LFS의 쓰기 성능 최적화를 위한 세그먼트 공간 재활용 기법

(A Segment Space Recycling Scheme for Optimizing Write Performance of LFS)

오 용 석 † 김 은 삼 ††
(Yongseok Oh) (Eunsam Kim)
최 종 무 ††† 이 동 희 ††††
(Jongmoo Choi) (Donghee Lee)
노 삼 혁 †††††
(Sam H. Noh)

요 약 LFS(Log-structured File System)는 쓰기 요청을 세그먼트 버퍼에 모으고, 세그먼트 단위로 순차 기록함으로써 무작위 쓰기에서도 최적의 성능을 보여준다. 그러나 디스크의 공간이 유한하여, LFS는 여유 세그먼트를 생성하는 클리닝을 수행해야 한다. 파일 시스템의 사용률이 증가함에 따라 세그먼트 클리닝 비용이 급격히 증가하는 단점이 있다. 본 논문에서는 LFS의 쓰기 성능 최적화를 위한 세그먼트 공간 재활용 기법을 설명한다. 이 기법은 유효 세그먼트를 재활용하여 여유 공간을 생성하는 방법으로 빈 세그먼트가 없이 쓰기요청을 처리 할 수 있다. 따라서 높은 비용의 클리닝 동작 없이, 데이터를 세그먼트 내 여유공간에 동적 재배치하여 쓰기요청을 처리한다. 또한 효율적인 세그먼트 공간 재활용을 위해 데이터 및 세그먼트의 지역성을 고려하는 분류기법을 설명한다. 실험 결과에서 이 기법은 파일 시스템의 사용률이 90%인 경우에도 기존 WOLF 기법을 사용한 LFS 보다 HDD에서 1.9배, SSD에서 1.6배의 성능향상을 보여준다

키워드 : 로그 구조 파일 시스템, 세그먼트 클리닝, 가비지 콜렉션

Abstract The Log-structured File System (LFS) collects all modified data into a memory buffer and writes them sequentially to a segment on disk. Therefore, it has the potential to utilize the maximum bandwidth of storage devices where sequential writes are much faster than random writes. However, as disk space is finite, LFS has to conduct cleaning to produce free segments. This cleaning operation is the main reason LFS performance deteriorates when file system utilization is high. To overcome painful cleaning and reduced performance of LFS, we propose the segment space recycling (SSR) scheme that directly writes modified data to invalid areas of the segments and describe the classification method of data and segment to consider locality of reference for optimizing SSR scheme. We implement U-LFS, which employs our segment space recycling scheme in LFS, and experimental results show that SSR scheme increases performance of WOLF by up to 1.9 times in HDD and 1.6 times in SSD when file system utilization is high.

Key words : Log-structured File System, Segment Cleaning, Garbage Collection

1. 서 론

LFS(Log-structured File System)[1]는 쓰기 요청을 세그먼트 버퍼에 모아 세그먼트 단위로 디스크에 순차적으로 기록한다. 따라서 무작위 쓰기 요청에 대해서도 순차 쓰기만큼의 성능을 보인다. 그러나 세그먼트 공간을 다 소비하면 세그먼트 클리닝을 해야 한다. 그리고 파일 시스템의 사용률이 증가할 수록 클리닝 비용이 높아져 전체 쓰기 성능의 저하로 이어진다. 이런 이유로 LFS의 성능향상을 위한 연구들은 주로 클리닝 비용을 최소화하려고 노력하였다[1-6].

본 논문에서는 그 동안의 연구와 다른 방향으로 클리닝 수행 없이, 소규모 데이터를 동적 재배치하는 세그먼트 공간 재활용 기법(Segment Space Recycling Scheme)을 설명한다. 파일 시스템이 노화(Aging)되면 이미 사용된 세그먼트에는 무효 블록과 유효 블록이 존재 한다. 무효 블록은 데이터가 다른 세그먼트로 재배치된 상태이고,

· 이 논문은 2008년도 서울시립대학교 연구년교수 지원에 의하여 연구되었음
· 이 논문은 2009 한국컴퓨터종합학술대회에서 '세그먼트 공간 재활용 기법을 이용한 LFS의 쓰기 성능 최적화 연구'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울시립대학교 컴퓨터통계학과
ysoh@uos.ac.kr
†† 정 회 원 : 홍익대학교 정보컴퓨터공학부 교수
eskim@hongik.ac.kr
††† 종신회원 : 단국대학교 컴퓨터학과 교수
choijm@dankook.ac.kr
†††† 정 회 원 : 서울시립대학교 컴퓨터통계학과 교수
dhl_express@uos.ac.kr
(Corresponding author임)
††††† 종신회원 : 홍익대학교 정보컴퓨터공학부 교수
samhnoh@hongik.ac.kr

논문접수 : 2009년 8월 14일
심사완료 : 2009년 10월 5일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제12호(2009.12)

유효 블록은 유효한 데이터가 존재하는 블록이다. 이 때, 세그먼트 공간 재활용 기법은 무효 블록을 여유 블록으로 재활용하여 데이터를 동적 재배치하는 방법이다.

이 기법의 사용으로 명시적인 클리닝 수행없이 세그먼트 내 여유 공간으로 활용할 수 있으며, 세그먼트의 지역성과 사용률을 고려하면 세그먼트 공간 재활용의 효율과 쓰기 성능을 향상시킬 수 있다. 세그먼트 공간 재활용기법을 최적화하기 위해 두 가지 추가 기능이 필요하다.

첫째, 소규모 데이터의 동적 재배치(Fine-grained Data Relocation)가 가능해야 한다. LFS의 경우 일반적으로 전체 세그먼트에 새로운 데이터를 기록하며, 경우에 따라 부분 세그먼트(Partial segment)단위로 데이터를 기록할 수 있다. 그러나 세그먼트 내 공간 재활용이 가능 하려면, 이보다 더 작은 단위로 데이터를 필요한 위치에 재배치하는 기능이 필요하며, 이는 기존 LFS의 구조에서는 불가능하다. 따라서 본 논문에서는 소규모 데이터 재배치가 가능한 새로운 LFS를 구현하였다.

둘째, 데이터의 즉석 분류 기법(On-the-fly Data Classification)이 필요하며, 이 기법으로 세그먼트 공간 재활용 기법의 효율을 향상 시킬 수 있다. 파일 데이터 블록의 종류는 Hot, Cold 특성으로 분류할 수 있다. Hot 데이터 블록은 자주 참조 되고, Cold 블록은 덜 참조되는 특성을 가진다[5]. 이와 같은 특성에 따라 데이터를 물리적인 그룹화 하는 것이 중요하다. 이를 이용하여 소규모 데이터 재배치 시 데이터를 특성에 따라 모으면 공간 재활용 및 쓰기 비용을 최적화 시킬 수 있다.

본 논문은 다음과 같이 구성되어있다. 2절에서는 배경지식 및 관련연구들에 대해서 살펴보고, 3절에서는 세그먼트 공간 재활용이 가능한 LFS의 설계 및 구현에 대하서 설명한다. 4절에서는 세그먼트 공간 재활용 기법의 최적화를 설명하고 5절에서는 성능평가에 대해서 논의한다. 마지막으로, 6절에서 결론을 맺는다.

2. 관련 연구

FFS(Fast File System)[7]는 메타데이터와 데이터를 되도록 동일한 실린더 그룹에 배치하여 탐색시간을 최소화 하였지만, 쓰기 요청이 여러 실린더 그룹에 분산되면 디스크 헤드의 움직임을 유발하여 성능 저하로 이어진다.

LFS[1]는 모든 쓰기요청을 세그먼트 단위로 모아 디스크에 기록을 하여서, 무작위 쓰기의 경우에도 성능이 좋지만 빈 세그먼트를 생성하는 클리닝 작업이 필수적이다. 이러한 클리닝 작업이 LFS의 성능을 결정하기 때문에 효율적인 클리닝 작업을 위한 다양한 연구가 수행되었다[1-6]. 세그먼트 클리닝 기법에는 크게 LFS Greedy [1], Cost-Benefit[1]과 Hole Plugging[4]기법이 있다.

Greedy기법은 유효 데이터가 가장 적은 세그먼트를 선택하여 유효 데이터를 한 세그먼트에 집합시키는 작업을 반복적으로 수행함으로써 빈 세그먼트를 생성하지만 지역성을 고려하지 않은 문제점이 있다. Cost-Benefit 기법은 Hot-Cold 참조 지역성을 고려하여 사용률이 낮고 오래 동안 참조되지 않은 Cold 데이터를 그룹화 함으로 Greedy 기법보다 클리닝 비용을 낮추었지만 Greedy 기법과 마찬가지로 물리적인 데이터 이동을 피할 수 없었다. 그리고 세그먼트 사용률이 낮은 경우 클리닝의 오버헤드가 작지만 사용률이 증가할수록 클리닝 비용은 지수적으로 높아지는 단점이 있다.

이런 문제점을 해결하기 위해 Hole Plugging은 세그먼트 클리닝을 수행할 때 부분적으로 빈 세그먼트의 유효한 블록을 다른 세그먼트의 Hole(무효 블록 공간)에 채움으로써 빈 세그먼트를 생성하는 것이다. 하지만 유효 블록을 재배치 할 때, 대상 세그먼트의 지역성을 고려하지 않은 문제점이 있다.

Greedy와 Cost-Benefit기법은 파일시스템 사용률이 낮을 때 효과적이며, Hole Plugging은 파일시스템 사용률이 높을 때 효과적이므로, [4]의 연구에서는 파일시스템 사용률에 따라 이들을 선택적으로 적용하여 클리닝 비용을 줄이려고 노력하였다.

Hole Plugging 기법은 클리닝을 위해서 세그먼트의 무효화된 영역을 이용한다는 점에서 세그먼트 공간 재활용 기법과 유사한 면이 있다. 즉, Hole Plugging 기법은 세그먼트 클리닝을 수행할 때 부분적으로 비어있는 세그먼트의 빈 공간에 유효한 블록을 복사한다. 결과적으로 Hole Plugging은 세그먼트 클리닝의 한 가지 방법으로 빈 세그먼트를 생성한다. 그러나 본 논문에서 제안하는 세그먼트 공간 재활용 기법은 새로운 데이터를 부분적으로 비어있는 세그먼트의 빈 공간에 곧바로 기록하며, 빈 세그먼트를 생성하는 세그먼트 클리닝 동작이 필요 없다.

WOLF(Write data re-Organization for Log-structured Filesystems)[5]에서 자주 참조되는 데이터를 Hot으로 그렇지 않은 것을 Cold로 분류하고 메모리 상에서 그룹화 하고 별도의 세그먼트에 기록하여 장기적으로 클리닝 비용을 줄이려고 노력하였다.

Hylog(Hybrid Log-structured)[6]에서는 클리닝 비용을 줄이기 위해서 Hot 데이터를 LFS 쓰기 방법으로 다루고 자주 참조 되지 않는 Cold 데이터는 덮어쓰기 방법으로 관리를 하였다. 그러나 현재까지 우리의 지식에 따르면, 이 연구는 시뮬레이션을 바탕으로 하였으며, 실제 시스템에 구현하여 실험한 결과는 제시되지 않았다. 본 논문에서는 Cost-Benefit, WOLF기법을 실제 구현하여 실험을 하였으며, 세그먼트 공간 재활용 기법을 이용한 LFS와 비교하여 성능 향상을 보일 것이다.

3. 세그먼트 공간 재활용을 위한 LFS의 설계 및 구현

본 절에서는 세그먼트 공간 재활용기법을 위한 LFS의 설계와 구현에 대해서 설명한다. 크게 두 가지 기법이 필요한데 세그먼트 내에 무효 블록을 여유 블록으로 생성하는 세그먼트 공간 재활용 기법과 세그먼트 버퍼의 데이터들을 소규모 재배치하는 기법이 필요하다.

본 논문에서 제안하는 세그먼트 공간 재활용 기법은 기존 세그먼트(유효 데이터가 존재하는)에 클리닝 수행 없이 곧바로 데이터의 쓰기를 위한 여유 공간을 생성하는 방법이다. 기존의 LFS는 쓰기를 위해서 세그먼트 크기의 공간이 필요하며, 세그먼트가 부족할 경우에 클리닝을 수행해야 하는 차이점이 있다.

세그먼트 공간 재활용 과정을 구체적으로 살펴보면 다음과 같다. 먼저 공간 재활용을 위해 Segment Usage Table[1]의 정보(유효 블록 수, 무효 블록 수, 생성시간, 변경시간)를 가지고 정책에 따라 대상 세그먼트를 선택한다(정책은 4절에 설명함). 그리고 나서 Segment Summary에 있는 finfo 자료구조(I-node 번호, 버전 번호, 블록 수, 논리 블록 번호)[1]와 현재 i-node가 가리키는 물리블록 번호와 현재 물리 블록 번호를 비교하여 무효 또는 유효 블록인지를 판별한다. 무효한 경우, 재활용하여 여유 공간으로 사용 가능해진다.

세그먼트 공간 재활용을 수행 후 세그먼트 내에 여유 공간이 마련되어 세그먼트 버퍼에 있는 소규모 데이터들을 재배치 할 수 있다. 일단 재활용 세그먼트의 여유 물리 블록번호를 세그먼트 버퍼의 데이터에 할당을 한 다음에 i-node와 Segment Summary[1]를 메모리 상에서 갱신한다. 이 과정을 재활용 세그먼트의 여유 공간이 없을 때까지 반복한 후에 변경한 데이터를 디스크로 요청을 보내면 소규모 데이터 재배치가 완료 된다.

4. 세그먼트 공간 재활용 기법의 최적화

본 절에서는 세그먼트 공간 재활용 기법을 효율적으로 수행하는 방법을 설명한다. Hot과 Cold 데이터를 분류를 하기 위해 WOLF[5]의 알고리즘을 적용하였다. 또한 세그먼트 공간 재활용을 위해 세그먼트 자체를 Hot과 Cold로 분류하였다.

즉석 분류된 데이터를 효율적으로 재배치 하기 위해서는 참조 지역성을 고려한 세그먼트 분류가 필요하다. Hot 세그먼트는 Hot 데이터로 이루어져 최근에 참조되었고 또한 많은 블록이 무효화 되어 낮은 사용률을 보인다. 그리고 Cold 세그먼트는 오래 전에 참조되고 데이터 블록이 자주 참조 되지 않아 유효한 데이터 블록이 많아 높은 사용률을 보인다[5]. 하지만 위의 이론을 토대로 Cold 데이터를 소규모 재배치 할 경우에 사용률이 높아 쓰기 효율이 저하된다. 때문에 참조 된지 오래된 것 중에 낮은 사용률의 세그먼트를 선택하도록 하였다. 이런 근거를 가지고 Hot 세그먼트, Cold 세그먼트를 선택할 수 있는 수식을 아래에 정의하였다.

$$Hottest\ Segment = min(U_{segment} * Age) \quad (1)$$

$$Coldest\ Segment = min(U_{segment} * Age^{-1}) \quad (2)$$

$U_{segment}$: 세그먼트의 사용률

Age : 세그먼트의 마지막 참조 시간

두 수식은 세그먼트의 사용률과 마지막 참조 시간을 이용하여 정의 하였다. 파일 시스템에서 여러 개의 세그먼트가 존재할 경우, $U_{segment} * Age$의 값이 가장 낮은 세그먼트를 Hot 세그먼트(식 (1))라 할 수 있고, $U_{segment} * Age^{-1}$의 값이 가장 낮은 세그먼트를 Cold세그먼트(식 (2))라 할 수 있다. 위 수식을 가지고 세그먼트 공간 재활용을 하면 비슷한 참조 지역성을 가지는 데이터들끼리 그룹화 된다. 결과적으로 쓰기 성능의 향상으로 이어진다.

5. 성능 평가

5.1 U-LFS 구현

세그먼트 공간 재활용 기법을 실험하기 위해서 U-LFS (User Level Log-Structured File System)라고 불리는 파일 시스템을 Linux Kernel상에서 FUSE (File-system In Userspace) 인터페이스[8]를 사용하여 구현하였다. U-LFS는 파일 시스템의 핵심 부분이 유저영역에서 실행이 되며, 실제 SSD(Solid State Drive), HDD (Hard Disk Drive)와 같은 저장장치를 사용할 수 있다.

U-LFS에 Cost-Benefit, WOLF, 세그먼트 공간 재활용(이하 Recycling이라 부름)의 세 가지 정책을 구현하여 성능을 비교하였다. 실험에서 사용한 환경은 표 1과 같으며, 운영체제가 설치 되지 않은 실험 전용 SSD와 HDD에서 파일 시스템의 성능을 측정하였다.

표 1 실험 환경

구분	제조사 규격 / 용량 / 모델명
OS	Ubuntu 7.10 (linux-2.6.22.16)
Processor	AMD 2.1GHz Opteron 1352
RAM	SAMSUNG 4GB DDR-2 800Mhz ECC
SSD	SAMSUNG 64GB MCC0E64G5MPP-0VA
HDD	HITACHI 160GB HDS721616PLA380

5.2 IO-TEST 벤치 마크(Synthetic Workloads)

본 논문에서 제안하는 Recycling과 Cost-Benefit 그리고 WOLF는 모두 빈 세그먼트가 없이 클리닝이나 데이터 재배치가 수행되는 경우에만 비교가 가능하다. 또

한 파일시스템 사용률에 따라 각 기법의 효율성, 나아가 성능이 다르기 때문에, 파일 시스템을 적절히 노화(Aging)시켜야 하고, 특정 사용률에 도달한 후에 성능을 비교할 필요가 있다. 위의 요구사항에 만족하기 위해서 IO-TEST라는 벤치마크 툴을 만들었다.

IO-TEST 툴에 사용한 작업 부하는 참조 지역성이 존재하는 Hot-Cold 쓰기 패턴과 참조 지역성이 없는 Uniform 쓰기 패턴이 있다. 두 가지 특성을 설명하면, Uniform은 모든 파일에 대한 참조가 균등하며, Hot Cold는 10% 범위의 파일 들이 90%만큼 참조가 되며 나머지 90%의 해당하는 파일들은 10%만 참조된다.

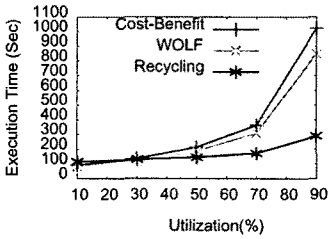
IO-TEST는 표 2와 같은 작업 부하를 생성하여 파일 시스템을 노화시키고 디스크의 사용률을 조절한다. 빠른 성능 측정과 기법간의 비교를 위하여 파티션의 크기는 4GB로 설정하였다. 세그먼트의 크기는 4MB이고 세그먼트 버퍼의 크기는 Cost-Benefit 정책을 사용시에 4MB, WOLF 및 Recycling은 Hot과 Cold버퍼 8MB (각 4MB+4MB)를 사용하였고 Recycling 정책을 제외한 나머지는 클리닝 버퍼를 추가적으로 사용하였다.

먼저 그림 1(a), (b)는 Uniform패턴으로 쓰기를 발생시켜 그 수행 시간을 측정한 결과로서, 파일 시스템 사용률에 따라 전체 수행 시간을 보여준다. Uniform패턴 쓰기의 결과를 보면 사용률이 높아질수록 다른 기법들은 수행 시간이 급격히 증가하는 반면, Recycling의 경우 수행 시간 증가율이 크기 않으며, 특히 파일시스템 사용률이 90%인 경우 WOLF의 기법보다 SSD에서 2배, HDD에서 3배 가량 좋은 성능을 보인다.

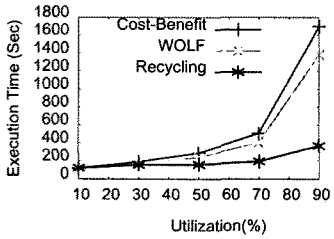
기존 Cost-Benefit과 WOLF기법의 경우 사용률이 증가함에 따라 수행 시간이 증가하며, 특히 사용률이 70% 이상이 되면 수행 시간이 급격히 증가하여 Recycling과의 성능 차가 점점 커진다. 이 실험에서 사용한 Uniform 패턴은 데이터의 참조 지역성이 거의 없으며, 이와 같이 데이터 지역성이 없는 경우 Cost-Benefit과 WOLF의 성능이 좋지 않음을 알 수 있다. 그러나 Recycling은 지역성이 없는 경우에도 세그먼트 클리닝을 수행하지 않으면서 데이터를 동적 재배치 하여 좋은 성능을 보여준다. 이 결과는 특히 Recycling은 지역성이 떨어지는 참조 패턴에서도 잘 동작함을 보여준다. 그림 1(c),(d)는 Hot-Cold패턴의 쓰기 에서 파일 시스템 사용률에 따라 수행 시간을 측정한 결과이다. 이 작업 부하는 강한 지역성을 가지고 있으며, 따라서 Cost-Benefit과 WOLF 기법의 성능이 Uniform패턴에 비해 좋음(실행 시간 감소)을 알 수 있다. 그렇지만 전반적으로 Recycling이 가장 좋은 성능을 보이며, 특히 파 일 시스템 사용률이 증가함에 따라 Recycling과 다른 두 기법의 성능 차이가 커진다. 특히 사용률이 높은 경우 Recycling은 WOLF와 비교하여 SSD에서 1.6배, HDD에서 1.9배 가량 성

표 2 IO-TEST 작업 부하 분류

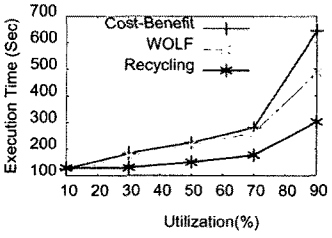
쓰기 패턴	파일사이즈	쓰기 횟수	쓰기 량
Uniform	32KB	131,072	4.0GB
Hot Cold	32KB	131,072	4.0GB



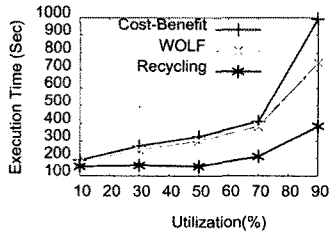
(a) Uniform 쓰기 패턴(SSD)



(b) Uniform 쓰기 패턴(HDD)

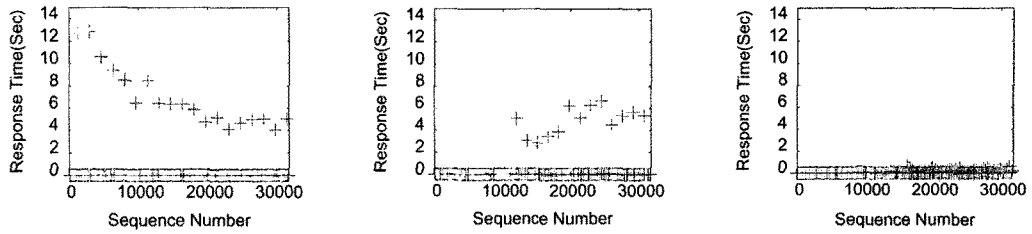


(c) Hot-Cold 쓰기 패턴(SSD)



(d) Hot-Cold 쓰기 패턴(HDD)

그림 1 사용률에 따른 쓰기 성능 측정



(a) Cost-Benefit (b) WOLF (c) Recycling

그림 2 32KB 쓰기 응답 시간 측정(디스크 사용률 90%)

능 향상을 보였다. 비록 Hot-Cold 패턴의 경우 지역성을 적절히 사용하여 Cost-Benefit과 WOLF의 클리닝 비용을 감소시킬 수 있지만, 결국 클리닝 동작의 존재만으로 전체 성능이 떨어지며, 클리닝을 전혀 수행하지 않고 데이터를 재배치하는 Recycling이 가장 효율적이라는 사실을 보여준다.

그림 2의 32KB 쓰기의 응답 시간의 측정 결과를 보면, Cost-benefit, WOLF기법은 최대 약 12초, 6초를 보이고 있는 반면에 Recycling은 최대 약 0.6초로 좋은 응답시간을 보이고 있다. 평균 응답시간의 결과는 Cost-Benefit(4,441us), WOLF(2,446us), Recycling(1,010us)이다. 기존의 클리닝 기법은 빈 세그먼트가 없는 경우에 이를 생성하는 클리닝 시간이 비교적 장시간 걸리기 때문에 전체적인 쓰기 응답 시간의 저하로 이어진다. 하지만 Recycling은 클리닝 과정이 없어 전체적으로 고른 응답시간을 보이며, 따라서 전체 쓰기 성능향상으로 이어진다.

표 3의 쓰기 통계 정보를 보면 Recycling이 기록한 총 데이터 크기는 약 5.27GB이고, WOLF는 23.17GB 그리고 Cost-Benefit은 27.00GB이다. 이 결과에 따르면 다른 기법들은 세그먼트 생성을 위해 클리닝을 수행하면서 많은 데이터를 복사해야 하지만, Recycling은 데이터 복사를 수행하지 않아 오버헤드가 현저히 감소하는 것을 알 수 있다.

표 3 32KB 쓰기 통계 정보 (디스크 사용률 90%)

	Cost-Benefit	WOLF	Recycling
클리닝 횟수	1458	1380	0
재활용 횟수	0	0	1112
읽기 (GB)	20.30	17.13	1.22
쓰기 (GB)	27.00	23.17	5.27

5. 결론 및 향후 연구

본 논문에서 제안하는 세그먼트 공간 재활용 기법은 클리닝을 수행하는 기존의 Cost-Benefit 그리고 WOLF 기법을 사용하는 LFS보다 좋은 성능을 보였다. 특히 쓰기가 계속적으로 요청되는 경우 클리닝은 전반적인 쓰기 지연을 유발하지만, Recycling은 쓰기 지연을 유발하지 않는다. 반대로 유휴 시간이 존재하는 경우 백그라운드 세그먼트 클리닝은 뒤따라 오는 쓰기 요청들을 최대의 대역폭으로 처리할 수 있도록 한다. 앞으로 이런 점들을 고려하여 클리닝과 Recycling 기법을 선택적으로 사용하는 알고리즘을 개발할 것이다.

참 고 문 헌

[1] M. Rosenblum, and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Trans. Comput. Syst.*, vol.10, no.1, pp. 26-52, 1992.

[2] M. Seltzer, K. Bostic, M. K. Mckusick et al., "An implementation of a log-structured file system for UNIX," in *Proceedings of the USENIX Winter Conference*, San Diego, California, 1993.

[3] T. Blackwell, J. Harris, and M. Seltzer, "Heuristic cleaning algorithms in log-structured file systems," in *Proceedings of the USENIX 1995 Technical Conference*, New Orleans, Louisiana, 1995.

[4] J. N. Matthews, D. Roselli, A. M. Costello et al., "Improving the performance of log-structured file systems with adaptive methods," in *Proceedings of the sixteenth ACM symposium on Operating systems principles*, Saint Malo, France, 1997.

[5] J. Wang, and Y. Hu, "WOLF - A Novel Reordering Write Buffer to Boost the Performance of Log-Structured File Systems," in *Proceedings of the Conference on File and Storage Technologies*, 2002.

[6] W. Wang, Y. Zhao, and R. Bunt, "HyLog: A High Performance Approach to Managing Disk Layout," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies, San Francisco*, CA, 2004.

[7] M. K. McKusick, W. N. Joy, S. J. Leffler et al., "A fast file system for UNIX," *ACM Trans. Comput. Syst.*, vol.2, no.3, pp.181-197, 1984.

[8] M. Szeredi, "Filesystem in userspace," Located at http://fuse.sourceforge.net, 2005.