleaning operation)'을 수행해주어야 한다. 이때 이 연산에 소요되는 비용을 최소화하도록 해야 한다. 그리고 두 번째 단점(초기화 횟수 제한)은 가능한 한 모든 플래시메모리 블록들을 균등하게 활용(초기화)하도록 하는 '사이클 평준화(cycle leveling)' 작업을 요구한다. 이는 플래시 저장시스템의 수명(lifetime) 또는 신뢰도를 결정짓는 요소이며 유지되는 플래시메모리 공간의 크기가 클수록 사이클 평준화의 중요성은 더욱 커진다[4]. 일반적으로 사이클 평준화를 위해서는 자주 접근되는 데이타를 별로 사용되지 않은 메모리 블록에 둔다. 그래서 어떤 메모리 블록이 지나치게 활용되었다고 판단이 되면, 그곳에 존재하는 데이타들을 덜 초기화 된 영역으로 이동시키는 것이 필요하다. 이때 데이타가 다른 주소의 블록으로 복사되면서 본래 데이타가 존재했던 블록들은 무효화 될 것이다.

결과적으로 플래시 저장시스템을 구축하기 위해 기본이 되는 것은 클리닝 연산과 사이클 평준화와 관련된 이슈들을 해결하는 것이라고 볼 수 있다. 그런데 이 두 가지 목적을 동시에 충족시키기는 쉽지 않다. 만약 플래시메모리 관리가 클리닝 연산의 목적에만 치중할 경우에 저장 공간의 쓰임새가 한쪽으로 치우치게 될 경향이 있어 시스템의 신뢰도가 떨어질 수가 있고, 반면 사이클 평준화에 치중하는 경우에는 많은 무효블록들을 생성시키기 때문에 클리닝 연산에 소요되는 비용을 증가시키게 된다.

본 논문의 목적은 신뢰성 있는 플래시 저장시스템의 구축을 위해서 상충되는 두 가지 목표를 조화시킬 수 있는 클리닝 기법과 사이클 평준화 기법을 제시하는 것이다. 이를 위해 본 논문에서는 플래시메모리 공간 관리의 기본 전략으로서 로그구조파일시스템(log-structured file system)[10]에서 제안된 로깅 방법을 사용한다. 로깅에서는 쓰기 연산이 '로그(log)'라는 자료 구조상에서 붙여쓰기(append)를 통해 실현된다. 그리고 로그구조파일시스템의 클리닝 메커니즘을 기반으로 클리닝 비용을 줄이면서 사이클 평준화를 하기 위한 메커니즘을 제안한다. 여기에서 클리닝 메커니즘의 성능은 로그 상에 퍼져 있는 자주 접근되지 않는 COLD데이타를 수집하는 '컬렉션(collection)' 연산에 의해 개선되며, 사이클 평준화 기능은 클리닝되는 세그먼트를 선택하기 위해 고안된 '클리닝 지표'를 통해 클리닝 메커니즘에 통합된다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 기술하고, 3장에서는 로깅 방법에 의해서 자유 메모리 공간이 관리되는 방법과 이와 연관된 본 논문의 구체적인 목표를 기술한다. 4장에서는 제시된 목표를 달성하기 위해 컬렉션 연산과 사이클 평준화를 결합한 여러 가지 플래시메모리 관리 방법을 소개한다. 5장에서는 제안된

2) 초기화 연산은 일반적인 읽기, 쓰기 연산보다 훨씬 많은 시간(약 0.6-0.8초)이 소요된다.

3) 초기화 횟수는 일반적으로 100,000번 정도로 제한되어 있다. 최근에 1,000,000번의 횟수를 갖는 제품도 출시되고 있다.

알고리즘의 효능을 실험을 통해 입증한다. 마지막으로 6장에서 결론을 맺는다.

2. 관련 연구

플래시 저장시스템과 관련된 과거 연구는 대부분 클리닝 연산에 소요되는 비용을 최소화하는데 중점을 두고 있다[4,5,7,13]. 클리닝 메커니즘의 성능을 높이기 위한 한가지 해결책은 접근이 별로 없는 COLD데이타와 그렇지 않은 NON-COLD데이타를 섞이지 않도록 하는 것이다. 이를 위해 [13]은 자주 수정되는 HOT세그먼트의 클리닝 비용을 줄이기 위해 로컬리티의 정도에 따라 데이타를 정렬하는 로컬리티 수집기법(locality gathering)을 사용하였으며, [7]은 COLD데이타가 NON-COLD데이타와 섞이지 않도록 하기 위해 COLD 세그먼트를 클리닝하기 위한 별도의 세그먼트를 사용하였다. 그리고 디스크 기반의 로그구조파일시스템은 버퍼 안에서 갱신된 유효 페이지들을 디스크에 방출하기 전에 나이(age)순으로 정렬하는 나이정렬기법(age-sorting)을 제안하였다[8,10]. 이런 격리 방법들은 그 특성상 데이타 입출력이 발생할 때마다 COLD데이타와 NON-COLD데이타를 분리하는데 항상 신경을 써야 하는 오버헤드를 가지게 된다. 본 연구에서는 이런 오버헤드를 줄이기 위해 COLD데이타의 분산된 정도를 측정하여 필요하다고 판단되는 시점에만 격리 작업을 수행하도록 한다.

그리고 사이클 평준화를 위해 제안된 기존의 방법들은 플립플롭(flip-flop)의 성격을 가지는 간단한 형태이다. 예를 들어, 세그먼트의 초기화 횟수 값들의 최대 차가 주어진 임계값을 초과하였을 때에 최대 초기화 횟수를 가지는 블록들과 최소 초기화 횟수를 가지는 블록들에 존재하는 데이타를 서로 교환하는 방법(A) [13]이 있다. 이와 유사하게 자유 공간 풀(pool)에서 최대 초기화 횟수를 갖는 블록을 할당받아 이곳으로 최소 초기화 횟수를 가지는 블록의 데이타를 이동시키는 방법(B) [4]이 있다. 이런 방법들은 사이클 평준화 동안에 많은 시스템 자원을 소모하게 되어 일반적인 데이타 입출력 연산을 크게 방해할 수 있다. 즉, (A)의 경우에 두 메모리 블록들간에 데이타 교환을 위해서 세 번의 데이타 이동(복사)작업이 이루어져야하며, 교환되는 데이타 크기만큼의 시스템 버퍼가 필요하다. 더구나 교환되는 데이타가 쓰여지기 위해서 미리 상대 메모리 블록에 대해 초기화 연산을 수행하는데 이 초기화 연산과 데이타 교환(이동) 작업이 병렬적으로 수행될 수가 없다. 본 연구에서는 사이클 평준화로 인한 성능 저하를 줄이기 위해 클리닝 작업을 수행할 때 동시에 사이클 평준화를 고려하게 된다.

3. 플래시메모리 관리

서론에서 언급한 바와 같이 본 연구에서는 플래시메모리 공간 관리를 위해 로그구조파일시스템의 로깅방법을 채택하였다. 기존의 여러 연구에서도 플래시메모리에 특성에 맞는 저장시스템의 구현을 위해 로깅방법이 추천되어 왔다[4,7,12]. 플래시 저장시스템이 로깅 기법을 사용하는 이유는 로그구조파일시스템의 경우와는 다르다. 로그구조파일시스템에서의 로깅은 쓰기버퍼(write buffer)에 누적된 데이타를 한꺼번에 디스크로 방출함으로써 쓰기 성능을 높이는데 기여하지만, 플래시 저장시스템에서 그런 효과를 기대하지는 않는다. 플래시메모리 저장시스템에서 로깅 기법을 채택하는 것은 그것이 플래시메모리의 단점들을 해결할 수 있는 여러 성질을 가지고 있기 때문이다. 즉, 로깅에서는 항상 쓰기(갱신) 연산이 로그 말단인 새로운 메모리 공간에서 발생하기 때문에 '제자리 덮어쓰기 불가능 문제'가 자연스럽게 해결되고, 더불어 다른 초기화된 유효 메모리 공간을 연속적으로 사용할 수 있어 약간의 사이클 평준화 효과를 기대할 수 있다. 또한 플래시 저장시스템은 그 매체의 하드웨어 특성상초기화 연산을 통해서 새로운 저장 공간을 창출해야 하기 때문에 로그파일시스템에서 제안된 클리닝 메커니즘(또는 클리너)의 원리를 응용할 수 있다.

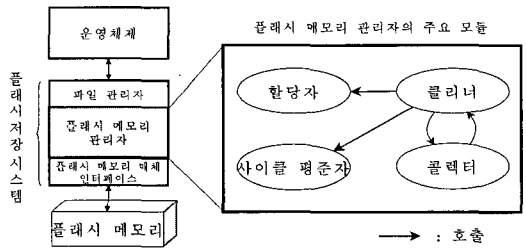


그림 1 플래시 저장시스템

본 논문에서는 로그 구조를 생성하기 위해 플래시메모리 공간을 '세그먼트' 단위로 구분한다. 각 세그먼트는 한번의 초기화 명령으로 한꺼번에 같이 초기화되는 '초기화 블록(erase block)'[4]들의 집합체이다. 그리고 각

4) 초기화 블록은 플래시메모리 칩의 구성에서 한번에 초기화할

세그먼트는 일정한 수의 페이지들로 구성된다. 이때 '페이지'라 함은 플래시메모리와 주기억장치간의 이동되는 데이타의 크기인 입출력 단위를 말한다.

여기에서 소개하는 '세그먼트'는 로그구조파일시스템에서 클리닝 연산의 단위가 되는 세그먼트와 유사하다. 그런데 플래시메모리는 초기화되는 메모리 블록의 수에 상관없이 일정한 시간이 걸리기 때문에 세그먼트의 크기는 충분히 큰 것이 좋다[5]. 그래서 플래시 저장시스템에서 사용되는 세그먼트는 로그구조파일시스템의 세그먼트보다 상대적으로 매우 크다[6].

본 장에서는 플래시 저장시스템의 프로토타입을 제시하고 이를 바탕으로 플래시메모리 공간 관리의 구체적인 목표를 기술한다.

3.1 시스템 구조

본 논문이 가정하는 플래시 저장시스템은 그림 1과 같이 파일 관리자(file manager), 플래시메모리 관리자(flash memory manager), 플래시 매체 인터페이스(flash media interface)로 구성된다. '파일 관리자'는 논리적 페이지들의 집합으로 파일을 관리하는 요소로서 운영체제가 요구하는 일반적인 파일 서비스를 제공한다. 그리고 '플래시메모리 관리자'는 플래시메모리를 물리적인 수준에서 관리하여 논리적인 페이지의 물리적인 위치를 결정해서 필요한 플래시 입출력 연산을 발생시킨다. 이는 디스크 기반 저장시스템에서의 디스크 관리자(disk manager)에 비유할 수 있다. 마지막으로 '플래시 매체 인터페이스'는 플래시메모리 하드웨어를 위한 저수준 연산들의 집합으로 구성된다. 이는 초기화(erasing), 프로그래밍(programming), 에러추적(error detecting) 연산 등을 포함한다.

본 논문은 플래시 저장시스템의 세 가지 요소 중 '플래시메모리 관리자'와 관련된 내용을 다루게 된다. 제안되는 플래시메모리 관리자는 클리너(cleaner), 할당자(allocator), 사이클 평준자(cycle-leveler), 컬렉터(collector) 등 네 개의 모듈로 구성된다. '클리너'는 새로운 자유공간을 창출하기 위해 로그 공간에 존재하는 무효 페이지들을 초기화해서 시스템에 반환하는 모듈이다. '할당자'는 클리너가 반환한 자유 세그먼트를 로그에 할당함으로써 지속적으로 로그 공간을 유지하게 한다. '사이클 평준자'는 플래시 세그먼트들이 균등하게 사이클 횟수를 가지도록 한다. 그리고 '컬렉터'는 로그 상에서 COLD데이타를 NON-COLD데이타로부터 격리시키는 모듈로서 클리너의 오버헤드를 줄이는데 기여한다.

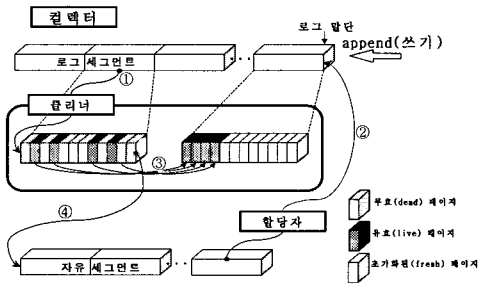


그림 2 플래시메모리 관리자

그림 2는 플래시메모리 관리자가 동작하는 절차를 보여준다. 어떤 데이타가 갱신(쓰기)되면 이는 로그 말단에 있는 새로운 페이지에 쓰여진다. (이때 그 새로운 페이지는 '유효(live)'로 표시되고, 이전 데이타의 페이지는 '무효(dead)'로 표시된다.) 이런 로깅 과정이 반복될 때 많은 무효 페이지들이 발생하게 되어 가용 메모리 공간의 크기가 점차 적어진다. 클리너는 남아 있는 가용 공간의 크기가 임계점 이하로 떨어질 때 구동된다. 여기서 클리닝 연산은 '유효데이타 복사', '세그먼트 초기화'의 두 단계를 거친다. 유효 데이타 복사 단계에서는 클리너가 로그 상에 존재하는 세그먼트들 중에서 클리닝 연산의 대상(이를 '클리닝 세그먼트'라고 부르겠다)을 결정하고(그림 2 ①), 자유 세그먼트들 중에서 필요한 만큼의 세그먼트를 로그에 할당한다(그림 2 ②). 이어서 클리닝 세그먼트로부터 유효 블록들을 수집하여 로그 말단에 이를 복사한다(그림 2 ③). 지금까지의 과정은 로그구조 파일시스템의 클리닝 연산과정과 유사하다. 세그먼트 초기화 단계에서는 선택된 세그먼트의 모든 초기화 블록에 대해 초기화 연산이 병렬적으로 수행된다. 이렇게 해서 클리닝된 세그먼트는 자유 세그먼트 풀(pool)로 반환된다(그림 2 ④). 세그먼트 초기화 단계는 단지 일정한 비용을 지불하는 하드웨어적인 초기화 연산만으로 구성되기 때문에 클리너의 성능은 주로 유효 데이타 복사 단계에서 결정된다. 그리고 유효데이타 복사 단계에서

수 있는 메모리 단위를 말한다. 대개 그 크기가 4-64킬로바이트이며, 이는 입출력 단위의 페이지와 구별되는 것이다.

5) 초기화 단위의 크기는 클리닝 비용을 고려해서 결정되어야 한다. 그 크기가 너무 크다면 클리닝 비용이 커지며, 반대로 너무 작게 되면 잦은 클리닝으로 인해 많은 수의 초기화 연산이 발생된다. 최적의 세그먼트 크기를 결정하는 문제는 본 연구의 범위를 벗어난다.

6) 디스크 기반 로그구조파일시스템인 Sprite-LFS[10]에서는 512킬로바이트, 플래시 저장시스템. eNVy[13]에서는 16메가바이트를 사용한다.

할당자 모듈이 어떤 자유 세그먼트를 로그에 할당할 지에 따라 클리너와 사이클 평준자의 성능에 영향을 미칠 수 있다(4.2.2절에서 설명).

또한 클리너는 클리닝할 때마다 세그먼트들이 골고루 사용되고 있는지를 검사한다. 만약 사이클의 편중화가 주어진 기준보다 심해질 때 클리너는 사이클 평준자를 구동시킨다. 사이클 평준자는 최소 사이클 횟수를 가지는 세그먼트의 데이타를 로그 말단으로 이동(복사)시킨다. 이 연산은 이동되는 데이타가 원래 있던 페이지들은 무효화시키기 때문에 관련 세그먼트가 클리닝 될 가능성을 높이게 된다.

위의 과정에 따라 많은 횟수의 로깅과 클리닝 연산이 진행되면 로그 공간에는 자주 접근되지 않는 COLD데이타와 그렇지 않은 데이타가 섞이게 되는데 이는 클리닝 효율을 떨어뜨리는 주요 원인이 된다(4.1절에서 설명). 그래서 클리너는 COLD데이타의 분산을 억제하기 위해 주기적으로 컬렉터를 호출하여 적당한 량의 COLD데이타를 NON-COLD데이타로부터 분리시킨다.

3.2 문제 정의 및 성능 측정치 소개

본 절에서는 플래시메모리 관리의 정량적인 분석을 위해 클리닝 비용, 사이클 평준화 정도, 그리고 시스템의 수명과 관련된 척도를 제안하고자 한다. 그리고 이를 토대로 본 논문이 제안하는 플래시메모리 관리의 목표를 기술한다.

본 논문에서 제안하는 클리닝 비용 모델은 플래시메모리 기반의 eNVy시스템[13]에서 정의된 클리닝 비용 모델에 기반을 둔다. 여기에서 μ를 세그먼트 활용률(전체 공간에서 유효데이타가 차지하는 비율)이라고 할 때 클리닝 비용을 $\mu/(1-\mu)$로 정의한다. 즉, 클리닝 비용은 새롭게 생성되는 공간($1-\mu$)에 대해서 복사(이동)되어야 하는 유효 데이타의 양(μ)을 의미한다. 비교해서 디스크기반의 Sprite-LFS시스템[10]은 클리닝 비용을 $2/(1-\mu)$로 정의한다. 이 식에서 분자 2는 읽기 비용(1)과 쓰기 비용(1)의 합을 나타내는데 디스크를 매체로 하는 시스템에서는 클리닝을 위해 세그먼트 전체에 대하여 읽기, 쓰기 연산을 수행함을 반영한 것이다. 이에 반해서 eNVy의 경우에는 한 세그먼트 내에서 유효 페이지들만을 복사하는 비용만을 고려하였다. 플래시메모리는 디스크처럼 탐색시간, 회전지연이 없을 뿐만 아니라 읽기 속도가 쓰기 속도에 비해 매우 빠르기 때문에 세그먼트를 읽어오는 비용은 무시할 수 있는 것이다.

그런데 이러한 클리닝 비용 모델은 플래시메모리가 시간과 비용이 많이 소요되는 초기화 연산을 가지고 있음을 전혀 고려하지 않은 것이다. 초기화 연산이 클리닝할 때마다 일정한 비용을 부담하지만 클리닝 연산이 많이 발생한다면 전체적인 비용이 커질 수밖에 없다. 본 논문에서는 eNVy[13]에서 제안된 클리닝 비용에 클리닝 회수 정보를 가미한 '누적 클리닝 비용'을 정의한다. 결국 클리너는 각 클리닝 연산의 비용을 최소화해야할 뿐만 아니라 전체 클리닝 연산의 횟수를 줄일 수 있어야 한다.

정의 1. **누적 클리닝 비용**은 $\sum_{i=1}^{f}\frac{\mu_i}{1-\mu_i}$이다. 이때 f는 시스템에 주어진 일정량의 쓰기 요청 스트림(write request stream)에 대해 수행된 클리닝 연산의 횟수를 의미하고, μ_i는 i 번째 클리닝을 위해 선택된 세그먼트의 활용률을 의미한다.

그리고 전체 세그먼트들의 사이클 횟수가 얼마나 균등하게 분포되었는지를 측정하는 척도로서 다음과 같은 사이클 평준화도(cycle leveling degree)를 제안한다.

정의 2. **사이클 평준화도**(Δ_e)는 최대 초기화 횟수(ε_{max})와 최소 초기화 횟수(ε_{min})의 차이다.

사이클 평준화도의 척도로서 모든 세그먼트들의 초기화 횟수값에 대한 분산값 또는 표준편차값 등과 같은 통계량을 사용하지 않은 것은 시스템의 수명이 사이클을 가장 많이 소비한 세그먼트의 수명에 의해 결정되기 때문이다.

일반적으로 사이클 평준화는 완벽하게(사이클 평준화도가 0에 가깝도록) 수행될 필요는 없다. 이는 초기화 횟수 제한이 세그먼트마다 약간의 오차를 가지고 있으며[7] 또한 과도한 사이클 평준화는 클리너의 성능을 떨어뜨리기 때문이다. 그래서 사이클 평준화의 목표를 사이클 평준화도가 초기화 제한 횟수의 오차 범위 내에 있는 것으로 삼는다.

마지막으로 시스템의 수명(lifetime)을 측정하기 위한 척도를 제안한다. 우선 주어진 플래시메모리 공간에서 허용 가능한 쓰기 연산의 최대량을 플래시 저장시스템이 갖게 되는 쓰기 용량(write capacity)으로 정의한다. 이를 '플래시 용량(flash capacity)'이라고 명명한다. 이 값은 고장(failure)시점까지 수행된 세그먼트 초기화 연산의 총 횟수에 비례할 것이다. 이때 고장시점은 초기화 횟수의 한도를 초과한 메모리 공간의 비율이 임계점을 넘어섰을 때를 말한다. 따라서 시스템 고장 때까지 사용

7) 이 오차는 측정이 어렵기 때문에 아직 보고되지 않고 있다. 본 논문에서는 이 오차범위가 특정한 값 e (예를 들어 100)를 갖는다고 가정한다.

된 플래시 용량이 시스템의 수명을 결정한다고 볼 수 있다. 이를 '사용 플래시 용량'으로 정의한다.

정의 3. 시점 t 에서 **사용 플래시 용량**은 $\sum_{i=1}^{N_{seg}} \epsilon_t(S_i)$이다. 이때 N_{seg}은 시스템 내에서 사용되는 세그먼트의 수이고, $\epsilon_t(S_i)$은 시점 t 에서 세그먼트 S_i의 초기화 횟수이다.

정의 3의 측면에서 우리가 목표로 하는 사이클 평준화는 고장 시점 t_f 에서 모든 세그먼트에 대한 $\epsilon_{t_f}(S_i)$ 값들이 초기화 제한 횟수의 오차 범위 내에 있게 하는 것이다.

이와 관련하여 플래시 저장시스템의 수명을 연장시키는 방법은 사이클 평준화를 이루면서 동시에 클리닝 횟수를 줄이는 것이다. 가령 사이클 평준화를 수행하지 않은 경우에 시스템이 사이클 용량을 a (segments · erases) 만큼을 소비하였을 때 고장이 발생하고, 사이클 평준화를 수행한 경우에는 사이클 용량을 b (segments · erases) 만큼을 소비하였을 때 고장이 발생한다고 가정하자. 그러면 사이클 평준화를 하였을때 사용플래시 용량은 b/a 비율만큼 증가한 것이고 그만큼 수명이 늘어난다고 할 수 있다. 게다가 클리닝 횟수가 f 만큼의 비율로 감소하였다면 시스템의 수명은 $\frac{b}{a\times(1-f)}$의 비율로 증가하게 될 것이다.

4. 플래시메모리 관리의 최적화

4.1 그리디 방법의 한계

클리닝 연산을 요구하는 시스템의 성능은 대개 그것의 클리닝 알고리즘에 의해 좌우된다. 그리고 앞서 소개된 클리닝 비용 모델에 따르면 클리닝 비용은 클리닝 세그먼트의 활용률(μ)에 의해 결정된다. 그래서 어떤 세그먼트를 클리닝 할 것인지가 클리닝 알고리즘의 가장 중요한 이슈가 된다. 직관적으로 생각할 수 있는 것은 클리닝 세그먼트로서 로그 상에서 가장 작은 활용률을 가지는 세그먼트를 선택하는 것이다. [10]에서는 이를 그리디 방법(greedy policy)이라고 한다.(여기서는 Greedy I 로 명명한다) 그러나 이 그리디 방법은 로컬리티 정도가 높을 때는 그 성능이 좋지 못하다[7,9,10,13]. 특히 세그먼트 활용률이 높을 때는 로컬리티가 클리닝 성능에 더 크게 악영향을 미친다. 이것은 클리닝 연산 과정에서 로그 공간에 분산된 COLD데이타가 더 이상 무효화되지 않으면서 필요 없이 여러 세그먼트로 옮겨 다니기 때문인데 로컬리티가 클수록 이러한 성향이 강해진다. 뿐만 아니라 클리닝 횟수도 증가하게 된다. 이는 증가된 COLD데이타로 인해 활용률이 커진 세그먼트가 쓰기(갱신)연산에 의해 보다 빠른 속도로 가용 공간이 소모되어 클리닝 연산을 많이 발생시키기 때문이다.

4.2 COLD데이타의 분리

위와 같은 그리디 방법의 제약점을 극복하기 위해서 본 논문은 COLD데이타를 NON-COLD데이타로부터 격리시키는 컬렉션(collection) 연산을 제안한다. 실제적으로 이는 플래시메모리 관리자의 컬렉터로 구현되며 이는 로그 상에 단편화된 COLD데이타를 강제적으로 수집해서 한 곳으로 모으는 작업을 수행한다. 이렇게 함으로써 COLD데이타가 수집된 세그먼트는 COLD데이타의 특성으로 인해 다른 세그먼트들에 비해서 무효화가 더디게 진행 될 것이며, 상대적으로 활용률이 작아진 다른 세그먼트들을 클리닝시키는 효과를 가지게 된다.

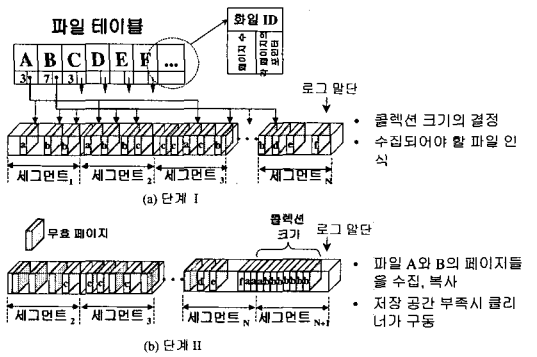


그림 3 컬렉션 연산 절차

그림 3은 컬렉션 연산의 두 단계 절차를 보여준다. (여기서 로그 말단에 기록되는 페이지는 영어 대문자로 표시된 파일 식별자의 소문자로 표시된다.) 우선 단계 I 에서 적당한 시점에 컬렉터가 호출되어 수집할 COLD 데이타의 양을 결정한 후에 COLD데이타 중에서 단편화가 많이 진행된 것을 찾는다. 이때 COLD데이타로 인식된 파일들이 단편화된 정도가 크지 않다면 컬렉션 작업을 수행할 필요는 없다. 그림 3의 예에서는 수집할 COLD데이타의 양으로 10이 주어졌으며 많이 단편화된 COLD데이타로 파일 A, B가 선택되었다. 단계 II에서는 수집된 COLD데이타의 페이지들이 로그 말단에 연속적으로 복사된다. 이때 가용 공간이 충분하지 못하다면 클리너가 구동되어 새로운 저장공간이 생성되어야 한다.

위 컬렉션 연산 과정에서 우리는 다음 세 가지 이슈에 대하여 합당한 정책을 가지고 있어야 한다. (컬렉터 모듈은 클리닝 연산이 n번 수행될 때마다 주기적으로 한번씩 호출되는데 이때 n 값을 '컬렉션 주기'라 부르고, 수집되는 COLD데이타를 '컬렉션 데이타', 수집되는 COLD데이타의 양을 '컬렉션 크기'라고 부르기로 한다.)

ㄱ. 어떤 COLD데이타를 수집해야 하는지? (컬렉션 데이타의 판정)

ㄴ. 언제 COLD데이타를 수집해야 하는지? (컬렉션 시점(주기)의 결정)

ㄷ. 얼마만큼의 COLD데이타를 수집해야 하는지? (컬렉션 크기의 결정)

ㄱ 과 관련해서, 컬렉션 데이타는 COLD데이타 중에서 로그 공간 전체에 많이 분산되어있는 것들이어야 한다. 이를 위해 COLD데이타를 판정하기 위한 기준과 그것의 단편화된 정도를 측정하는 것이 필요하다. ㄴ과 ㄷ은 컬렉터의 성능을 결정짓는 요인이 된다. 컬렉터가 너무 자주 호출되거나 또는 지나치게 많은 수의 페이지들을 수집하는 경우에는 과다한 페이지 이동으로 인해 전체 클리닝 비용을 오히려 상승시킬 수 있다.

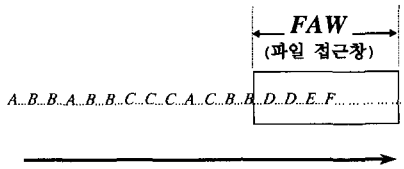


그림 4 파일접근창 (File Access Window)

4.2.1 COLD데이타의 판정

COLD데이타를 판정하기 위해서 컬렉터는 파일이 접근되는 패턴을 파악하는 '파일접근윈도우(File Access Window, 이하 FAW)'라는 데이타 구조를 유지한다. 이는 그림 4에서 보는 바와 같이 가장 최근에 접근된 파일들의 리스트를 보관하는 큐(queue)의 형태를 가진다. 이 FAW는 가상 메모리 공간 관리를 위해 제안된 working set[3]과 유사하다. Working set의 원리에 따르면 로컬리티가 존재할 때 working set에 해당하는 윈도우 크기만큼의 프로세스 시간동안 같은 페이지가 반복적으로 접근되는 경향이 있다. 이와 유사하게 최근에 갱신되고 단편화되는 파일들은 FAW내에 있다고 가정하고 FAW내에 포함되지 않은 '파일'들을 COLD로 판정한다. 이때 FAW의 크기는 보다 정확하게 COLD데이타를 가려낼 수 있도록 잘 조정해주어야 한다. 여기서는 최적의 클리닝 성능을 가질 때 FAW가 COLD데이타를 가장 잘 가려낸 것으로 간주해서 그때의 값을 FAW의 크기로 정한다. 이렇게 FAW를 통해 얻은 컬렉션 데이타의 후보 중에서 단편화가 많이 진행된 것을 컬렉션 데이타로 정하게 된다.

여기서 컬렉터는 파일 관리자(그림 1 참조)가 유지하는 파일 테이블[8](그림 3 참조)을 이용한다. 컬렉터가 페이지 수준이 아닌 파일 수준에서 COLD데이타를 판정하는 이유는 컬렉션 데이타는 많이 단편화된 COLD데이타이어야 하는데 페이지 수준에서 단편화된 정도를 가리는 것이 불가능하기 때문이다.

4.2.2 COLD데이타의 단편도 측정

위에서 언급한 바와 같이 컬렉션 데이타를 판정하기 위해서 최종적으로 단편화된 정도인 단편도를 측정해야 하는데, 이는 한 파일이 가지는 페이지들이 로그상에 얼마나 퍼져 있는지를 나타낸다. 그래서 단편도 F는 다음과 같이 정의하며 결과적으로 0과 1사이의 값을 가진다.

$$F = \frac{s}{p} \quad (\text{만약, } p > L_{seg} \text{이면 } p = L_{seg}) \tag{1}$$

여기서 p는 검사되는 파일이 점유하는 페이지들의 수이고, s는 그 파일이 분포되어 있는 세그먼트들의 개수를 의미한다. L_{seg}는 로그 상에 존재하는 세그먼트들의 개수이다. 여기에서 s는 L_{seg}보다 클 수 없기 때문에 크기가 큰 파일은 많이 단편화되었다 하더라도 작은 F 값을 가지게 된다. 그래서 파일의 페이지 수 p가 L_{seg}보다 큰 경우에는 p값을 L_{seg}로 대체한다. 이렇게 하면 같은 s값에 대해서 크기가 큰 파일의 단편도 값이 상대적으로 크기가 작은 파일의 단편도 값보다 작아지는 것을 방지할 수 있다.

4.2.3 컬렉션 시점과 컬렉션 크기의 결정

기본적으로 컬렉션 연산이 수행되는 최적의 시점은 COLD데이타가 여러 세그먼트로 쪼개져서 클리닝 비용을 증가시키는 원인으로 작용하기 시작하는 시점이 될 것이다. 그리고 알맞은 컬렉션 데이타의 양은 COLD데이타 중에서 실제로 크게 단편화된 데이타의 총량일 것으로 예측한다. 그런데 이 컬렉션 시점(또는 주기)과 컬렉션 크기를 정하는 것은 매우 방대한 탐색 공간을 가지는 문제이다. 그래서 본 연구에서는 컬렉션 연산의 구동 시점과 그것의 작업량을 세그먼트 활용률과 로컬리티에 따라 결정하는 휴리스틱을 사용한다. 즉, 식 (2),

8) 파일과 관련된 페이지 정보를 가지는 테이블을 말한다.

(3)에서 보는 바와 같이 컬렉션 크기(*ColSize*)는 활용률과 로컬리티에 비례하게 하고, 컬렉션 주기(*ColPeriod*)는 활용률과 로컬리티에 반비례하도록 한다. 여기에서 *ColSize*의 단위는 페이지 수, *ColPeriod*의 단위는 클리닝 횟수이다.

$$ColSize = k_s \cdot \mu_{avg} \cdot locality_f \quad (2)$$

$$ColPeriod = \frac{k_p}{\mu_{avg} \cdot locality_f} \quad (3)$$

여기서 μ_{avg}는 모든 세그먼트들에 대한 세그먼트 활용률의 평균값이며 $locality_f$는 로컬리티 정도를 나타낸다. k_s와 k_p는 비례상수로서 최적의 클리닝 효율을 가지도록 실험적으로 조정된 값을 취한다.

이와 같이 컬렉션 크기와 컬렉션 주기를 결정하기 위해 세그먼트 활용률(μ_{avg})을 고려한 것은 세그먼트 활용률이 높을수록 더 많은 COLD데이타를 생성하기 때문이며 이런 경향은 로컬리티가 높을수록 더욱 커지기 때문이다[13]. 또한 COLD데이타는 로컬리티가 높을수록 필요 없이 다른 세그먼트로 이동하는 경향이 강해진다. 그래서 세그먼트 활용률이 높을수록 그리고 로컬리티가 높을수록 보다 자주, 보다 많은 양의 COLD데이타를 수집할 필요가 있다. 이외에 다른 인자들을 포함해서 보다 정밀한 방법으로 컬렉션 주기와 컬렉션 크기를 정할 수도 있을 것이다.

식 (2), (3)의 계산을 위해서 로컬리티를 정량화 하는 것이 필요한데, 이를 위해 본 논문에서는 최근에 입출력 연산이 집중된 파일 집합이 얼마나 큰지를 측정한다. 그래서 앞서 소개된 FAW를 이용하여 로컬리티를 $\frac{|FAW|}{|FAW|_f}$로 계산할 수 있다. 여기에서 $|FAW|$는 파일접근윈도우의 크기이며 $|FAW|_f$는 FAW내에 존재하는 최근에 접근된 각 파일들의 수를 나타낸다. 이 식이 0과 1 사이의 값을 갖도록 다음과 같이 변형한다.

$$locality_f = \frac{|FAW| - |FAW|_f}{|FAW| - 1} \quad (4)$$

만약 $|FAW|$ 크기만큼의 입출력 연산이 진행되는 동안 단 하나의 파일만이 접근된다면 (즉, $|FAW|_f = 1$) $locality_f$는 최대값 1이 된다.

본 절에서 설명한 COLD데이타의 컬렉션 기법과 Greedy I 과 결합한 것을 CICL I [9]로 부르기로 한다.

4.2.4 클리닝 비용의 수정

컬렉션 연산도 클리닝 연산과 마찬가지로 유효 블록을 복사하는 작업을 수행하기 때문에 이것을 정의 1(3.2절 참조)의 누적클리닝 비용식에 반영해 주어야 한다. 그래서 한번의 클리닝에 해당하는 비용식($\frac{\mu}{1-\mu}$)을 다음과 같이 수정한다.

클리닝 비용 = 순수 클리닝 비용 + 컬렉션 비용

$$= \frac{\mu}{1-\mu} + \frac{c}{1-\mu} = \frac{\mu + c}{1-\mu} \quad (5)$$

$$\left(c = \frac{ColSize}{ColPeriod} \times \frac{1}{SegSize} \right) \quad (6)$$

식(5)에서 μ는 클리닝되는 세그먼트의 활용률을 나타내며 $c/(1-\mu)$은 컬렉션 연산에 소요된 비용을 나타낸다. 이 컬렉션 비용은 한 세그먼트에서 $(1-\mu)$ 비율만큼의 자유공간을 생성하기 위해 c 비율만큼의 유효 블록이 복사된 것을 나타낸다. 그래서 수정된 클리닝 비용식은 이 컬렉션 비용을 정의 1의 클리닝 비용(순수 클리닝 비용)에 더한 것이 된다. 컬렉션 비용값은 클리닝 시마다 컬렉션 연산이 수행된다고 가정할 때(실제로는 컬렉션 주기만큼의 클리닝 연산이 발생하고 나서야 한 번의 컬렉션 연산이 수행된다) 요구되는 비용으로 설명할 수 있다. 그래서 c 값은 식(6)과 같이 계산된다. 여기서 *SegSize*는 한 세그먼트에 포함된 페이지들의 수를 의미한다. 예를 들어, *SegSize*가 500페이지, 컬렉션 주기가 100, 그리고 한번의 컬렉션을 위해서 1,000페이지가 복사된다고 할 때 c 값은 $\frac{1,000}{100} \times \frac{1}{500} = 0.02$가 된다. 즉, 100번의 클리닝 연산이 발생하는 동안에 매번 0.02 만큼의 컬렉션 연산 비용을 추가적으로 보태주게 된다.

4.3 사이클 평준화

플래시 저장시스템에서 데이타 접근이 어떤 유형으로 발생하든지 사이클 사용이 한쪽으로 치우치게 마련이다. 최악의 경우에 전혀 접근되지 않은 데이타가 장시간 존재할 때는 극심한 세그먼트 편중화 현상이 발생한다. 본 논문이 플래시메모리 관리를 위해 채택한 로깅전략이 잠재적으로 사이클 평준화 효과를 가진다 할지라도 신뢰성 있는 플래시 저장시스템의 구축을 위해서는 2절에서 기술한 기존의 사이클 평준화 기법보다 안정적인 방법이 요구된다. 특히 플래시메모리 공간이 크고 쓰기 연산이 빈번한 환경에서 사이클 평준화의 중요성은 더욱 커진다.

9) CICL은 본 연구의 핵심 부분인 클리닝 지표(Cleaning Index)와 컬렉션 연산(CoLlection operation)의 약자이다. 클리닝 지표에 대한 자세한 설명은 4.2.1절에 있다.

4.3.1 클리닝 지표

클리닝 지표는 클리닝 세그먼트를 선택하는 기준값을 의미한다. 다시 말해서 클리닝 세그먼트는 가장 적은 클리닝 지표값[10]을 갖는 세그먼트가 된다. 본 논문에서는 클리닝 연산의 비용 절감과 사이클 평준화를 동시에 도모하기 위해서 식(7)과 같은 클리닝 지표를 정의한다.

$$\text{클리닝 지표} = (1-l)\times\mu_i + l\times\frac{\varepsilon_i}{\varepsilon_{\max}+1} \quad (7)$$

여기에서 μ_i와 ε_i는 세그먼트 i의 활용률과 초기화 횟수를 의미하며 $\varepsilon_{\max}$는 최대 초기화 횟수를 나타낸다. 정의된 클리닝 지표는 세그먼트의 활용률 관련 인자(μ_i)와 초기화 횟수 관련 인자($\frac{\varepsilon_i}{\varepsilon_{\max}+1}$)로 구분될 수 있는데, 이 두개의 인자는 정규 사이클 평준화도(normalized cycle leveling degree) l 에 의해 가중치가 주어진다. 이 l 은 3.2절에서 정의된 사이클 평준화도를 0과 1사이의 범위로 정규화시킨 값이다.

이 클리닝 지표는 사이클 평준화가 된 상태에서는 클리닝 비용을 줄이는 방향으로 작용하고, 사이클이 편중화된 경우에는 사이클 평준화를 위해 클리닝 세그먼트가 선택되도록 작용한다. 다시 말해서 사이클이 균등하게 분배되어 사이클 평준화도가 임계점 이하인 경우에는 l 값이 0에 가깝기 때문에 μ_i가 세그먼트를 선택하는 주요 인자가 되며, 이와 반대로 그 임계점을 넘어서 l 값이 1에 접근하면 사이클 평준화를 위해 초기화 횟수가 큰 세그먼트를 선택하게 한다. 이때 과도한 사이클 평준화를 방지하기 위해서 어느 정도 이상의 사이클 편중화가 진행된 후에야 l 값이 클리닝 지표값에 영향을 미치도록 하는 것이 필요하다. 그래서 l 값의 계산을 위해 다음과 같이 사이클 평준화도 Δ_ε에 대해 S자형으로 단조증가하는 함수를 사용한다.

$$l(\Delta_\varepsilon) = \begin{cases} \dfrac{2}{1+e^{\frac{k_\varepsilon}{\Delta_\varepsilon}}} & \text{단, } \Delta_\varepsilon \neq 0 \\ 0 & \text{단, } \Delta_\varepsilon = 0 \end{cases} \quad (8)$$

식 (8)에서 k_ε은 이 함수의 기울기를 결정하는 상수가 되며 그림 5에서 k_ε값이 커질수록 l 값이 완만하게 증가하는 것을 관찰할 수 있다. 그래서 이 k_ε값을 사이클 평준화도를 제어하는데 사용할 수 있다. 만일 엄격하지 않게 사이클을 평준화시키고자 한다면 k_ε값을 크게 하고 엄격하게 사이클 평준화를 하려고 한다면 k_ε값을 작게 하면 될 것이다. 예를 들어, 그림 5에서 k_ε값을 10으로 설정한 경우에는 사이클평준화도가 작은 값을 가져도, 즉 약간의 사이클 편중화가 일어나도 l 값이 급격히 증가하게 된다. 그리하여 클리너는 초기화 횟수 인자에 큰 비중을 가지는 클리닝 지표값을 가지게 되면서 사이클 평준화 작업을 수행하게 된다. 반대로 k_ε값을 500으로 설정하면 사이클평준화도가 100이 될 때까지는 l 값이 완만하게 증가하기 때문에 사이클 평준화 작업이 거의 일어나지 않을 것이다.

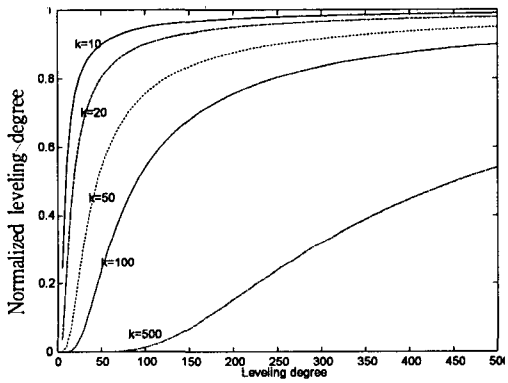


그림 5 k_ε의 변화에 따른 l의 변화

위와 같이 클리닝 지표를 통해 사이클 평준화를 이루는 기법을 앞서 소개한 CICL I에 결합시킨다. 이를 CICL II로 부르겠다.

4.3.2 세그먼트 할당 정책

플래시메모리 관리자의 할당자 모듈은 로그 공간 유지를 위해 어떤 새로운 세그먼트를 할당하느냐에 따라서 사이클 평준화를 더 개선할 여지를 가지고 있다. 이를 위해 할당자가 세그먼트를 할당하게 되는 경우를 두 가지로 분류해보면, (ㄱ) 일반적인 입출력이 수행되면서 계속적으로 로그 공간을 확보하고자 할 때와 (ㄴ) COLD데이타를 수집한 후에 이를 로그 말단에 복사할 때가 있다.

사이클 평준화를 위해서 할당자는 (ㄱ)의 경우에는 최소 초기화 횟수를 가지는 세그먼트를, (ㄴ)의 경우에는 최대 초기화 횟수를 가지는 세그먼트를 할당한다. (ㄱ)의 경우에 적용된 휴리스틱의 근거는 로그에 새로이 할당된 세그먼트는 그것이 갖는 데이타가 현재 빈번하게 접근되는 HOT데이타일 것이기 때문에 할당된 세그먼트는 클리닝될 수 있는 기회가 커질 것이라는 예측에

10) Sprite-LFS[10]의 cost-benefit 정책의 경우에는 제안된 클리닝 지표값($\frac{age\times(1-\mu)}{\mu}$)이 큰 것을 선택하게 된다.

있다. 그리고 (ㄴ)의 경우에는 컬렉션 연산에 의해 많은 양의 COLD데이타가 새로 할당된 세그먼트 안에 군집되어 저장될 것이기 때문에 당분간은 다른 세그먼트에 비해 상대적으로 무효화가 덜 진행될 것이다. 그래서 이 경우에는 가장 적게 사이클을 소비한 세그먼트를 할당해준다.

이러한 세그먼트 할당 기법을 가지는 할당자를 CICLⅡ에 결합시켜 CICLⅢ로 명명한다.

5. 성능 분석

5.1 시뮬레이션과 작업부하의 설계

성능 분석을 위해 우리는 저장 매체로서 64개의 세그먼트로 구성된 1기가 바이트 용량의 플래시메모리를 사용하는 플래시 저장시스템 시뮬레이터를 구현하였다. 각 세그먼트는 256개의 초기화 블록으로 구성된다. 플래시메모리 관리자의 구현에 있어서 클리너가 요구하는 유효 페이지의 수와 각 세그먼트의 초기화 횟수 등 세그먼트 관련 정보는 시뮬레이터 내에서 관리된다. 클리너는 이 세그먼트 정보를 이용하여 쓰기 연산이 100번 수행될 때마다 자유공간의 비율이 0.1이하인지를 검사하여 클리닝 연산을 시도한다. 그리고 컬렉터는 단편도가 0.8을 초과하는 COLD데이타에 대해 수행하며 사이클 평준화도를 제어하는 상수 k_ε는 엄격한 사이클 평준화를 위해 100으로 설정하였다. 다른 자세한 내용은 표 1에 기술되어 있다.

본 실험에서는 성능 실험의 신빙성을 높이기 위해 두 가지 유형의 작업부하(workload)를 사용하였다. 한 유형은 'UNIX'작업부하로서 [6]의 UNIX 파일 크기 분포에 따라서 생성되며 여기에 COLD데이타가 HOT데이타보다 그 크기가 크다는 성질[11]을 반영한다. 또 하나의 'LFS'유형의 작업부하는 Sprite-LFS[10]에서 사용된 'hot-and-cold' 작업 부하와 유사한 것으로서 일정한 작은 크기(4킬로바이트)의 파일에 로컬리티를 가지는 갱신(쓰기)연산을 가지고 있다.

본 논문이 다루는 문제는 읽기 연산의 성능과는 무관하기 때문에 설계된 작업부하는 읽기 요청을 포함하지 않는다. 그리고 각 작업부하는 여러 수준의 로컬리티에 따라 생성된다. (이때 로컬리티의 정도는 작업부하 이름과 함께 괄호 안에 표시된다.) 본 논문에서는 로컬리티를 Sprite-LFS[10]에서와 유사한 방식으로 표기한다. 예를 들어, '90→10'는 쓰기 연산의

90%가 전체 데이타의 10%에 집중되고, 그 나머지 10%의 쓰기 연산은 90%의 다른 데이타 영역에 접근함

표 1 시뮬레이션 패러미터

플래시메모리 관련 패러미터	
플래시메모리 타입	1Mbit/chip
플래시메모리 용량	1Gbytes
세그먼트 크기	16Mbytes
세그먼트의 수	64개
초기화 블록의 수	256개/세그먼트
페이지 크기	512bytes

클리너 패러미터	
클리닝 주기	100번 쓰기연산 마다
클리닝 시점	$\frac{\text{자유공간크기}}{\text{전체공간크기}} < 0.1$
클리닝 크기	1 세그먼트
컬렉터 패러미터	
컬렉션 데이타 인식	FAW크기 = 200 단편도 > 0.8
컬렉션 주기	k_p=50
컬렉션 크기	k_s=0.3×전체메모리크기
사이클 평준자 패러미터	
사이클 평준화도	k_ε=100

을 나타낸다. 그래서 '50→50'은 로컬리티가 없음을 의미한다. 실제 실험에서 UNIX(95→5) 작업부하를 사용한 경우, FAW의 크기를 200으로 하였을 때 *locality*는 약 0.7-0.8, UNIX(50→50)의 경우에는 0.1이하로 측정된다. 또한 갱신이 일어나지 않은 정적 데이타가 존재하는 작업부하를 표현하기 위해 또 하나의 패러미터를 추가할 수 있다. 예를 들어, '90→10, 80'은 쓰기 연산의 90%가 전체 데이타의 10%에 집중되고, 나머지 10%의 쓰기 연산이 80%의 다른 데이타 영역에 접근함을 의미한다. 이 로컬리티를 가지는 작업부하에는 접근되지 않는 10%의 데이타 영역이 존재하게 된다.

모든 작업부하가 가지는 쓰기 요청은 1,000,000페이지 분량이고 관련된 파일의 수는 500-2,000개 정도이다. 각 작업부하는 초기에 파일들을 랜덤하게 생성하고 이 생성된 파일들을 갱신 또는 삭제하는 연산 내용을 담고 있다. 쓰기 요청시마다 관련 파일은 의사 난수(pseudo-random number)에 따라 선택되며 선택된 파일 내에 갱신되는 지점도 같은 방식으로 선택된다.

5.2 실험 결과

본 연구에서는 사이클 평준화를 이룬 상태에서 클리닝 비용을 줄이는데 역점을 두기 때문에 제안한 기법의 성능평가를 위해 그리디 방법과 기존의 사이클 평준화 기법을 조합한 것을 기준으로 삼는다. 기존의 사이클 평준화 기법 중에서 로그에 새로운 세그먼트를 할당할 때 최소 초기화 횟수를 갖는 세그먼트를 선택하는 기법을 Greedy Ⅰ 방법에 결합시켜 GreedyⅡ로 명명하고, 이보다 강한 사이클 평준화를 위해 최소, 최대 사이클 횟수를 가지는 세그먼트들간에 데이타를 서로 교환하는 기법을 Greedy Ⅰ에 병합한 것을 GreedyⅢ로 명명한다.

표 2 플래시메모리 관리 방법의 분류

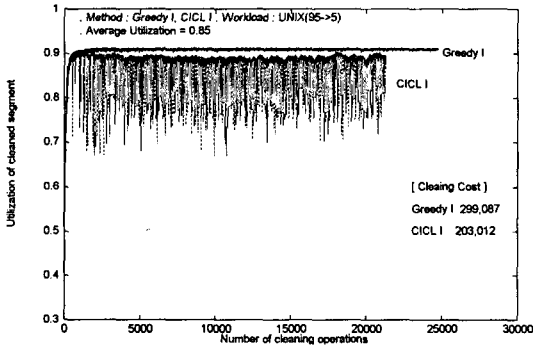
관리 방법	클리닝 지표	COLD데이타 분리 방법	사이클 평준화 방법
Greedy I	μ	없음	없음
Greedy II			최대 초기화 세그먼트 할당
Greedy III			데이타 교환
Cost-Benefit	$\frac{age\times(1-\mu)}{\mu}$	나이정렬기법	없음
CICL I	μ	컬렉션 연산 (4.2절 참조)	최대 초기화 세그먼트 할당
CICL II			
CICL III	$\delta\times\mu+(1-\delta)\times\frac{\epsilon}{\epsilon_{max}+1}$		최대/최소 초기화 세그먼트 할당 (4.3.2절 참조)

또한 순수하게 클리닝 비용 측면에서 다른 로깅기법과 비교하기 위해서 로그기반 파일시스템인 Sprite-LFS의 cost-benefit기법을 평가할 것이다. 표 2에는 본 논문에서 제안한 플래시메모리 공간 관리 방법들과 이와 비교 평가하기 위해 제시된 여러 가지 기법들이 정리되어 있다.

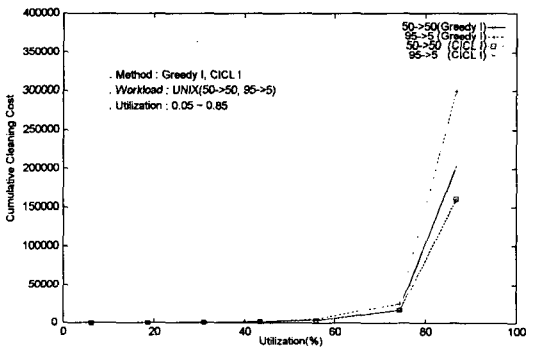
5.2.1 클리닝 비용 측면

그림 6(a)는 클리닝이 계속 진행될 때마다 클리닝 세그먼트의 활용률 값의 변화를 보여준다. 이 그림에서 Greedy I 은 전술한 바와 같이 COLD데이타의 단편화로 인해 클리닝 세그먼트의 활용률 값이 수렴하는 현상을 볼 수 있다. 그리고 CICL I 의 클리닝 효율이 Greedy I 과 비교해서 클리닝 세그먼트의 활용률과 클리닝 횟수가 감소하였는데 평균적으로 누적 클리닝 비용이 약 30% 감소되었다. 그리고 제안한 CICL 방법에서 클리닝되는 세그먼트 활용률 값이 진동하는 것은 컬렉션 연산에 의해 COLD데이타가 군집화 되었다가 다시 단편화되는 상황을 나타낸다. 그림 6(b)는 누적 클리닝 비용의 측면에서 CICL I 과 Greedy I 의 클리닝의 성능을 비교한 것이다. 이 그림에서 세그먼트 활용률과 로컬리티가 높아짐에 따라 컬렉션 연산의 효과가 높아짐을 알 수 있다.

그림 7은 각 작업부하에 대해 여러 가지 플래시메모리 관리 기법을 수행하였을 때 소요되는 누적클리닝 비용을 보여준다. 대부분의 작업 부하에서 제안된 방법(CICL I,II,III)이 그리디 방법보다 우세한 성능을 보였다. 주목할 것은 로컬리티가 존재하고 세그먼트 활용률이 높을수록 그리디 방법(Greedy I,II,III)과 Sprite-LFS의 cost-benefit방법(Cost-Benefit)보다 누적 클리닝 비용이 적다는 것이다. 이는 컬렉터가 상대적으로 적은 비용을 지출하면서 클리닝 연산 비용을 줄이는데 크게 기여하였기 때문이다. 예를 들어, CICLIII는 UNIX (95→5) 작업부하에서는 전체 클리닝 비용 중에서 약 8%정도(그림 7에서 컬렉션 비용은 막대 상단에 표시되어 있음)의 컬렉션 비용을 부담하고 Greedy I,II의 경우보다 약 35%정도의 비용을 감소시켰다.



(a)



(b)

그림 6 (a) **Greedy I** 과 **CICL I** 의 비교(클리닝 세그먼트의 활용률)
(b) **Greedy I** 과 **CICL I** 의 비교(누적 클리닝 비용)

CICL II의 경우에는 클리닝 지표에 의한 사이클 평준화로 인해 컬렉션에 의한 클리닝 성능 향상 효과가 조금 약화되었다. 그러나 4.2.3절에서 제안한 세그먼트 할당 휴리스틱을 가지는 CICLIII는 CICL II에서 상실했던 비용 감소 효과를 회복하게 된다. 사이클 평준화를 위해 마련된 휴리스틱이 클리닝 비용을 낮추는 부차적 효과를 얻은 것이다. UNIX(95→5) 작업부하에서는 오히려

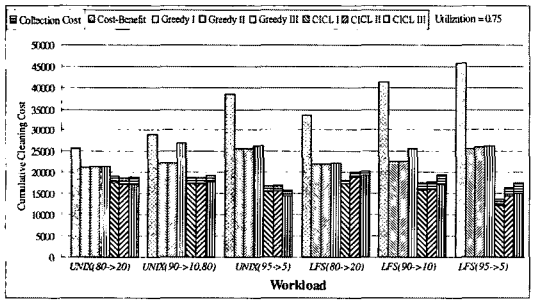
CICLⅢ가 CICL Ⅰ,Ⅱ보다 사이클 평준화 측면에서 우세하면서(그림 7(a) 참조) 동시에 클리닝 비용 측면에서도 CICL Ⅰ,Ⅱ를 능가한다.

UNIX(90→10,80) 작업부하에서는 제안된 방법의 클리닝 비용의 절감 효과가 다른 작업부하에서보다 적게 나타나고 있다. 이는 UNIX(90→10,80)작업부하 10%의 비율로 접근되지 않는 정적 데이타를 보유하고 있기 때문에 다른 작업부하에 비해 단편화되는 데이타가 덜 발생하기 때문이다. LFS작업부하에서도 일정한 작은 크기의 파일들만을 포함하고 있기 때문에 불필요하게 복사되는 COLD데이타의 양이 UNIX작업부하의 경우보다 많아 제안된 방법의 효과가 크지 못한 것이다. 결과적으로 제안된 CICL 방법은 클리닝 연산시 필요 없이 로그 공간을 이동하는 COLD데이타가 많을수록 그 효과가 커짐을 확인할 수 있다.

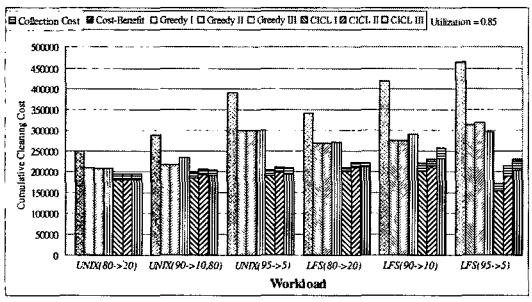
Cost-Benefit 방법은 [8,10]에서 사용된 cost-benefit정책과 age-sorting기법을 포함하고 있는 것이다. 이 기법은 디스크 기반의 로깅 시스템에서는 좋은 성능을 보이는 것으로 알려져 있지만 본 실험에서는 모든 경우에 다른 기법에 비해 클리닝 성능이 가장 좋지 않게 나타났다. 이것은 플래시 저장시스템에서 사용하는 세그먼트의 크기가 Sprite-LFS에서 설정한 세그먼트 크기보다 매우 크기 때문이다. cost-benefit정책은 세그먼트 활용률이 작으면서 동시에 age가 큰 세그먼트를 클리닝함으로써 성능을 높인다. 하지만 age에 따라 페이지들을 정렬한다 할지라도 매우 큰 세그먼트가 가지는 페이지들의 age값들이 일관되지 않아 본래 가지는 cost-benefit정책의 효과가 없어진다.

5.2.2 클리닝 횟수 측면

그림 8은 제안한 기법이 클리닝 횟수를 얼마나 감소시키는지를 알아보기 위해서 각 클리닝 연산 사이에서 발생한 쓰기 요청 회수를 측정한 것이다. 이 값은 총 쓰기 요청 수를 클리닝 총 횟수로 나눈 것으로서 이를 '클리닝간 쓰기허용량'이라고 부르겠다. 이 클리닝간 쓰기

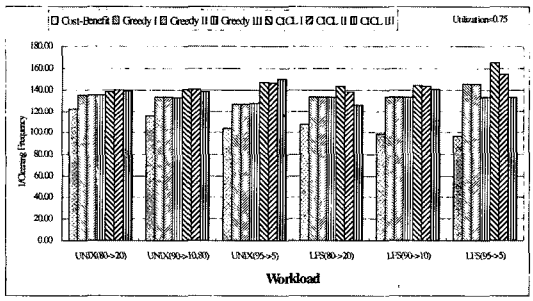


(a)

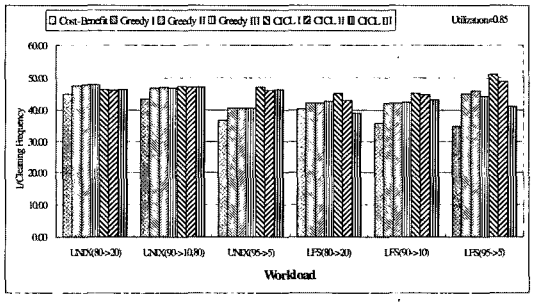


(b)

그림 7 (a) 누적 클리닝비용의 비교 (평균 활용률 = 0.75)
(b) 누적 클리닝비용의 비교 (평균 활용률 = 0.85)



(a)



(b)

그림 8 (a) 클리닝간 쓰기 용량의 비교 (평균 활용률 = 0.75)
(b) 클리닝간 쓰기 용량의 비교 (평균 활용률 = 0.85)

허용량은 정의에 따라 누적클리닝 비용과 깊은 관련을 맺는 측정치가 되며 시스템의 수명을 판단하는 지표가 될 수 있다. 이 값이 상대적으로 크다는 것은 일정한 쓰기 요청에 대해서 클리닝이 지연되어 초기화 연산이 덜 발생하였음을 의미한다. 다시 말해서 세그먼트 사이클을 적게 소비하는 것이므로 그만큼 사용 플래시 용량을 절약했다는 것을 나타낸다. 그림 8에서 대부분의 경우에 제안된 CICL 방법들이 다른 방법에 비해 크게 나타나고 있으며 로컬리티가 클수록 다른 기법들이 가지는 값과의 차가 커지고 있다.

그런데 이 클리닝간 쓰기허용량이 적다고 해서 반드시 누적 클리닝 비용이 커지는 것은 아니다. 예를 들어, 컬렉션 연산의 기능이 효과적으로 작용하여 클리닝 연산의 비용이 크게 적어지는 경우가 그렇다. 그림 7(b), 8(b)에서 평균활용률이 0.85이고 UNIX(80→20)작업부하에 대해 측정한 것을 보면 CICL방법이 다른 방법에 비해 클리닝 연산사이에서 많은 쓰기 요청이 발생했지만, 클리닝 비용은 증가하지 않았음을 관찰할 수 있다.

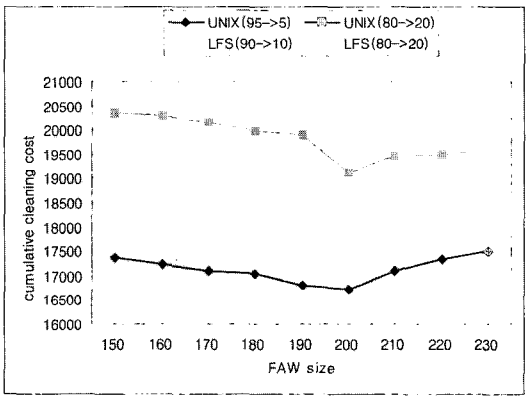


그림 9 FAW크기의 변화에 따른 누적 클리닝 비용

5.2.3 컬렉션 연산의 효과

컬렉션 연산의 성능을 결정짓는 인자는 FAW의 크기, 컬렉션 크기, 컬렉션 주기 등이다. 전술한 바와 같이 FAW는 본 논문에서 COLD데이타를 판단하는 도구로 사용되는 것인데 그 크기가 너무 작거나 너무 크면 그 기능을 상실하게 된다. 그림 9는 제안하는 기법을 사용하여 여러 작업부하에 대한 누적 클리닝 비용을 측정한 것이다. FAW의 크기가 200 전후일 때 클리닝 비용이 가장 적게 나타나고 있다. 그래서 본 실험에서는 FAW의 크기를 200으로 설정한다(표 1 참조).

또한 컬렉션 연산은 컬렉션 크기 또는 컬렉션 주기의 변화에 따라 민감하게 동작한다. 그런데 본 논문은 컬렉션 주기와 컬렉션 크기의 정밀한 분석보다는 컬렉션 연산 자체의 개략적인 효과에 더 큰 의의를 두고 있기에 컬렉션 크기에 대해서만 다루고자 한다. 컬렉션 주기마다 항상 컬렉션 연산이 발생하는 것이 아니기 때문에 정밀한 분석을 위해서는 각 컬렉션 주기값마다 컬렉션 크기값의 모든 조합에 대한 상당한 실험이 요구된다.

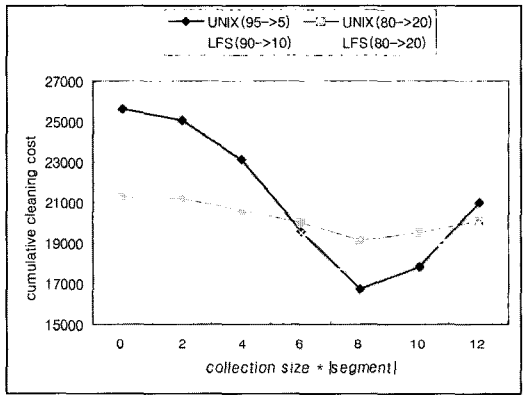


그림 10 컬렉션 크기의 변화에 따른 누적 클리닝 비용

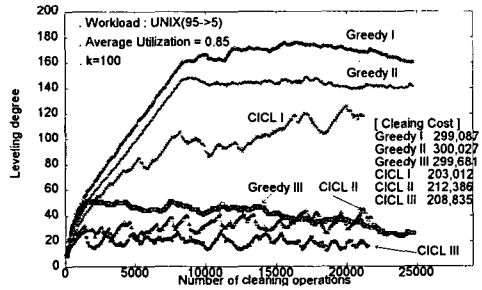
그림 10은 CICLI기법을 사용했을 때 컬렉션 주기값을 고정시켜놓고 컬렉션 크기의 변화에 따른 누적 클리닝 비용값을 측정한 것이다[11]. 이때 컬렉션 크기에 해당하는 세로축은 세그먼트의 크기, 즉 페이지 개수의 배수로 나타낸다. 이 그림을 통해서 컬렉션 연산이 클리닝 비용을 감축하는 효과를 가지기 위해서는 컬렉션 크기가 어느 정도 이상이 되어야 하며, 또한 지나치게 많은 데이타를 수집하는 경우에는 오히려 전체 클리닝 비용이 높아질 수 있음을 확인하였다. 컬렉션 주기와 관련해서도 그림 10과 유사한 그래프를 얻을 수 있음을 충분히 짐작할 수 있으며 자세한 실험 설명은 생략한다. 본 실험에서는 그림 10에 근거하여 컬렉션 크기를 결정하는 비례상수 k_s 값을 전체메모리 크기의 0.3배수로, 컬렉션 주기를 결정하는 비례상수 k_p값은 50으로 설정하였다(표 1 참조).

5.2.4 사이클 평준화 측면

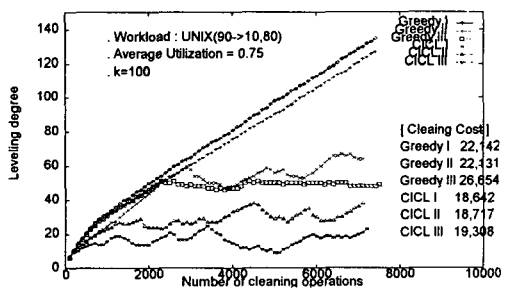
그림 11은 클리닝 연산이 진행되고 있을 때마다 사이클 평준화도 Δ_ϵ를 측정한 것이다. 그래프의 모양이 작은 값을 가지면서 진동하는 것은 사이클 평준화가 되어 가

11) 그림 10은 여러 컬렉션 주기 값 중에서 가장 성능이 좋은 때를 취한 것이다.

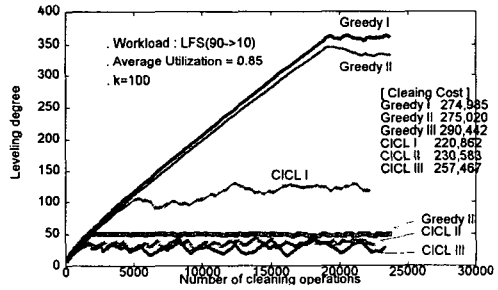
는 과정을 보여준다. UNIX(90→10,80) 작업부하의 경우에 Greedy Ⅰ,Ⅱ기법은 극심한 사이클 편중화 현상을 보인다. 이에 반해서 클리닝 지표를 사용하는 CICLⅡ기법과 세그먼트 할당 휴리스틱을 사용하는 CICLⅢ은 누적 클리닝 비용이 적음에도 불구하고 안정된 평준화 현상을 보이는 것을 관찰할 수 있다. 특히 CICLⅢ기법은 다른 방법에 비해 사이클 평준화 정도가 우수하면서도 클리닝 비용 측면에서도 뒤떨어지지 않는다.



(a)



(b)



(c)

그림 11 (a) 사이클 평준화도의 변화 (UNIX(95→5) 작업부하) (b) 사이클 평준화도의 변화 (UNIX(90→10,80) 작업부하) (c) 사이클 평준화도의 변화 (LFS(90→10) 작업부하)

그리디 클리닝 방법에 기존의 사이클 평준화기법을가미한 GreedyⅢ방법은 CICL방법에 근접하는 사이클 평준화 효과를 보이고 있지만 클리닝 비용 면에서는 Cost-Benefit을 제외해서 가장 낮은 성능을 보이고 있다. 이것은 초기화 횟수의 최대, 최소값을 갖는 세그먼트 사이에서 데이타 교환으로 많은 쓰기 연산이 발생했기 때문이다.

그리고 본 실험에서 클리닝 비용을 줄이기 위해 제안된 컬렉션 연산이 사이클 평준화에도 기여함을 알 수 있다. 그림 11에서 컬렉션 연산 기능만을 가지는 CICL Ⅰ기법은 사이클 평준화를 위한 기능이 포함되어 있지 않음에도 불구하고 모든 경우에 사이클평준화 효과를 보이고 있다. 이것은 컬렉션 연산이 COLD데이타를 포함하는 세그먼트내의 페이지들을 무효화시킴으로써 이 세그먼트들이 클리닝 될 가능성을 높이는 것으로 설명할 수 있다.

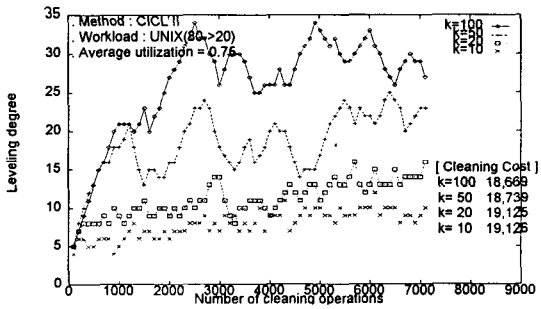
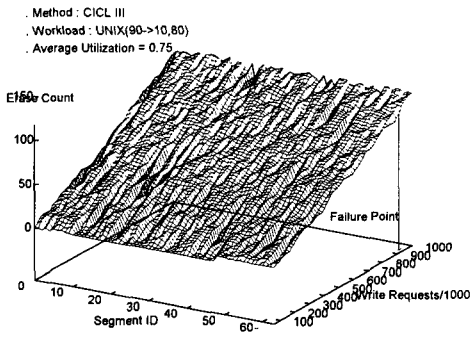


그림 12 k_ϵ에 의한 사이클평준화도의 제어

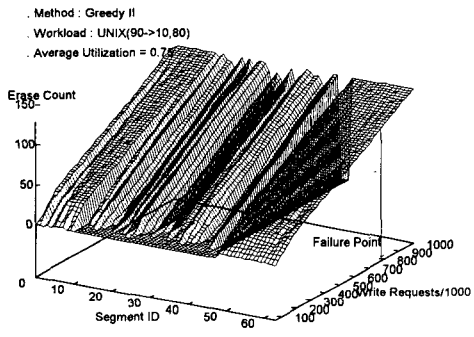
또한 앞에서 제안된 클리닝 지표의 상수 k_ϵ는 사이클 평준화도를 제어할 수 있다고 언급한 바가 있는데 그림 12에서 이 사실을 확인 할 수 있다. 이 그림은 CICLⅡ기법에서 상수 k_ϵ값이 적어짐에 따라 사이클 평준화가 보다 엄격하게 이루어지고 있음을 보여준다. 그리고 사이클 평준화 효과가 강해지면서도 누적 클리닝 비용이 크게 증가하고 있지는 않는다. 실제로 이 k_ϵ값은 플래시 저장시스템이 사용되는 환경에 맞게 적절히 조정하면 될 것이다. 만약 플래시메모리의 사이클 제한 횟수의 오차 범위를 알 수 있다면 이 값도 k_ϵ에 반영할 수 있다.

그림 13은 시간(쓰기요청)의 진행에 따라서 모든 세그먼트들의 초기화 횟수값이 변하는 양상을 보여준다. 그래서 사이클 편중화가 발생하는 경우에는 골짜기 형태의 그래프가 형성될 것이다. 이 그림에서 GreedyⅡ의

경우에는 시간이 지나면서 깊은 골짜기가 생기고 있지만, CICLⅢ의 경우에는 대체로 편평한 면이 만들어짐을 관찰할 수 있다. 이를 통해서 제안된 사이클 평준화 기법이 시스템의 저장 공간을 잘 활용하면서 시스템 수명을 높이는 기여함을 확인할 수 있다.



(a)



(b)

그림 13 (a) 초기화 횟수의 변화(Greedy Ⅱ방법)
(b) 초기화 횟수의 변화(CICLⅢ 방법)

이 두 예를 가지고 사이클 평준화 기법에 따라 시스템의 수명이 어떤 차이를 보이는지 알아보자. 만일 사이클링 횟수가 100번으로 제한되고, 가용 플래시메모리 공간의 비율이 0.2이하일 때 시스템 고장이 발생한다고 가정하자. 그리고 그림 13에 표시된 고장시점(failure point)까지 GreedyⅡ의 사용 사이클 용량은 약 5,700 (segments · erases)이고 CICLⅢ은 약 6,200 (segments · erases) 사이클 용량을 소비한다. 게다가 CICLⅢ기법은 GreedyⅡ보다 일정량의 쓰기 요청에 대해 클리닝 횟수가 약 0.1의 비율로 감소하였다. 결과적으로 CICLⅢ은 시스템의 수명을 GreedyⅡ보다 약 20% $\left(=\frac{6,200}{5,700\times(1-0.1)}-1.0\right)$ 증가시킬 수 있다.(3.2절 참조)

지금까지의 여러 가지 실험을 통해서 제안된 사이클 평준화 기법이 클리닝 메커니즘과 매우 조화롭게 동작하고 있음을 확인할 수 있었다. 기존의 사이클 평준화 방법이 클리닝 연산과 일반적인 입출력 연산의 성능을 저하시키는 성질을 가지고 있는 반면에 제안된 사이클 평준화 기법은 클리너의 클리닝 지표를 통해 간단하게 구현되어 시스템에 큰 부하를 주지 않으면서 효과적으로 수행되었다. 이 방법이 클리닝 연산 비용을 높이지 않으면서 안정된 효과를 가질 수 있는 것은 플래시 공간 관리 전략으로 사용된 로깅과 클리닝 비용을 줄이기 위한 컬렉션 연산, 세그먼트 할당을 위해 사용한 휴리스틱 등이 잠재적으로 사이클 평준화의 성질을 이미 가지고 있기 때문이다.

6. 결 론

본 논문은 플래시메모리 공간 관리에서 반드시 해결해야 할 사이클 평준화와 클리닝 비용 절감 문제를 COLD데이타의 수집과 특별히 고안된 클리닝 지표를 통해 해결하였다. 더불어 제안된 방법을 평가하기 위해 플래시메모리에 적합한 새로운 클리닝 비용 모델을 제안하였다. 플래시메모리를 저장 매체로 하였을 때의 중요한 문제는 플래시메모리 사용을 균등하게 분포시키는 것인데 이는 클리닝 지표와 세그먼트 할당과 관련된 휴리스틱, 컬렉션 연산에 잠재된 평준화 효과 등에 의해 해결된다. 결과적으로 제안된 플래시메모리 관리자가 클리닝 비용 절감 문제와 사이클 평준화 문제를 동시에 해결함으로써 신뢰성 있는 시스템을 구축하는데 큰 도움을 줄 수 있다. 그리고 실험 결과, 제안된 COLD데이타의 컬렉션 기법은 특히 플래시메모리 공간 활용률이 높거나 로컬리티 정도가 높을 때 좋은 성능을 보이고 있기 때문에 쓰기연산이 로컬리티를 가지고 빈번하게 발생하는 OLTP환경이나 사무/엔지니어링(office/engineering)환경에 적합할 것으로 본다. 또한 제안된 방법은 로깅을 전제로 하고 있지만, 실제로는 로깅에 크게 의존하지는 않아서 다른 저장 전략을 가지는 시스템에 쉽게 활용할 수 있다.

참 고 문 헌

[1] T. Blackwell, J. Harris, and M. Seltzer, "Heuristic Cleaning Algorithms in Log-Structured File

Systems," *Proceedings of '95 Winter, USENIX,* pp.277-287, 1995
[2] R. Cáceres, F. Douglis, K. Li, and B. Marsh, "Operating System Implications of solid-state mobile computers," Technical Report MITL-TR-56-93, Matsushita Information Technology Laboratory, 1993
[3] P. J. Denning, "Working Sets Past and Present," *IEEE Transactions on Software Engineering,* Vol. SE-6, No.1, pp. 64-84, 1980
[4] B. Dipert, and M. Levy, "Designing with FLASH MEMORY," Annabooks, pp. 227-271, 1994
[5] F. Douglis, R. Cáceres, F. Kaashoek, K. Li, B. Marsh, and J. A. Tauber, "Storage Alternatives for Mobile Computers," *Proceedings of the 1st Symposium on Operating Systems Design and Implementation,* pp.25-37, 1994
[6] G. Irlam, "Unix File Size Survey," *http://www.base.com/gordoni/gordoni.html,* 1993
[7] A. Kawaguchi, S. Nishioka, and H. Motoda, "Flash memory Based File System," *Proceedings of '95 Winter, USENIX,* pp.155-164, 1995
[8] J. N. Matthews, D. Roselli, A. M. Costello, R. Y. Wang and T. E. Anderson, "Improving the Performance of Log-Structured File Systems with Adaptive Methods," *Proceedings of the 16th ACM Symposium on Operating Systems Principles,* pp.238-251, 1997
[9] J. T. Robinson, "Analysis of Steady-State Segment Storage Utilization in a Log-Structured File System with Least-Utilized Segment Cleaning," *Operating System Review,* Vol.30, No.4, pp.29-32, 1996
[10] 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
[11] C. Ruemmler, and J. Wilkes, "UNIX disk access patterns," *Proceedings of. '93 Winter USENIX,* pp.405-420, 1993
[12] D. See, and C. Thurlo, "Managing Data in an Embedded System Utilizing Flash Memory," Intel Technical Note,1995, http://www.intel.com/design/flcomp/papers/esc_flsh.htm
[13] M. Wu, and W. Zwaenepoel, "eNVy : A Non-Volatile Main Memory Storage System," *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems,* pp.86-97, 1994
[14] Intel, Flash Memory Databook, pp.21-29, 1995

김 한 준
1994년 서울대학교 계산통계학과 졸업(학사). 1996년 서울대학교 계산통계학과 졸업(석사). 1996년 3월 ~ 현재 서울대학교 컴퓨터공학부 박사과정. 관심분야는 파일시스템, 데이타베이스, 데이타마이닝 임

이 상 구
1985년 서울대학교 자연과학대학 계산통계학과 졸업(학사). 1987년 Northwestern Univeristy 졸업(석사). 1990년 northwestern University 졸업(박사). 1989년 9월 ~ 1990년 6월 U. of Minnesota 전임강사. 1990년 ~ 1992년 7월 EDS 연구원. 1992년 8월 ~ 현재 서울대학교 전산과학과 부교수. 관심분야는 논리 데이타베이스, 질의 최적화, 정보검색