플래시 메모리 기반 파일 시스템에서 병합동작을 이용한 조각 모음 기법

현철승○ 이동희
서울시립대학교 컴퓨터통계학과
{cshyun○, dhlee}@uos.ac.kr

De-fragmentation Scheme Exploiting Merge Operation in Flash Memory-based File System

Choulseung Hyun○ Donghee Lee
Department of Computer Science, University of Seoul

요 약

플래시 메모리는 무게, 내구성, 전력 소비량 측면에서 기존 디스크보다 우수하기 때문에 주로 휴대용 기기의 저장장치로 사용되었다. 최근에는 집적도가 향상되면서 SSD(Solid State Disk)형태로 노트북에서도 활용되고 있다. 이러한 플래시 메모리는 제자리 갱신이 불가능한 특징 때문에 저장장치로 사용하기 위해서는 FTL(Flash Memory Translation Layer)이라는 주소사상 소프트웨어가 필요하다. 그리고 FTL은 블록을 재활용하기 위해 병합 연산을 수행하게 되는데 이 병합 연산의 비용이 시스템 성능에 큰 영향을 미친다. 아울러 FTL 상에서 동작하는 파일 시스템의 경우도 디스크 기반 파일 시스템과 같이 단편화 문제로 인한 성능 저하가 발생하게 된다. 본 논문에서는 플래시 메모리 기반 파일 시스템에서 단편화 현상을 줄이기 위해 FTL의 병합동작의 특성을 활용한 조각 모음 기법을 제안한다. 실험결과는 제안한 기법이 결국 FTL에서 병합 연산의 비용을 줄임으로써 성능을 향상시킬 수 있음을 보여준다.

1. 서 론[1]

휴대용 기기에서 저장 장치로 많이 사용되고 있는 플래시 메모리는 최근 SSD[1] 형태로 노트북 컴퓨터의 저장장치로도 사용되고 있다. 이러한 플래시 메모리의 가장 큰 특징은 제자리 갱신이 불가능하다는 점과 읽기속도에 비해 쓰기속도가 느리다는 것이다. 이런 특징들로 인해 플래시 메모리를 저장 장치로 사용하기 위한 파일 시스템의 형태는, 첫째로 Log-Structured 파일 시스템 형태를 따르는 JFFS[3]나 YAFFS[4][5]를 사용하는 방법이 있고, 둘째로 섹터 사상을 통해 플래시 메모리를 블록 장치로 인식을 시켜주는 FTL 소프트웨어[6, 7]를 사용하여 기존의 디스크 기반 파일 시스템을 사용하는 방법이 있다.

FTL은 플래시 메모리를 관리하여 상위 레벨의 파일 시스템에게 섹터 단위 읽기/쓰기 연산을 제공한다. 이를 위해 FTL은 논리적인 주소인 섹터번호와 물리적인 플래시 메모리 주소간에 변환을 위한 테이블을 가지고 있어야 한다. 이 변환 테이블은 정밀도에 따라 플래시 메모리의 블록 단위 혹은 페이지 단위로 논리적인 주소와 물리적인 주소를 사상한다. 전자를 블록 수준 사상 기법, 후자를 페이지 수준 사상 기법이라 하는데, 대용량 플래시 메모리 저장 장치의 경우 주로 블록 수준 사상 기법이 사용된다. 그 이유는 페이지 수준 사상 기법을 사용하게 되면 플래시 메모리 크기가 커질수록 사상을 위한 테이블의 크기가 커지게 되므로 실제 데이터가 저장될 공간이 줄어드는 단점이 있기 때문이다.

데이터가 이미 기록된 위치에 덮어쓰기가 불가능하다는 플래시 메모리의 특징 때문에 FTL은 새로운 데이터를 사용되지 않은 빈 블록에 기록하고 사상테이블을 변경 시켜 기존의 데이터를 무효화 시키는 방식을 사용한다. 이런 쓰기 동작이 계속적으로 수행되기 위해서는 추가적으로 빈 블록을 확보해야 하며, FTL은 이를 위해 병합 동작을 수행한다. 병합 동작은 다수의 블록에 존재하는 유효한 데이터를 하나의 빈 블록에 복사하고 기존 블록을 빈 블록으로 지정하여 재활용하는 작업을 의미한다. 이러한 병합 작업의 오버헤드는 FTL의 성능과 파일 시스템 성능에 큰 영향을 미치는 요소이다.

또한, 파일 시스템에서는 저장장치의 사용률이 높아지고 파일 생성, 삭제 연산이 빈번히 일어남에 따라 파일 및 파일 시스템 수준의 단편화가 발생한다[2]. 디스크 기반 파일 시스템에서 파일 수준의 단편화는 파일의 읽기 또는 갱신 연산의 성능에 영향을 미치게 되며, 파일 시스템 수준의 단편화는 새로운 파일이 생성될 때 논리적으로 연속된 블록을 할당하는 것이 어려워지기 때문에 파일 수준의 단편화를 더욱 더 가속화시킨다. 플래시 메모리 기반 파일 시스템에서 단편화는 FTL의 병합동작의 비용을 증가시켜 전체 성능을 급격하게 떨어지게 하는 원인이 된다.

이러한 단편화 문제가 일으키는 성능 저하를 해결하기 위해 파일 시스템 사용자들은 인위적으로 조각 모음을 수행함으로써 단편화를 해결하고 있다. 그렇지만 플래시 메모리 기반 파

[1] 본 연구는 정보통신부 및 정보통신연구진흥원의 IT신성장동력핵심기술개발사업[2006-S-040-01, Flash Memory 기반 임베디드 멀티미디어 소프트웨어 기술 개발]과 한국과학재단 특정기초연구(R01-2004-000-10188-0) 사업의 일환으로 수행하였음.

일 시스템의 경우 병합 연산은 저비용으로 파일 시스템 단편화를 줄이는 기회를 제공할 수 있으며, 본 논문에서는 FTL이 병합 연산을 수행할 때 파일 시스템의 단편화를 줄이는 작업을 같이 수행하도록 하여 성능을 향상시킬 수 있는 온라인 조각 모음 기법을 제안하고 있다.

이하 구성은 다음과 같다. 2장에서는 파일 시스템에서 단편화 문제에 대한 기존의 연구를 살펴보고 3장에서는 왜 플래시 메모리에서 단편화가 문제가 되는지에 대해 설명할 것이다. 4장에서는 본 논문에서 제안한 온라인 조각 모음 기법에 대해 설명한다. 5장에서는 온라인 조각 모음 기법의 구현과 실험결과를 분석하고 6장에서 본 논문의 끝을 맺는다.

2. 관련 연구

디스크 기반 파일 시스템에서의 성능은 파일의 저장된 형태에 의해 큰 영향을 받는다. 그 이유는 디스크의 물리적인 특성 때문이다. 그래서 디스크 기반 파일 시스템의 성능을 향상시키기 위해 파일 데이터를 물리적으로 연속적으로 배치하기 위해 파일 시스템의 배치 정책[8, 9], 할당정책[10], 그리고 조각 모음과 관련된 다수의 연구가 수행되었다. 뿐만 아니라 데이터를 디스크에 중복시킴으로써 탐색시간과 데이터의 연속적인 읽기 시간을 줄일 수 있는 기법에 대해서도 연구되었다[11]. 그렇지만 디스크 기반 파일 시스템의 연구는 주로 읽기 연산을 효율적으로 수행하는데 초점이 맞추어져 있기 때문에 플래시 메모리 기반 파일 시스템에서 적용하기가 어렵다.

플래시 메모리 기반 파일 시스템에서 단편화 문제에 대한 연구는 로그 구조 (Log-structured) 파일 시스템에서 쓰레기 수집(Garbage Collection)을 어떻게 해야 하는가에 대한 형태로 많이 수행되었으며, FTL상에서 단편화에 대한 연구로는 [12]가 있다. 이 논문에서는 작은 파일인 경우 플래시 메모리의 한 블록을 여러 파일이 공유하고, 파일이 정해진 크기 이상으로 커지면 이후 블록 단위로 할당하는 할당정책으로 파일 레벨과 파일 시스템 레벨의 단편화를 효과적으로 방지할 수 있음을 보였다.

본 논문에서는 FTL을 사용하는 플래시 메모리 기반 저장장치에서 파일 시스템의 단편화 방지에 초점을 두고 있다. 특히 파일 수준의 단편화보다는 파일 시스템 수준에서의 단편화가 성능에 더 큰 영향을 주기 때문에 이 부분에 초점을 맞춘다. 조각 모음을 통하여 빈 블록들을 생성하게 되면 파일 시스템에서 데이터를 저장할 때 블록을 효율적으로 할당할 수 있다. 그리고 이렇게 파일 및 파일 시스템 수준의 단편화를 낮은 수준으로 유지하면 파일 시스템의 전체적인 성능을 향상시킬 수 있게 된다.

3. 플래시 메모리 기반 파일 시스템에서 단편화 문제

플래시 메모리 저장 장치를 사용하는 경우 파일 시스템에서 파일을 쓰거나 갱신할 때 파일 시스템의 단편화 정도에 따라 향후 FTL의 병합 연산에 추가적인 오버헤드를 발생시킬 수 있다. 이를 이해하기 위해서는 먼저 플래시 메모리의 물리적인 구조와 데이터 배치를 설명해야 한다. 그림 1은 파일 시스템에서 플래시 메모리의 논리적인 구조를 그린 것이다. 하나의 블록 내에 6개의 페이지가 존재하고 전체 볼륨의 크기는 5개의 블록으로 구성되어 있다고 가정하였다.

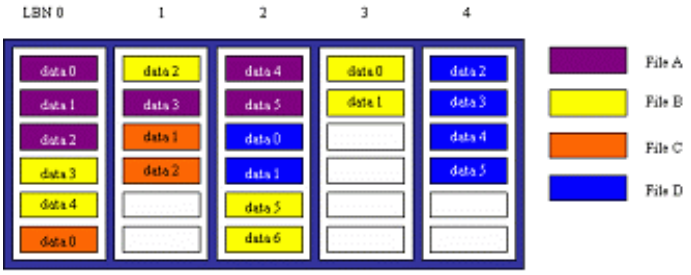

그림 1 플래시 메모리내의 파일 분포

그림 1에서 File A는 세 개의 논리적인 블록에 흩어져 있다. 여기서, File A에 대한 갱신 연산이 발생했을 때 FTL에서 발생하는 병합 연산의 비용은 0, 1, 2번 블록에 있는 파일 데이터를 갱신하기 위해 세 번의 병합 연산이 일어나야 한다. 세 번의 병합 연산은 그리고 3번의 소거 연산과 12번의 카피백 연산을 수행한다. 만약 File A의 모든 데이터가 두 개의 블록에 존재했다면 두 번의 병합 연산과 몇 번의 카피백으로 File A의 갱신 연산 요청을 처리할 수 있을 것이다.

파일 시스템 레벨에서 빈 공간의 단편화 역시 문제가 된다. 새로운 파일이 생성될 때 생성된 파일의 데이터를 쓰기 위하여 빈 블록을 할당해야 한다. 이 때 빈 공간의 단편화가 심해지면 파일 시스템에서 논리적으로 여러 개의 블록을 할당해야 하기 때문에 그만큼의 병합 연산으로 인해 쓰기 성능이 떨어지게 되며 이후 이 파일에 대한 갱신 연산이 발생하게 되면 더 많은 비용이 추가된다. 예를 들어 네 개의 섹터가 필요한 파일이 생성된다고 한다면, 위의 그림에서 1번과 3번, 혹은 3번과 4번 블록내의 페이지가 할당된다. 이 때 FTL에서 발생하는 병합 연산은 두 번의 소거 연산과 몇 개의 카피백 연산이 유발하게 되는데, 만약 빈 공간이 하나로 밀집되어 있다면 연속적으로 할당할 수 있게 되어 한 번의 병합 연산으로 처리할 수 있을 것이다. 따라서 플래시 메모리 기반 파일 시스템에서 조각 모음 동작은 가능하면 빈 블록을 많이 만들어 내는 것으로 정의할 수 있다.

4. 병합 동작을 활용한 조각 모음 기법

그림 1에서 File C가 쓰여질 때 블록 0에 위치한 File C의 “data 0”을 1번 블록의 4번 페이지로 옮기게 되면 File C의 갱신 연산이 발생했을 때 1번의 병합 연산으로 처리할 수 있다. 이 경우 조각 모음의 잠재적 이득은 다음과 같다.

조각 모음 수행 전 갱신 비용: (1E + 5C) + (1E + 2C)

조각 모음 수행 후 갱신 비용: (1E + 2C) + 1C + α

(α는 추가적인 연산비용)

잠재적 이득: 1E + 4C – α

그러나 파일 시스템 입장에서는 File C의 “data 0”이 옮겨진 위치를 빈 공간으로 인식하기 때문에 새로운 파일이 생성되면

해당 위치를 할당해 주게 된다. 이 시점에서 플래시 메모리는 제자리 갱신이 불가능하기 때문에 새로운 로그블록을 할당하여야 하고 결과적으로 병합 연산이 일어나게 되어 그 비용은 조각 모음으로 얻을 수 있는 기대 이익보다 커지게 된다. 즉, 파일 레벨의 조각 모음으로 인한 이득은 미미하다고 볼 수 있다.

그림 1에서 블록 1번에 파일 데이터가 쓰여지고 난 다음, 다른 블록에 데이터가 쓰여질 때 로그블록을 확보하기 위해 블록 1번이 병합된다고 한다면, 블록 1번의 데이터 블록과 로그블록의 유효한 데이터만 새로운 빈 블록에 복사된다. 이 시점에서 블록 3번에 있는 두 개의 데이터를 1번 블록으로 옮기면 블록 3번은 빈 블록으로 남게 된다. 이와 같이 조각 모음을 수행했을 때의 기대 이익은 다음과 같다.

조각 모음 이전 비용: (1E + 2W) + (1E + 4W)

조각 모음 이후 비용: (1E + 6W) + 2C

이러한 조각 모음은 파일 데이터를 가능하면 하나의 블록에서 할당할 수 있게 해주며, 결국 파일 시스템 및 파일 레벨에서의 단편화를 낮추어 FTL의 병합 연산 비용을 줄이고 궁극적으로 성능 이득을 얻을 수 있다.

5. 구현 및 실험 결과

5장에서 설명한 조각 모음 기법을 FAT 호환 파일 시스템과 FTL에 구현하였다. 조각 모음 기법을 구현하기 위해 FTL에 다음과 같은 새로운 API를 추가하였다.

```
ftl_defrag_copy(int flash_dev,
                int sec_num,
                int src_sector,
                int tgt_sector)
```

여기서, flash_dev는 플래시 메모리 장치 번호이며, sec_num는 이동될 섹터의 개수를 나타내고, src_sector는 이동될 섹터의 첫 번째 번호를 나타내며, tgt_sector는 이동될 곳의 섹터 번호이다.

FTL의 동작 방식을 고려하여 파일 시스템이 조각 모음 실행 시간을 결정해야 한다. 그렇지 않고 파일 시스템에서 계속 쓰기 요청이 이루어지는데 조각 모음을 수행한다면 추가적인 병합 연산이 발생할 수 있기 때문에 손해가 될 수도 있다. 따라서 파일이 안정적일 때 즉, 파일이 닫힐 때 조각 모음을 수행하도록 하였다. 조각 모음을 수행하면 파일 시스템은 파일 데이터가 이동하였으므로 FAT을 갱신하여 파일 데이터의 이동을 반영해야 한다. 그런데, 이 시점에서 파일 데이터는 이전 위치와 새로운 위치 모두에 존재하기 때문에 비교적 안전하게 FAT을 갱신할 수 있다.

조각 모음을 위하여 한 블록 내에 어느 정도의 미사용 영역이 존재하는지를 나타내는 Threshold를 정의한다. Threshold는 1/2, 1/4, 1/8, 1/16중 하나로 정의된다. 페이지 크기가 2K 바이트이고 블록 내 페이지 수가 64개인 대블록 플래시인 경우 Threshold 값들은 각각 64K, 32K, 16K, 8K 크기에 해당한다. 그리고 어떤 파일이 기록된 후 이 파일이 마지막으로 기록된 블록 내에 Threshold 만큼의 미사용 클러스터가 존재한다면, 해당 블록 내에 데이터가 존재하는 파일들 중 다른 블록에 저장되어 조각난 클러스터들을 마지막으로 기록된 블록으로 복사하여 조각 모음을 수행한다.

FAT 파일 시스템에서 클러스터 할당정책에 따라 조각 모음의 성능이 달라질 수 있다. 따라서 클러스터 할당 정책이 미치는 영향을 알아보기 위해 일반적인 클러스터 할당정책(SCA)과 [12]에서 제안한 AFCA 할당정책을 사용하여 실험하였다. SCA 정책은 마지막 할당된 위치부터 순차적으로 FAT 테이블을 검사하면서 빈 클러스터를 할당하는 정책이다. 반면에 AFCA 정책은 파일이 미리 정해진 크기보다 커지게 되면 큰 파일로 가정하여 하나의 플래시 블록을 할당하고 작은 파일인 경우 이들이 하나의 블록을 공유하도록 할당하는 정책이다.

조각 모음은 오랜 시간 동안 파일 시스템을 동작시켜 파일 시스템에 저장된 파일의 형태를 관찰하여야 한다. 그러나 이것은 현실적으로 불가능하고 조각 모음의 성능 측정을 위한 벤치마크가 없기 때문에 인위적으로 파일 시스템을 단편화시키고 난 후 Postmark 벤치마크를 이용하여 얼마나 빨리 회복하는지에 대한 실험을 통해 제안한 기법을 평가하였다.

파일 시스템을 단편화시키기 위해 1K 크기의 파일을 생성하여 파일 시스템 전체를 채우고 난 다음 홀수번째 파일을 삭제하였다. 그리고 표 1과 같이 Postmark 벤치마크의 파라미터를 설정한 후 Postmark 벤치마크를 수행하였다.

파라미터	값
초기 파일 생성 수	500
트랜잭션 수	1000
파일 크기	0.5K – 10K
파일 생성/삭제 비율	5
파일 읽기/쓰기 비율	5

표 1 Postmark 벤치마크의 파라미터

표 2와 표 3은 각각 SCA 정책과 AFCA 정책을 사용하였을 때 FTL이 수행한 page read, page write, page copyback, block erase 연산의 횟수다. SCA 정책에서는 Threshold가 1/4일 때 조각 모음을 수행하지 않았을 때보다 좋은 성능을 보였다. 연산의 횟수로 보면, page read/write 연산의 횟수는 증가하였지만, page copyback과 block erase 연산은 줄어들었음을 확인할 수 있다.

threshold	page write	page read	page copyback	block erase
없음	236218	478907	215846	5480
1/16	236218	478973	215846	5480
1/8	236218	478973	215846	5480
1/4	247740	489677	198945	5434
1/2	250858	494412	212956	5563

표 2 SCA 정책에서 조각 모음 수행 결과

threshold	page write	page read	page copyback	block erase
없음	208908	446946	130318	3718
1/16	202773	440864	133853	3698

1/8	202773	440864	133853	3698
1/4	202773	440864	133853	3698
1/2	201504	440851	135121	3698

표 3 AFCA 정책에서 조각 모음 수행 결과

AFCA 정책에서도 조각 모음은 조각 모음을 수행하지 않은 경우보다 성능을 향상시키지만, SCA정책과는 달리 page read/write 연산의 횟수는 줄어들고 page copyback 연산은 증가하였음을 볼 수 있다. 또한 Threshold가 1/4일 때 가장 좋은 성능을 보인다. 이는 너무 적극적으로 조각 모음을 수행하거나 너무 소극적으로 수행하면 오히려 조각 모음으로 인해 손해를 볼 수 있음을 보여준다.

두 실험을 통하여 page read/write와 page copyback 연산의 증감은 서로 상쇄하여 실행 시간 상으로 봤을 때 차이를 나타내지 않으며, 조각 모음을 통하여 약간 block erase 연산이 감소함을 확인할 수 있다. 그리고 또한 소거 연산의 횟수가 제한되어 있는 플래시 메모리의 수명을 증가시킬수 있음을 보여준다.

6. 결론

파일 시스템은 시간이 지남에 따라 단편화가 발생하여 파일 시스템의 성능을 저하시킨다. 이러한 현상은 디스크 기반 파일 시스템뿐만 아니라 플래시 메모리 기반 파일 시스템에서도 동일하게 관찰된다. 그렇지만 저장 장치의 물리적 특성차이로 인해 플래시 메모리 기반 파일 시스템에서의 단편화는 디스크 기반 파일 시스템과 그 정의와 해결에 대한 접근 방법이 달라져야 한다. 본 논문에서는 플래시 메모리 저장 장치에서 FTL의 병합 동작을 활용하여 저렴하게 온라인으로 파일 시스템의 조각 모음을 수행하는 기법을 제안하고 있다. 그리고 Postmark 벤치마크 실험을 통하여 저비용 온라인 조각 모음이 파일 시스템의 단편화를 줄이고 파일 시스템 전체의 성능을 향상시킬 수 있음을 보였다.

7. 참고 문헌

[1] Sanglyul Min and Eyeehyun Nam, “Current Trends in Flash Memory Technology”, In Proceedings of the 2006 conference on Asia South Pacific design automation, p. 332-333, 2006.

[2] Keith A. Smith and Margo I. Seltzer, “File System Aging - Increasing the Relevance of File System Benchmarks”, In Proceedings of the 1997 ACM SIGMETRICS Conference, pp. 203-213, June 1997.

[3] D. Woodhouse, “JFFS: The Journaling Flash File System”, Ottawa Linux Symposium 2001, 2001.

[4] YAFFS (Yet Another Flash File System) Specification Version 0.3, http://www.aleph1.co.kr/yaffs, 2002.

[5] A. Kawaguchi, S. Nishioka, and H. Motoda, “A Flash Memory Based File System”, In Proceedings of '95 Winter USENIX Technical Conference, 1995, pp. 155-164.

[6] Understanding the Flash Translation Layer (FTL) Specification, Intel Corporation, 1998.

[7] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min and Yookun Cho, “A Space-efficient Flash Translation Layer For CompactFlash Systems”, IEEE Transactions on Consumer Electronics, May 2002.

[8] Marshall K. McKusick, William N. Joy, Samuel J. Leffler and Robert S. Fabry, “A Fast File System for UNIX”, ACM Transactions on Computer Systems, Vol. 2, No. 3, pp. 181-197, 1984.

[9] Keith A. Smith and Margo I. Seltzer, “File Layout and File System Performance”, Harvard University Computer Science Department Technical Report TR-34-94, 1994.

[10] Keith A. Smith and Margo I. Seltzer “A Comparison of FFS Disk Allocation Policies”, 1996 USENIX Annual Technical Conference, pp. 15 – 26, 1996.

[11] Windsor W. Hsu, Alan Jay Smith and Honesty C. Young, “The Automatic Improvement of Locality in Storage Systems”, ACM Transactions on Computer Systems, Vol. 23, No. 4, November 2005, pp 424 – 473, 2005.

[12] Sungkwan Kim, Donghee Lee and Sanglyul Min , “An Efficient Cluster Allocation Scheme for NAND Flash Memory Based FAT File Systems”, In Proceedings of the 1st International Workshop on Software Support for Portable Storage, 2005.