creative commons

COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사학위논문

리매핑 기법을 이용한 리눅스 시스템 읽기/쓰기 성능 최적화 기법

2018년 2월

서울대학교 대학원

컴퓨터공학부

유 정 석

리매핑 기법을 이용한 리눅스 시스템 읽기/쓰기 성능 최적화 기법

지도교수 김 지 홍

이 논문을 공학석사 학위논문으로 제출함

2017년 10월

서울대학교 대학원
컴퓨터공학부
유 정 석

유정석의 석사 학위논문을 인준함
2017년 12월

위 원 장 _____ 하 순 회 (인)

부위원장 _____ 김 지 홍 (인)

위 원 _____ 유 승 주 (인)

요 약

낸드 플래시 메모리는 덮어쓰기가 불가능하여 호스트에서 덮어쓰기를 수행하면 새로운 물리 페이지에 수정된 데이터를 기록해야 한다. 이로 인해 호스트의 논리 블록 주소와 플래시 메모리의 물리 페이지 간의 매핑이 지속적으로 바뀌어 플래시 변환 계층에서는 이 정보를 주소 매핑 테이블에 저장하여 관리한다. 이러한 매핑 특성을 이용해 주소 매핑 테이블의 정보만을 수정하여 논리 블록 주소 상으로만 데이터를 이동/복사 시키는 기법을 리매핑이라고 한다.

리매핑을 이용하면 데이터의 논리 블록 주소를 유저가 원하는 대로 움직일 수 있어 이를 이용해 두 가지의 최적화가 가능하다.

먼저, 데이터의 이동/복사 시간을 크게 단축시킬 수 있다. 리매핑을 이용하면 물리 페이지의 읽기/쓰기 없이 메모리에 있는 매핑 테이블 만을 수정하여 전체 수행시간이 크게 줄어들게 된다.

또한, 시스템 내의 단편화를 제거하여 읽기 성능을 향상 시킬 수 있다. 단편화는 읽고자 하는 데이터들이 흩어져 저장되어 있는 경우를 말하는데, 리매핑을 이용해 이러한 단편화를 제거하면 소프트웨어 스택 오버헤드가 줄어들어 읽기 성능이 향상되게 된다.

본 논문에서는 앞서 말한 리매핑 기법으로 최적화 가능한 문제

들을 갖고 있는 리눅스 파일 유틸리티와 로그 구조 파일 시스템을 각각 소개하고 최적화한다. 리눅스 파일 유틸리티는 파일들을 처리하는 과정에서 파일들의 단편화로 인해 읽기 성능 하락이 발생하고, 로그 구조 파일 시스템은 데이터 쓰기 시 발생하는 단편화와 주기적인 세그먼트 클리닝으로 인해 읽기/쓰기 성능 하락이 발생하게 된다.

두 대상을 각각 리매핑 기법을 통해 최적화하여 리눅스 파일 유틸리티의 수행 속도는 최대 3.4배, 로그 구조 파일 시스템의 읽기/쓰기 성능은 각각 최대 2.1배, 2.2배 향상되는 것을 확인하였다.

주요어 : 낸드 플래시 메모리, 리매핑 기법, 파일 시스템 단편화, 리눅스 파일 유틸리티, 파일 시스템, 읽기/쓰기 성능 최적화

학 번 : 2016-21219

목 차

제 1 장 서 론 ····· 1

1.1 연구 배경 ····· 1

1.2 연구 동기 ····· 3

1.3 연구 기여 ····· 5

1.4 논문 구성 ····· 6

제 2 장 리매핑 기법 소개 ····· 7

2.1 리매핑 기법의 동작 과정 ····· 7

2.1 리매핑 기법의 효과와 문제점 ····· 9

제 3 장 리매핑 기법 적용 가능 문제 소개 ····· 11

3.1 리눅스 파일 유틸리티 ····· 11

3.2 로그 구조 파일 시스템 ····· 13

제 4 장 리매핑 기법을 통한 최적화 ····· 15

4.1 리눅스 파일 유틸리티 최적화 ····· 15

4.2 로그 구조 파일 시스템 최적화 ····· 19

제 5 장 실험 및 분석 22

5.1 실험 환경 22

5.2 실험 결과 23

제 6 장 결론 29

6.1 결론 29

6.2 향후 연구 30

참고문헌 31

Abstract 33

그림 목차

[그림 1] Samsung 830 SSD의 읽기/쓰기 요청 크기 별 성능 3
[그림 2] 리매핑 기법의 동작 과정 ····· 7
[그림 3] 리눅스 파일 유틸리티의 읽기 성능 저하 문제 ····· 10
[그림 4] 로그 구조 파일 시스템의 성능 저하 문제 ····· 12
[그림 5] 리매핑 유저 레벨 라이브러리의 동작 과정 ····· 16
[그림 6] 리매핑 조각모음 세그먼트 클리닝 동작 과정 ····· 19
[그림 7] 세그먼트 클리닝 동작 과정 비교 ····· 20
[그림 8] 파일 크기 별 qtar의 수행 시간 ····· 22
[그림 9] tar와 qtar의 Linux 4.4.52 처리 결과 분석 ····· 23
[그림 10] 대상 프로그램 디렉터리 별 qtar의 수행 시간 ····· 24
[그림 11] 리매핑 조각모음 세그먼트 클리닝의 쓰기 성능 ····· 25
[그림 12] 조각 모음 효과와 파일 읽기 성능 비교 ····· 26
[그림 13] 쓰기 요청 크기 별 단편화 수준 및 읽기 성능 ····· 27

표 목차

[표 1] 리매핑 유저 레벨 라이브러리의 함수 ····· 14
[표 2] 대상 프로그램 디렉터리 정보 ····· 24
[표 3] 매크로 벤치마크 동작 설명 ····· 25

제 1 장 서 론

1. 1 연구 배경

낸드 플래시 메모리(NAND flash memory)는 하드디스크(hard disk)에 비해 충격 및 고온에 강하고 빠른 임의 읽기/쓰기 속도를 갖고 있어 저장 매체 시장에서 하드디스크를 빠르게 대체하고 있다.

그러나 낸드 플래시 메모리는 하드디스크와 달리 덮어쓰기가 안 되는 특징이 있어 기존의 하드디스크와 동일하게 동작하기 위해서 플래시 변환 계층(flash translation layer, FTL)을 필요로 한다. 호스트에서 덮어쓰기가 내려오면 하드디스크는 기존의 물리적 위치에 덮어쓰기를 하는 반면, 플래시 변환 계층은 새로운 물리 페이지를 할당하여 데이터를 저장하고 해당 물리 페이지 주소(physical page address, PPA)를 덮어쓰기가 내려온 논리 블록 주소(logical block address, LBA)에 매핑시킨다. 이로 인해 호스트의 논리 블록 주소와 낸드 플래시 메모리의 물리 페이지 주소 간의 매핑이 고정되지 않고 지속적으로 바뀌어 플래시 변환 계층은 이 정보를 주소 매핑 테이블(address mapping table)에 저장하여 관리한다. 이러한 낸드 플래시 메모리의 매핑 특징을 이용해 주소 매핑 테이블의 정보만을 수정하여 논리 블록 주소 상으로만 데이터를 이동/복사시키는 기법을 리매핑이라고 한다[1, 2, 3].

리매핑은 낸드 플래시 메모리의 물리 페이지는 그대로 두고 이를 가리키고 있는 주소 매핑 테이블 만 수정하여 데이터를 이동/복사시킨다. 주

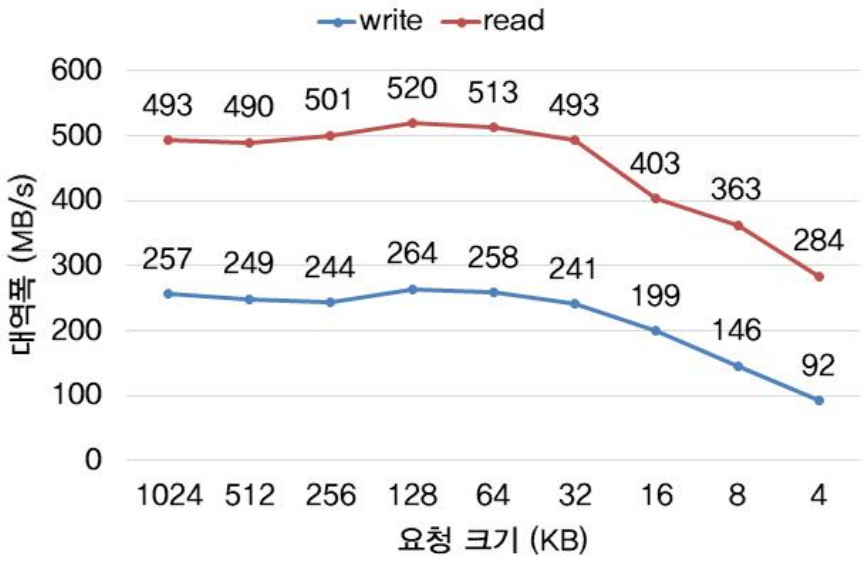
소 매핑 테이블은 낸드 플래시 메모리 내의 DRAM 공간에서 관리되어 리매핑을 이용한 이동/복사는 메모리 접근 시간만 발생하기 때문에 빠르게 완료된다. 즉, 리매핑 기법을 이용하면 논리 블록 주소 상의 데이터 저장 상태를 빠른 시간 안에 유저가 원하는 대로 바꿀 수 있게 된다.

리매핑을 이용해 최적화 할 수 있는 문제는 크게 두 가지가 있다. 먼저, 데이터의 이동/복사 시간을 크게 단축시킬 수 있다. 데이터의 이동/복사 시간은 대부분 낸드 플래시 메모리의 물리 페이지를 읽고 쓰는 시간이기 때문에, 리매핑을 이용해 데이터의 이동/복사를 물리 페이지의 읽기/쓰기 없이 메모리에 있는 매핑 테이블 만을 수정하여 완료하면 전체 수행시간이 크게 줄어들게 된다.

또한, 시스템 내의 단편화를 제거하여 읽기 성능을 향상 시킬 수 있다. 단편화는 읽고자 하는 데이터들이 흩어져 저장되어 있는 경우를 말하는데, 단편화 정도에 따라 소프트웨어 스택 오버헤드가 증가하여 읽기 성능이 저하되게 된다. 리매핑을 이용해 이러한 단편화를 제거하면 소프트웨어 스택 오버헤드가 줄어들어 읽기 성능이 향상되게 된다.

1. 2 연구 동기

낸드 플래시 메모리는 하드 디스크에 비해 높은 작은 읽기/쓰기 성능을 갖고 있다. 하지만 낸드 플래시 메모리 또한 읽기/쓰기의 크기가 작을수록 성능이 크게 저하된다. 그림 1은 Samsung 830 SSD의 읽기/쓰기 크기 별 성능을 나타낸 것이다. FIO 벤치마크[4]를 이용해 총 1GB의 읽기/쓰기를 다양한 크기로 나누어 발생시킨 후 대역폭을 측정하였는데, 그 결과 읽기/쓰기의 크기가 작아질수록 대역폭이 감소하여 약 2배의 성능 차이가 나는 것을 확인하였다. 따라서 시스템 내에서 발생하는 작은 크기의 읽기/쓰기들을 파악하여 이를 최적화하는 연구가 필요함을 알 수 있다. 특히, 작은 크기의 읽기/쓰기 중에서 1)데이터의 내용을 수정하지 않고 2)낸드 플래시 메모리의 물리 페이지 단위로 데이터를 이



[그림 1] Samsung 830 SSD의 읽기/쓰기 요청 크기 별 성능

동/복사하는 경우 리매핑 기법을 적용해 높은 성능 향상 효과를 얻을 수 있다. 본 논문에서는 이러한 조건을 만족하는 대상으로 리눅스 파일 유틸리티[5]와 로그 구조 파일 시스템[6]을 소개한다.

리눅스 파일 유틸리티는 파일에 특정 동작을 수행하는 리눅스 기본 유틸리티들을 말한다. 리눅스 파일 유틸리티는 유틸리티 마다 동작은 다르지만, 모두 파일 단위로 작업을 수행한다는 특징이 있다. 이때 읽기/쓰기 요청 또한 각 파일 단위로 발생하기 때문에, 각 파일의 크기가 동작 과정에서 발생하는 읽기/쓰기 요청들의 크기를 결정하게 된다. 따라서 대상 파일 중에 작은 파일이 많을 경우 극심한 성능 저하가 발생하게 된다.

로그 구조 파일 시스템은 모든 쓰기를 로그 헤더에 기록하기 때문에 여러 프로세스가 동시에 쓰기를 요청하면 각 프로세스의 데이터가 섞여 파일 단편화가 심하게 발생하게 된다. 이렇게 발생한 단편화는 파일 데이터 읽기 요청을 작은 크기의 읽기 요청으로 나누어지게 만들어 시스템의 읽기 성능을 저하시키는 주요인이 된다[7]. 또한 로그 구조 파일 시스템은 주기적으로 세그먼트 클리닝을 수행해야 하는데, 이 순간 쓰기 성능이 급격히 하락하는 문제가 있다. 그리고 세그먼트 클리닝을 수행한 후에도 파일 단편화는 사라지지 않고 유지되어 시간이 지날수록 단편화 수준이 더 심해지게 된다.

1. 3 연구 기여

본 논문의 기여는 크게 세 가지로 정리 할 수 있다.

첫째, 리매핑 기법을 소개하고, 효과와 한계에 대해 정리하였다. 리매핑 기법은 효과가 좋은 만큼 적용하기 위한 필요한 조건이 있다. 이 필요조건을 정리하여 특정 문제에 리매핑 기법의 적용 여부를 판단할 수 있도록 하였다.

둘째, 리매핑을 적용할 수 있는 문제 두 가지를 소개하고 최적화하였다. 정리한 필요조건을 만족하면서 리매핑의 성능 효과를 볼 수 있는 대상으로 리눅스 파일 유틸리티와 로그 구조 파일 시스템을 소개하고 리매핑을 적용하여 각각의 성능 향상 효과를 확인하였다.

셋째, 리매핑 기능을 지원하는 에뮬레이터를 구현하였다. 현재 상용화되어 있는 SSD들은 리매핑 기능을 지원하지 않는다. 따라서 SSD 에뮬레이터[8]에 리매핑 기능을 구현하여 성능 향상 효과를 검증하였다.

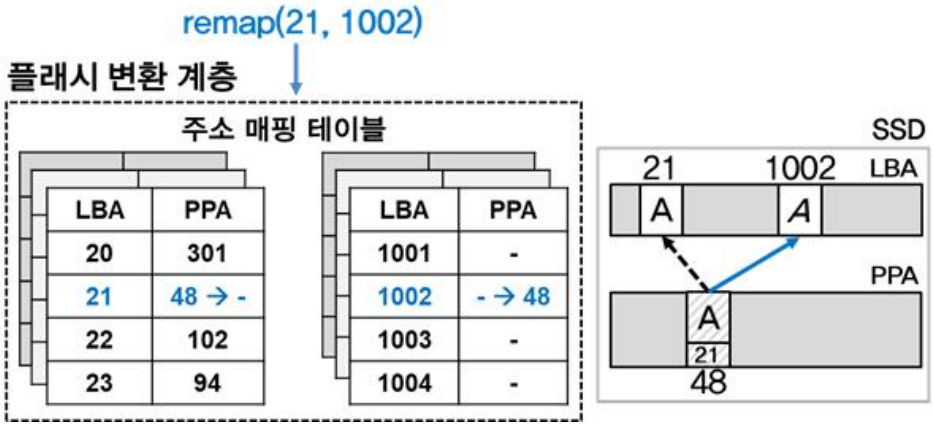
1. 4 논문 구성

본 논문의 구성은 다음과 같다. 2장에서는 리매핑 기법의 동작 과정과 효과 및 한계에 대해서 설명한다. 3장에서는 리매핑 기법을 적용할 수 있는 대상으로 리눅스 파일 유틸리티와 로그 구조 파일 시스템을 소개하고, 각각에서 발생하는 성능 저하 문제를 분석한다. 4장에서는 소개한 두 가지 문제를 리매핑 기법으로 최적화하고 기존 동작 과정과 최적화 후 동작 과정을 비교 및 분석 한다. 5장에서는 리매핑을 통한 최적화 효과를 SSD 에뮬레이터를 통한 실험으로 검증한다. 마지막으로 6장에서는 본 연구의 결론 및 향후 연구 방향에 대해 기술한다.

제 2 장 리매핑 기법 소개

2. 1 리매핑 기법의 동작 과정

리매핑 기법이란, 주소 매핑 테이블의 매핑 정보만을 수정하여 데이터를 이동시킨 효과를 얻는 기법을 말한다[1, 2, 3]. 그림 2는 리매핑 기법의 동작 과정을 나타낸 것이다. 먼저 호스트에서 리매핑 명령어 remap(sLBA, tLBA)을 통해 소스(source) 논리 블록 주소(LBA)와 타깃(target) 논리 블록 주소(LBA)를 플래시 변환 계층으로 넘겨준다. 플래시 변환 계층은 넘어온 소스 논리 블록 주소를 주소 매핑 테이블에서 넣어 매핑 되어 있는 물리 페이지 주소를 알아낸다. 그리고 주소 매핑 테이블의 타깃 논리 블록 주소가 이 물리 페이지 주소를 가리키게 한



[그림 2] 리매핑 기법의 동작 과정

후 소스 논리 블록 주소의 매핑 정보를 삭제한다. 모든 과정이 끝나면 호스트 입장에서는 소스 논리 블록 주소의 데이터가 타깃 논리 블록 주소로 물리적으로 이동한 것으로 보이게 된다. 만약 소스 논리 블록 주소의 매핑 정보를 삭제하지 않으면, 데이터를 복사한 효과를 얻게 된다.

2. 2 리매핑 기법의 효과와 한계

주소 매핑 테이블은 낸드 플래시 메모리 내부의 DRAM에 올려져있기 때문에 리매핑을 통한 데이터의 이동은 물리적으로 데이터 쓰기가 발생하지 않기 때문에 마치 요청하자마자 데이터가 이동/복사 된 것처럼 빠르게 완료된다. 그리고 원래 발생했을 쓰기가 발생하지 않게 되기 때문에 쓰기 횟수 제한이 있는 낸드 플래시 메모리의 수명을 늘이는 효과 또한 얻을 수 있다.

그러나 리매핑 기법은 매핑 테이블 형성에 쓰이는 데이터가 저장된 물리 페이지의 OOB영역의 대상 논리 블록 주소 값을 손상시키기 때문에 이를 따로 관리해주는 기법이 필요하다[7]. OOB영역의 대상 논리 블록 주소 값이란, 물리 페이지에 데이터를 기록할 때 매핑된 논리 블록 주소를 같이 적어놓은 것을 말한다. 이 정보는 플래시 변환 계층에서 세그먼트 클리닝을 수행할 때나 갑작스런 전원 차단으로 주소 매핑 테이블이 손상되었을 때 이를 복구하기 위해 사용되어 항상 올바른 값을 갖고 있어야 한다. 그림 2를 보면 리매핑이 끝난 후 물리 페이지 주소 48에 저장되어 있는 논리 블록 주소의 값을 보면 여전히 21임을 알 수 있다. 이 상태에서 전원이 차단된 후 매핑 테이블을 복구하면 물리 페이지 주소 48이 논리 페이지 주소 21에 매핑되어 버리는 문제가 발생한다. 결국 리매핑 기법을 올바르게 사용하기 위해서는 리매핑 결과를 로깅(logging)하고 세그먼트 클리닝과 전원 차단 시 참고하도록 하는 관리 기법이 추가되어야 한다.

그러나 가장 중요한 한계점은 물리 페이지를 읽거나 쓰지 않기 때문에, 물리 페이지의 데이터를 수정해야 하거나 물리 페이지 크기 단위와 다른 크기 단위의 데이터를 이동/복사 시키는 경우에는 리매핑 기법을 사용할 수 없다는 것이다.

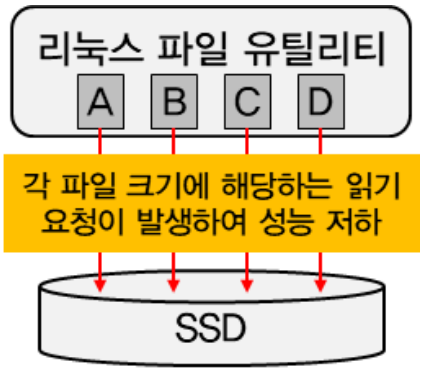
즉, 리매핑 기법은 확실한 성능 향상 효과를 보장하지만, 적용하기 위한 조건이 까다롭고 이를 관리하기 위한 기법이 추가적으로 필요하다. 따라서 리매핑 기법을 활용하기 위해서는 관찰한 문제가 리매핑 적용에 적합한지에 대한 여부를 먼저 확실히 분석하는 것이 필요하다.

제 3 장 리매핑 기법 적용 가능 문제 소개

3. 1 리눅스 파일 유틸리티

리눅스 파일 유틸리티는 대상 파일들의 크기가 작을 때 성능이 급격히 하락하는 문제가 있다. 그 이유는 리눅스 파일 유틸리티는 모든 동작을 파일 단위로 수행하기 때문에 데이터의 읽기/쓰기도 파일 단위로 이루어지기 때문이다. 이 때문에 파일의 크기가 작거나, 파일의 데이터가 작은 조각으로 단편화 되어 있을 경우 극심히 읽기 성능 저하가 발생한다.

파일 단위로 발생하는 읽기로 인한 성능 저하는 여러 파일의 데이터를 미리 읽어오는 것으로 해결할 수 있다. 그러나 미리 읽어올 각 파일들의 크기가 작고 흩어져 저장되어 있을 경우 파일 단편화로 인해 읽기 성능이 저하되는 것과 같은 이유로 읽기 성능이 저하되게 된다.



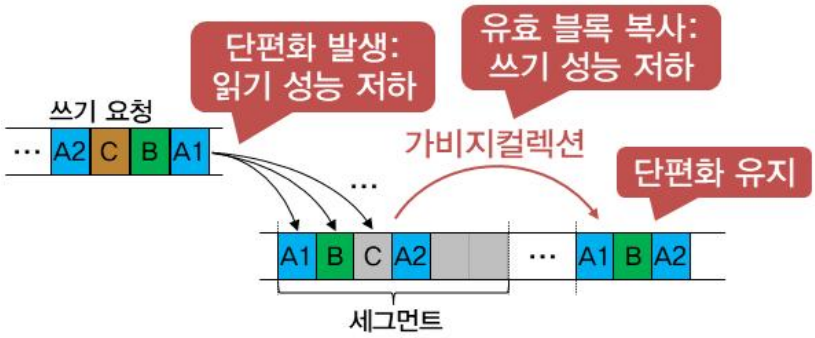
[그림 3] 리눅스 파일 유틸리티의 읽기 성능 저하 문제

따라서 리눅스 파일 유틸리티의 읽기 성능을 최적화하기 위해서는 1) 데이터의 읽기를 파일 하나 단위가 아닌 여러 파일 단위로 요청해야 하고 2)함께 요청된 파일들의 데이터가 연속된 논리 블록 주소에 위치해 있도록 만들어야 한다.

3. 2 로그 구조 파일 시스템

로그 구조 파일 시스템은 새로 들어온 데이터와 수정된 데이터를 모두 로그 헤더가 가리키고 있는 여유 공간에 추가하기 때문에 높은 임의 쓰기 성능을 갖고, 이전 데이터가 남기 때문에 시스템 복구가 쉬운 장점을 갖는다. 그러나 무효화된 이전 데이터들이 계속 남아있기 때문에 주기적으로 세그먼트 클리닝을 수행하여 이를 정리해주어야 한다. 이때, 유효한 블록을 골라내어 다시 로그 헤더에 복사하게 되는데 이 과정동안 시스템의 쓰기 성능이 크게 저하되는 문제가 발생한다.

또한 들어오는 모든 데이터를 로그 헤더에 추가하기 때문에 여러 파일의 쓰기가 동시에 발생한 경우 파일들의 데이터가 섞여 단편화가 발생한다. 파일이 단편화되면 파일 읽기 시 작은 크기의 읽기 요청을 발생시켜 시스템의 읽기 성능이 저하되는 문제가 발생한다.



[그림 4] 로그 구조 파일 시스템의 성능 저하 문제

따라서 로그 구조 파일 시스템의 읽기/쓰기 성능을 최적화하기 위해서는 세그먼트 클리닝의 유효 블록 복사 시간을 최소화하고, 파일 단편화를 주기적으로 제거해주는 기법이 필요하다.

위에서 제시한 읽기/쓰기 성능을 각각을 최적화한 연구는 이미 진행되었다[9, 10]. 그러나 이 두 가지 기법을 한 번에 적용하여 로그 구조 파일 시스템의 전반적인 읽기/쓰기 성능을 모두 최적화하려는 연구는 아직 진행되지 않았다. 본 연구에서는 두 기법을 모두 적용하여 각각의 연구에서 제시했던 성능 향상 효과가 제대로 나타나는 것을 확인하고 향후 연구로 두 가지 기법의 취약점을 파악하여 개선하는 것을 목표로 한다.

제 4 장 리매핑 기법 적용을 통한 최적화

4. 1 리눅스 파일 유틸리티 최적화

4. 1. 1 리매핑 유저 레벨 라이브러리

본 연구에서는 리눅스 파일 유틸리티의 읽기 요청을 여러 파일 단위로 발생하도록 바꿔주고 요청된 데이터들이 모두 연속된 논리 블록 주소에 놓여있도록 만들어주는 리매핑 유저 레벨 라이브러리를 구현하였다. 리매핑 유저 레벨 라이브러리는 리눅스 파일 유틸리티에서 발생하는 읽기 요청을 받아 최대 크기의 읽기 요청으로 변환한다. 따라서 리눅스 파일 유틸리티는 대상 파일들의 크기와 단편화 정도에 관계없이 항상 최대 대역폭을 활용할 수 있게 된다. 리매핑 유저 레벨 라이브러리는 리눅스 파일 유틸리티의 소스 코드에 리매핑 함수들을 추가하여 적용 시킬 수 있다. 리매핑 함수는 표 1과 같이 remap_create(), remap_delete(), remap_read() 세 가지로 구성되어 있다. 리눅스 파일 유틸리티의 타깃

함수 명	동작 설명
create_remap()	대상 파일들의 크기 합에 해당하는 임시 파일을 생성하고 모든 데이터를 갖도록 리매핑
read_remap()	리매핑한 파일을 통해 데이터를 크게 읽고 읽기 요청을 한 파일의 데이터를 반환
delete_remap()	임시 파일의 매핑정보를 삭제하고 임시 파일 삭제

[표 1] 리매핑 유저 레벨 라이브러리의 함수

파일들의 리스트를 remap_create()로 넘겨주고, 리눅스 파일 유틸리티 내의 read() 시스템 콜(system call)을 remap_read()로 대체시키고 프로그램 종료 시점에 remap_delete()를 추가하면 리매핑을 통한 성능 향상 효과를 볼 수 있다. 다시 말해, 개발자는 각 리눅스 파일 유틸리티의 소스 코드를 완전히 이해할 필요 없이 타깃 파일들의 리스트가 저장된 위치와 read() 시스템 콜의 위치만 파악하면 간단하게 리매핑 기법을 적용할 수 있다.

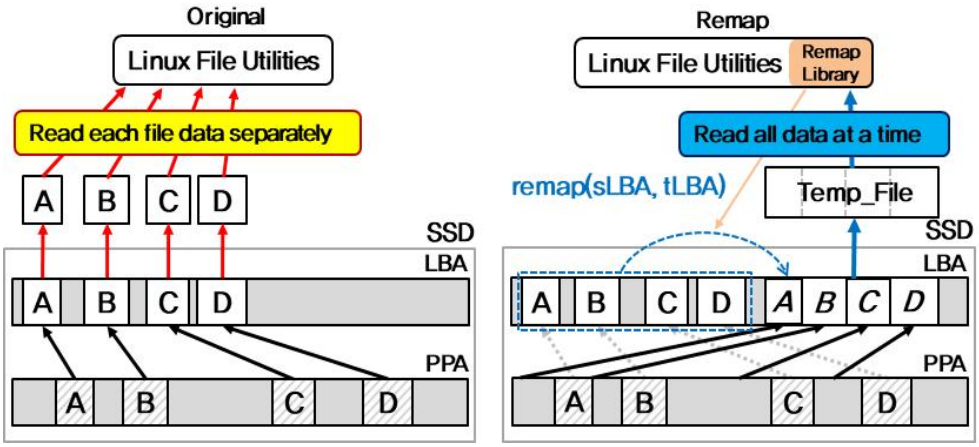
4. 1. 2 리매핑 유저 레벨 라이브러리의 동작 과정

유저가 대상 파일들의 리스트와 함께 리눅스 파일 유틸리티를 실행 시키면, 리매핑 유저 레벨 라이브러리는 리눅스 파일 유틸리티의 실행을 멈추고 유저가 입력한 대상 파일들의 리스트를 받아온다. 그리고 대상 파일들의 전체 크기 합을 구해 이와 같은 크기를 갖는 임시 파일(Temp_File)을 생성하고, 리매핑을 통해 이 파일이 대상 파일들의 데이터를 읽혀질 순서에 맞게 갖고 있도록 만든다. 다시 말해, 모든 대상 파일들의 데이터를 각 데이터가 읽혀질 순서에 따라 하나의 임시 파일이 갖고 있도록 만든다. 그리고 리눅스 파일 유틸리티에서 발생하는 읽기 요청들이 모두 리매핑 라이브러리로 들어오도록 설정한 후 리눅스 파일 유틸리티를 실행 시킨다. 리눅스 파일 유틸리티가 실행되어 첫 번째 파일 데이터에 대한 읽기가 발생하면, 리매핑 유저 레벨 라이브러리는 데이터 버퍼를 생성하고 임시 파일의 전체 데이터를 모두 읽어 이 버퍼에 저장한 후 각 파일의 읽기 요청에 해당하는 데이터를 이 데이터 버퍼에

서 꺼내 반환 해준다. 만약 임시 파일의 크기가 너무 커서 전체가 메모리에 수용될 수 없는 경우, 수용 가능한 만큼만 먼저 읽어 처리한 후 나머지 데이터를 읽어온다. 결과적으로 리매핑 유저 레벨 라이브러리는 모든 파일들의 데이터를 한 번에 읽어 처리하기 때문에 대상 파일들의 크기가 작을 때 발생하던 읽기 성능 하락 문제를 해결하게 된다.

4. 1. 3 리매핑 유저 레벨 라이브러리가 적용 전과 후의 동작 과정 비교

그림 5는 기존 리눅스 파일 유틸리티와 리매핑 유저 레벨 라이브러리가 적용된 리눅스 파일 유틸리티의 동작 과정을 비교한 것이다. 기존 리눅스 파일 유틸리티의 경우 한 파일의 처리가 모두 끝난 후 다음 파일의 읽기 요청이 발생하기 때문에 항상 각 파일의 크기에 해당하는 읽기 요청이 발생하는 것을 볼 수 있다. 반면에 리매핑 유저 레벨 라이브러리가



[그림 5] 리매핑 유저 레벨 라이브러리의 동작 과정

적용된 경우, 모든 파일들의 데이터를 하나의 임시 파일로 리매핑 한 후 한 번에 모든 데이터를 읽어와 항상 최대 크기의 읽기 요청이 발생하는 것을 볼 수 있다. 따라서 리매핑 유저 레벨 라이브러리의 성능 향상 효과는 각 파일의 크기가 작아질수록 커지게 된다.

4. 2 로그 구조 파일 시스템 최적화

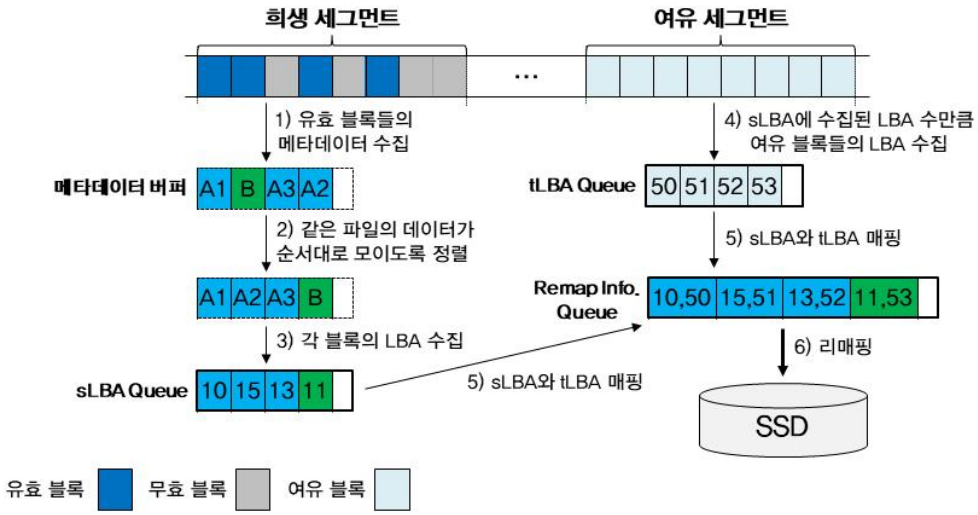
4. 2. 1 리매핑 조각모음 세그먼트 클리닝

본 연구에서는 로그 구조 파일 시스템의 세그먼트 클리닝에 리매핑 기법과 조각모음 기능을 추가하여 빠르게 완료되며 단편화를 제거해주는 리매핑 조각모음 세그먼트 클리닝을 구현하였다. 리매핑 조각모음 세그먼트 클리닝은 빠른 시간 안에 완료되며 조각모음도 수행하기 때문에 로그 구조 파일 시스템의 쓰기 성능과 읽기 성능을 모두 크게 향상 시킬 수 있다.

4. 2. 2 리매핑 조각모음 세그먼트 클리닝의 동작 과정

그림 6은 리매핑 조각모음 세그먼트 클리닝의 동작 과정을 나타낸 것이다. 먼저 1)세그먼트 클리닝의 타깃 세그먼트에서 유효 블록들의 메타데이터를 읽어 메타데이터 버퍼에 저장한다. 그리고 2)inode와 offset정보를 이용해 같은 파일의 데이터가 순서에 맞게 모이도록 정렬하고, 3)각 블록의 논리 블록 주소를 수집하여 소스 논리 블록 주소 큐(source LBA queue)에 저장한다. 다음으로 4)유효 블록들을 복사할 여유 세그먼트에서 준비된 소스 논리 블록 주소의 수에 해당하는 여유 블록들의 논리 블록 주소를 수집하여 타깃 논리 블록 주소 큐(target LBA queue)에 저장한다. 그리고 5)소스 논리 블록 주소 버퍼와 타깃

논리 블록 주소 버퍼의 각 항목들을 순서대로 짝지어 리매핑 정보 큐(remap info queue)에 저장한다. 마지막으로 6)리매핑 정보 큐를 이용해 리매핑 명령을 생성하여 낸드 플래시 메모리에 요청하여 리매핑 세그먼트 클리닝을 완료한다.

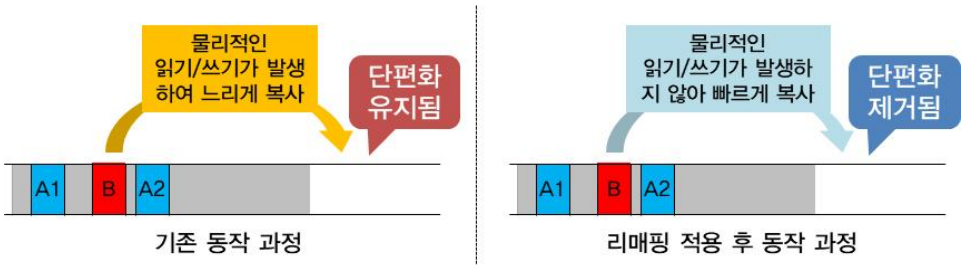


[그림 6] 리매핑 조각모음 세그먼트 클리닝 동작 과정

4. 2. 3 기존 세그먼트 클리닝과 리매핑 조각모음 세그먼트 클리닝의 동작 과정 비교

그림 7은 로그 구조 파일 시스템의 기존 세그먼트 클리닝과 리매핑과 조각모음이 적용된 세그먼트 클리닝의 동작 과정을 비교한 것이다. 기존의 세그먼트 클리닝은 타깃 세그먼트 내 유효 블록들을 찾아 여유 세그먼트에 복사한다. 이 과정에서 물리적인 읽기/쓰기가 발생하기

때문에 타깃 세그먼트 내 유효 블록의 수가 많을수록 세그먼트 클리닝의 수행시간이 길어지게 된다. 또한 유효 블록의 순서를 그대로 복사하기 때문에 기존에 존재하던 파일의 단편화가 그대로 유지되는 것을 볼 수 있다. 그러나 리매핑과 조각모음이 적용된 경우 유효 블록 복사에 물리적인 읽기/쓰기가 발생하지 않고 논리적인 연산만 발생하기 때문에 빠른 시간 안에 유효 블록 복사가 끝나게 된다. 또한 유효 블록 들을 메타 데이터를 통해 정렬하여 복사하기 때문에 타깃 세그먼트에서 존재하던 파일 단편화가 모두 사라지게 된다.



[그림 7] 세그먼트 클리닝 동작 과정 비교

제 5 장 실험 및 분석

5. 1 실험 환경

본 연구의 실험은 Intel i7-2600 CPU, 16 GB 메인 메모리 환경에서 수행하였다. 저장 장치는 상용화되어 있는 낸드 플래시 메모리는 리매핑 기능이 없기 때문에 SSD 에뮬레이터[3]에 리매핑 기능을 추가하여 실험하였다. 리매핑 유저 레벨 라이브러리의 실험은 Linux 4.4.52, F2FS 파일 시스템의 실험은 Linux 4.10.17 환경에서 이루어졌다.

실험은 먼저 마이크로 벤치마크(micro benchmark)를 이용해 최소/최대 성능 향상 효과를 확인한 후, 매크로 벤치마크(macro benchmark)를 이용해 다양한 시나리오에서의 성능 향상 효과를 확인하였다.

벤치마크 이름	동작 설명
copyfiles	1MB 파일들을 순차적으로 복사
tiobench	4개의 쓰레드가 4KB-16KB의 크기로 임의 쓰기
varmail	1000개의 파일을 16개의 쓰레드가 생성 후 fsync
dbench	8개의 쓰레드가 여러 기존 파일 연산 작업 후 fsync

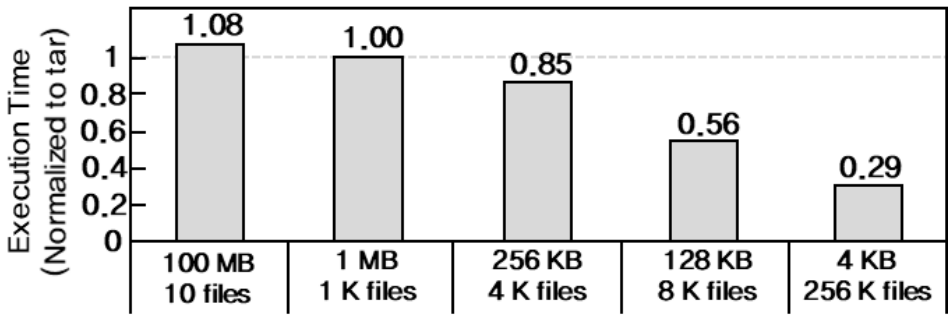
[표 2] 매크로 벤치마크 동작 설명

5. 2 실험 결과

5. 2. 1 리매핑 유저 레벨 라이브러리

리매핑 유저 레벨 라이브러리는 리눅스 파일 유틸리티 중 tar에 리매핑 기법을 적용하여 성능 향상 효과를 확인하였다. 먼저 대상 크기 별 성능 향상 효과를 확인하기 위해 총 1GB가 되는 다양한 크기의 파일들을 기존 tar로 처리하는데 걸리는 시간과 qtar(리매핑이 적용된 tar)가 처리하는 시간을 비교하였다. 다음으로 리매핑 적용 효과를 자세히 파악하기 위해 Linux 4.4.52 소스 코드를 기존 tar가 처리하는 과정과 qtar가 처리하는 과정에서 발생하는 논리 블록 주소 접근을 로깅하여 분석하였다. 마지막으로 실제 파일들을 처리하는 시간을 비교하기 위해 여러 종류 프로그램 파일들을 기존 tar와 qtar가 처리하는 시간을 비교하였다.

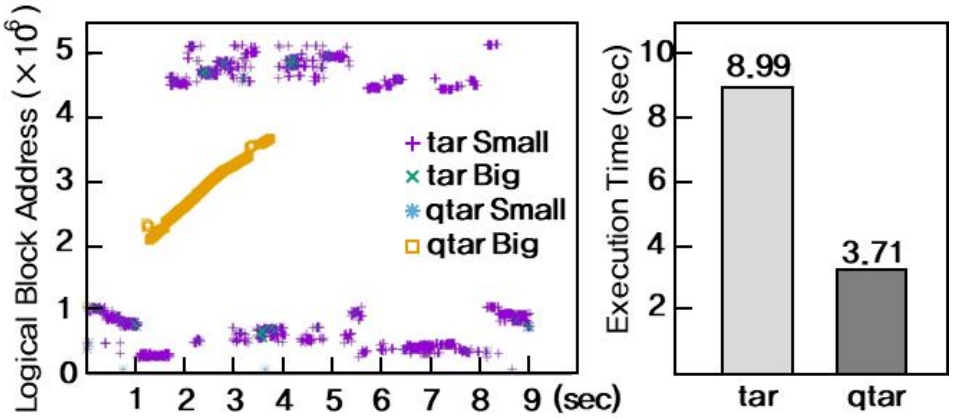
그림 8은 총 1 GB가 되는 다양한 크기의 파일들을 tar와 qtar(리매핑이 적용된 tar)로 처리하는데 걸린 시간을 tar를 기준으로 정규화 하여



[그림 8] 파일 크기 별 qtar의 수행 시간

나타낸 것이다. qtar는 읽기 성능을 향상시키기 위해 리매핑을 이용한다. 이는 추가된 작업이기 때문에 만약 테이핑 대상이 읽기 성능을 개선할 여지가 없이 이미 큰 파일들로 이루어진 경우 오히려 qtar의 실행 시간이 tar에 비해 증가할 수 있다. 그러나 파일 크기가 1 MB 이하일 경우 qtar가 tar보다 항상 빠르고, 파일 크기가 4 KB 일 경우 qtar가 tar보다 3.4배 빠르게 동작하였다.

그림 9는 Linux 4.4.52 (725 MB, 파일 58779 개)를 tar와 qtar를 이용해 처리한 결과를 비교한 것이다. 그림 9(a)는 tar와 qtar에서 발생한 읽기 요청들을 먼저 크기순으로 분류한 후 각 요청들이 접근한 논리 블록 주소를 시간 순으로 나타낸 것이다. tar는 파일 하나 단위로 읽기 요청이 발생하여 주로 작고 비순차적인 읽기 요청들이 발생하고 있는 것에 반해 qtar는 리매핑 기법으로 인해 주로 크고 순차적인 읽기 요청들이 발생하고 있는 것을 볼 수 있다. 이러한 읽기 요청 패턴의 변화는 I/O 성능을 크게 향상 시켜 그림 9(b)와 같이 전체 실행 시간이 감소되



(a) 읽기 요청 LBA 접근 패턴 (b) 수행 시간

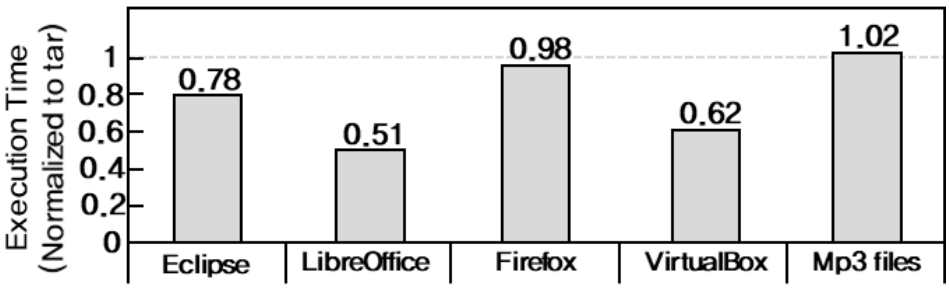
[그림 9] tar와 qtar의 Linux 4.4.52 처리 결과 분석

는 효과를 얻게 된다. 여러 차례의 실험을 통해 Linux 4.4.52를 처리하는 속도가 qtar가 tar에 비해 약 2.4배 향상된 것을 확인하였다.

표 3과 그림 10은 리눅스에서 자주 사용되는 프로그램들 중 다섯 가지를 선정하여 qtar와 tar의 실행 시간을 비교한 결과이다. LibreOffice와 VirtualBox와 같이 작은 파일을 많이 포함하고 있는 디렉터리의 경우 qtar가 tar보다 약 2배의 성능 향상이 있는 것을 확인할 수 있었다. 다만 Mp3파일처럼 각 파일의 용량이 크거나 Firefox처럼 큰 파일이 전체 용량의 절반이상을 차지하는 경우, 성능향상이 거의 없거나 약간의 성능 하락이 발생하는 것을 확인하였다.

Name	*Total size (MB)*	*Num. of files*
Eclipse	367	2973
LibreOffice	979	77639
Firefox	141	66
VirtualBox	511	30152
Mp3 files	441	89

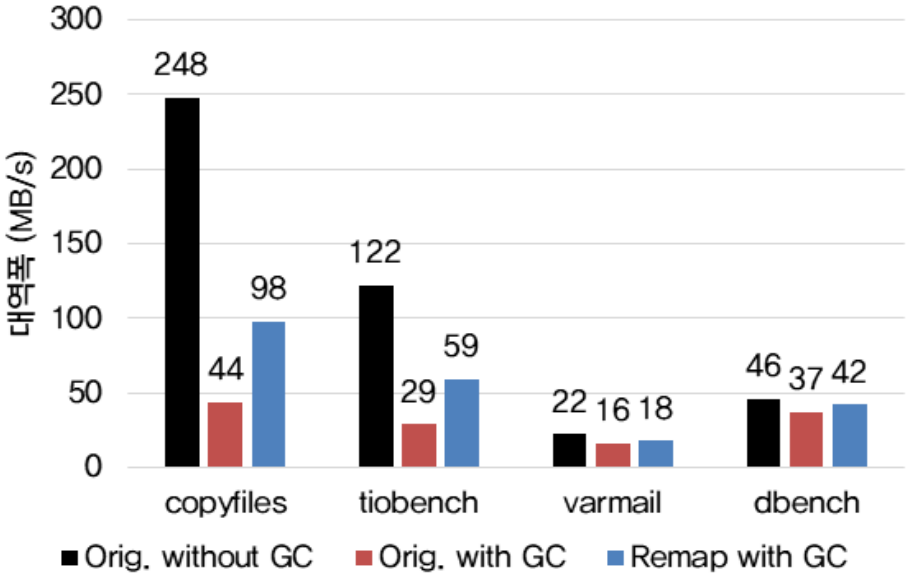
[표 3] 대상 프로그램 디렉터리 정보



[그림 10] 대상 프로그램 디렉터리 별 qtar의 수행 시간

5. 2. 2 리매핑 조각모음 세그먼트 클리닝

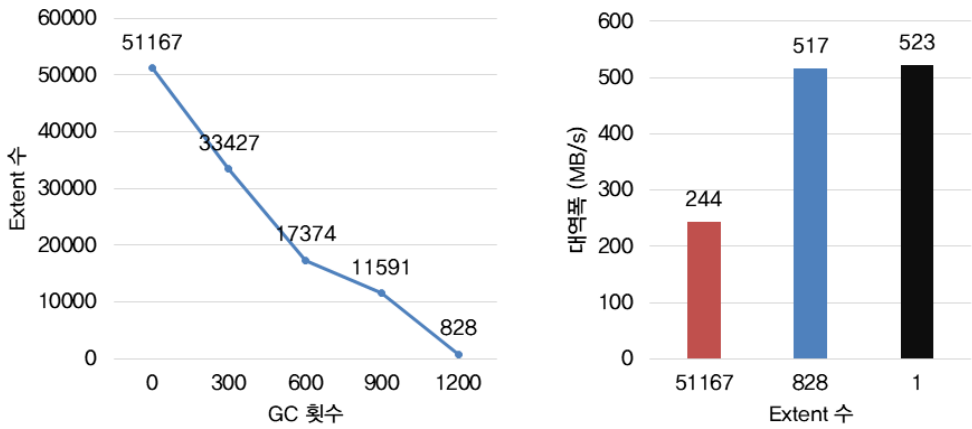
리매핑 조각모음 세그먼트 클리닝은 F2FS 파일 시스템의 세그먼트 클리닝에 리매핑 기법을 적용하여 성능 향상 효과를 검증하였다. 먼저 그림 11은 리매핑을 통한 쓰기 성능 향상 효과를 확인하기 위해 다양한 벤치마크를 실행하며 세그먼트 클리닝이 발생하는 동안 쓰기 성능을 비교한 것이다. 실험 결과 copyfiles와 tiobench는 세그먼트 클리닝 발생 시 유효 블록 복사의 비중이 높아 쓰기 성능이 2배 이상 향상되는 것을 볼 수 있다. 그러나 varmail과 dbench는 좁은 논리 블록 주소 공간에서 집중적으로 쓰기가 발생하여 세그먼트 클리닝 발생 시 유효 블록 복사가 거의 발생하지 않아 리매핑을 적용해도 쓰기 성능이 향상되지 않는 것을 확인 하였다. 다음으로 그림 12 (a)는 조각모음의 효과를 확인하기 위



[그림 11] 리매핑 조각모음 세그먼트 클리닝의 쓰기 성능

해 동시에 여러 파일의 데이터를 4 KB 단위로 기록하여 파일 단편화를 발생 시킨 후 세그먼트 클리닝을 지속적으로 유발하며 단편화가 줄어드는 것을 관찰한 것이고, 그림 12 (b)는 최종적으로 단편화가 줄어든 파일과 처음 기록한 상태의 파일을 읽을 때의 성능을 비교한 것이다. 실험 결과 처음 쓰기 완료 시 51167개로 흩어져있던 파일의 데이터가 세그먼트 클리닝이 진행되면서 결국 828개의 조각으로 합쳐지는 것을 볼 수 있다. 이렇게 파일의 데이터가 828개의 조각으로 합쳐진 효과로 파일 읽기 성능이 약 2.1배 향상 된 것을 확인하였다. 이는 단편화가 전혀 없는 상태의 파일 읽기 성능과 비슷한 수준인 것을 알 수 있다.

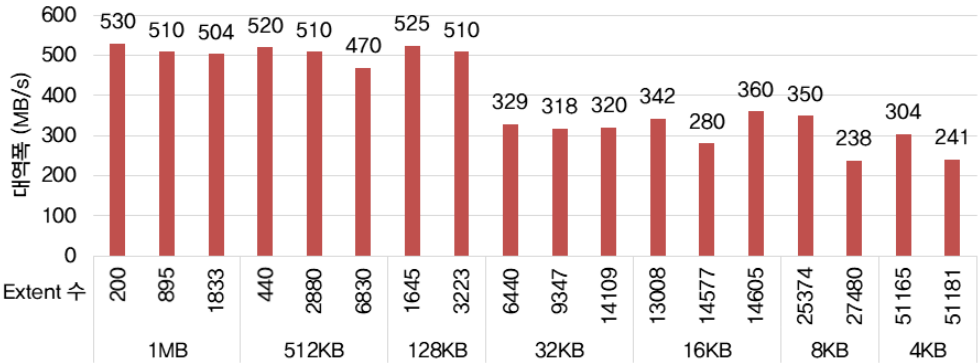
그림 13은 파일의 쓰기 요청 크기 별 단편화 수준을 분석한 것이다. 그래프를 보면 데이터의 단편화 조각 수와 읽기 성능이 비례하지 않고, 파일 쓰기 요청의 크기가 단편화로 인한 읽기 성능 저하 수준에 영향을



(a) 조각 모음 효과 (b) 파일 읽기 성능

[그림 12] 조각 모음 효과와 파일 읽기 성능 비교

주는 것을 확인할 수 있다. 결국 파일 단편화로 인한 성능 저하를 없애기 위해서는 전체 단편화 수를 줄이는 것보다는 각 단편화 조각의 크기를 크게 만들어야 하는 것을 알 수 있다. 본 연구에서 구현한 리매핑 조각모음 세그먼트 클리닝은 한 세그먼트 내의 단편화를 없애 각 단편화 조각의 크기를 키우는 기능을 하여 효과적으로 읽기 성능을 최적화하게 된다.



[그림 13] 쓰기 요청 크기 별 단편화 수준 및 읽기 성능

제 6 장 결 론

6. 1 결 론

본 논문에서는 낸드 플래시 메모리의 특성으로 사용이 가능해진 리매핑 기법에 대한 동작 과정과 효과 및 한계를 소개한다. 그리고 리눅스 시스템 내에서 리매핑 기법의 적용으로 성능 향상 효과를 얻을 수 있는 대상들을 찾아 소개하고 리매핑 기법을 적용하여 성능 향상 효과를 검증하였다. 적용 가능 문제로써 리눅스 파일 유틸리티와 로그 구조 파일 시스템을 소개하고 이 두 대상에서 발생하는 성능 하락 문제의 원인과 리매핑을 통한 최적화 방법을 고안하였다. 리매핑을 적용한 효과로 리눅스 파일 유틸리티에서는 수행 속도가 최대 3.4배, 로그 구조 파일 시스템에서는 쓰기 성능이 최대 2.2배 읽기 성능은 2.1배가 향상된 것을 확인하였다.

6. 2 향후 연구

본 논문에서는 크게 두 가지 문제에 리매핑 기법을 적용하여 성능 향상 효과를 확인하였다. 이 중에서 로그 구조 파일시스템의 세그먼트 클리닝에 추가한 조각모음은 상황에 따라 효율이 많이 떨어질 수 있다. 세그먼트 클리닝 시 한 세그먼트 내에 있는 단편화만 제거할 수 있기 때문에 만약 한 파일의 데이터가 여러 세그먼트에 한 블록 씩 퍼지게 되면, 조각모음의 효과를 전혀 볼 수 없게 된다. 이러한 경우를 방지하기 위해 세그먼트 클리닝 시 파일의 크기와 단편화 수를 고려하여, 파일 단위 조각 모음이 필요한 파일을 찾아내어 적용하는 기법이 추가적으로 필요하다.

참 고 문 헌

[1] H. Choi, S. Lim, and K. Park, "JFTL: A Flash Translation Layer based on a Journal Remapping for Flash Memory," ACM Transactions on Storage, Vol. 4, No. 14, 2009.

[2] H. Kim, D. Shin, Y. Jeong, and K. Kim, "SHRD: Improving Spatial Locality in Flash Storage Accesses by Sequentializing in Host and Randomizing in Device," Proc. of the USENIX Conference on File and Storage Technologies, pp. 271-283, 2017.

[3] G. Oh, C. Seo, R. Mayuram, Y. Kee, and S. Lee, "SHARE Interface in Flash Storage for Relational and NoSQL Databases," Proc. of the 2016 ACM SIGMOD International Conference on Management of Data, pp. 343-354, 2016.

[4] Flexible I/O Tester. Available: https://githu- b.com/axboe/fio.

[5] GNU core utils - http://www.gnu.org/software/coreutils.

[6] C. Lee, D. Sim, J. Hwang, and S. Cho, "F2FS: A New File System for Flash Storage", Proc. of the USENIX Conference on File and Storage Technologies, pp. 273-286, 2015.

[7] S. S. Hahn, S. Lee, C. Ji, L. Chang, I. Yee, L. Shi, C. Xue, and J. Kim, "Improving File System Performance of Mobile Storage Systems Using a Decoupled Defragmenter," Proc. of

the 2017 USENIX Annual Technical Conference, pp. 759-771, 2017.

[8] S. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, and Arvind, "BlueDBM: An Appliance for Big Data Analytics," Proc. of the ACM/IEEE Annual International Symposium on Computer Architecture, pp. 1-13, 2015.

[9] J. Park, D. Kang, and Y. Eom. "File Defragmentation Scheme for a Log-Structured File System," Proc. of ACM SIGOPS Asia-Pacific Workshop on System, p. 19, 2016.

[10] 강윤지, 한규화, and 신동군, "로그 구조 파일 시스템의 성능 향상을 위한 논리적 복사 기법," 한국정보과학회 학술발표논문집, pp. 1258-1260, 2015.

Abstract

Linux System Read / Write Performance Optimization Using Remapping Technique

Jeongseok Ryoo
Department of Computer Science and Engineering
Collage of Engineering
The Graduate School
Seoul National University

NAND flash memory can not be overwritten, so if the host performs overwriting, the modified data must be written to the new physical page. As a result, the mapping between the logical block address of the host and the physical page of the flash memory is continuously changed, and the flash translation layer stores and manages this information in the address mapping table. The technique of move/copy data by modifying only the information of the address mapping table without read/write the

physical page using this mapping property is called "remapping".

With remapping, two optimizations are possible with this technique, since the logical block address of the data can be moved arbitrarily.

First, the move/copy speed of the data can be improved. With remapping, only the mapping table which is cached in memory is modified without read/write of the physical page, so the overall move/copy execution time is greatly reduced.

In addition, it is possible to improve read performance by eliminating fragmentation in the system. Fragmentation refers to the case where the data to be read is scattered and stored. By eliminating this fragmentation using remapping, the software stack overhead is reduced and the read performance is improved.

In this paper, we introduce and optimize Linux file utilities and log structure file systems, which have the optimizations as the remapping technique mentioned above. The Linux file utility causes read performance degradation due to fragmentation of files during processing of the files, and the log structure file system causes read/write performance degradation due to fragmentation occurring during data write and periodic segment cleaning.

It is confirmed that the performance of the Linux file utility is

improved by up to 3.4 times and the read/write performance of the log structure file system is improved by 2.1 times and 2.2 times, respectively, by optimizing the two objects through the remapping technique.

Keywords : flash memory, remapping, file system fragmentation, Linux file utility, file system, I/O performance optimization

Student Number : 2016-21219