JFFS2를 위한 효율적인 Garbage Collector의 설계 및 구현

정하용° 김진수 한환수 최기선
한국과학기술원 전자전산학과 전산학전공
hymanse@kaist.ac.kr° {jinsoo, hshan, kschoi}@cs.kaist.ac.kr

Design & Implementation of an Efficient Garbage Collector for JFFS2

Ha-Yong Jung° Jin-Soo Kim, Hwansoo Han, Key-Sun Choi
Dept. of Computer Science, KAIST

요 약

플래시 메모리는 저전력 소비, 빠른 읽기속도, 비휘발성 등 좋은 특징을 많이 가지고 있다. 하지만 반대로 플래시 메모리는 치명적인 약점도 가지고 있는데, 그것은 덮어쓰기가 불가능 하다는 것 과 삭제속도가 극단적으로 느리다는 것이다. 따라서 이와 같은 치명적인 약점을 극복하기 위한 효율적인 파일시스템과 Garbage Collector(GC)의 설계는 플래시 메모리 연구의 핵심적인 부분이 되어 왔고 JFFS2(Journaling Flash File System version 2)는 그러한 연구의 결과 중 하나이다. 본 논문은 기존에 JFFS2에서 사용된 GC와 비교해 좀 더 효율적인 GC를 제안한다. 성능향상을 위해서 사용된 두 가지 핵심적인 알고리즘은 Cost Age Times (CAT) 방법과 Dynamic dAta Clustering (DAC) 방법이며, 결과적으로 지역성(Locality)이 높은 데이터의 쓰기에서 최고 3배 정도의 성능향상을 보였다.

1. 서 론

최근 플래시 메모리는 다양한 분야에서 널리 쓰이고 있다. 이와 같이 플래시 메모리가 널리 쓰이는 이유는 그것의 작고 가벼운 외형과 저전력 소비, 빠른 읽기 속도와 비휘발성 등이 최근의 임베디드 시스템을 비롯한 각종 이동 컴퓨팅기기에 대단히 적합하기 때문이다. 하지만 이런 다양한 장점에도 불구하고 플래시 메모리는 치명적인 약점을 가지고 있는데 그것은 느린 쓰기속도와 극단적으로 느린 삭제속도이다. 뿐만 아니라 플래시 메모리는 덮어 쓰는 것도 불가능하다. 따라서 덮어쓰기나 수정, 갱신 등의 작업을 위해서는 언제나 기존의 데이터를 먼저 삭제한 뒤 다시 써야 한다. 따라서 적절한 시점에서의 데이터 삭제를 관리하는 Garbage Collector(GC)는 플래시 메모리 파일시스템의 설계에 있어서 대단히 중요하다.

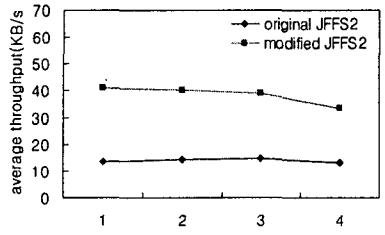
이와 같은 문제를 해결하기 위해서 기존의 대부분의 접근방법들은 파일시스템의 논리적 주소와 플래시 메모리의 물리적 주소를 따로 유지하면서 그사이의 효과적인 변환 방법(Flash Translation Layer, FTL)을 설계하려 했다. 논리적 주소와 물리적 주소를 따로 유지함으로써 같은 논리적 주소에 갱신이 일어난다 해도, 다른 물리적 주소에 그 내용을 갱신하고 두 주소를 연결해 주는 방식으로 문제를 해결하려 한 것이다.

한편, 이와는 전혀 다른 방식의 접근도 시도되었는데, 그것이 바로 JFFS(Journaling Flash File System)이다.[1] JFFS 이전의 방법들이 FTL을 설계하는데 치중한 것과 달리 JFFS는 그 자체가 바로 하나의 완전한 파일시스템으로서 다른 파일시스템이나 변환방법 등을 전혀 필요로 하지 않는다. JFFS의 기본적인 아이디어는 간단하다. JFFS는 순수한 순차 기록 구조 파일시스템으로서[5,6] 데이터와 메타데이터를 포함하는 노드들이 플래시 메모리 위에 순차적으로 저장되게 된다. JFFS는 문제점 해결과 더불어 성능향상을 거쳐 JFFS2에 이르렀지만 JFFS2의 GC는 여전히 너무 단순한 구조를 가지고 있다. JFFS2는 오직 유효한 데이터만 들어 있는 블록들은 Clean list에 모아두고, 하나라도 무효인 데이터가 들어 있는 블록들은 Dirty list에 모아 둔 뒤, GC의 대상 블록을 언제나 Dirty list에서만 선택하는 방식을 취한다.

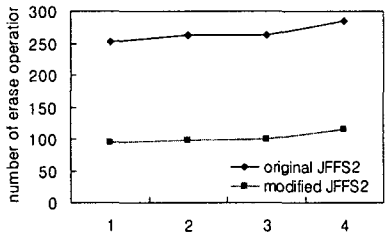
2. 접 근 방 법

전술한 바와 같이 단순한 구조인 JFFS2의 GC를 좀 더 효율적으로 만들기 위해 주목한 특성은 데이터의 지역성이다. 지역성은 데이터 접근 패턴에서 대단히 일반적으로 나타나는 패턴으로서 읽기패턴 에서는 물론 쓰기패턴에서도 드러난다. 즉, 자주 써지는 데이터만 늘 자주 써진다는 것이다. 따라서 저장매체에 쓰여지는 데이터들은 쓰기 빈도에 따라 다음과 같이 분류될 수 있다. 우선 고빈도(Hot) 데이터는 갱신이 자주 일어나는 데이터로서, 그 예로 실제 데이터를 위한 메타데이터가 있다. 메타데이터는 실제 데이터중의 일부에만 변화가 생겨도 이를 반영하기 위해 대단히 빈번히 갱신된다. 반면에 갱신이 거의 일어나지 않는 데이터는 저빈도(Cold) 데이터로 분류할 수 있다. 예를 들어, 일반적인 컴퓨터 프로그램의 코드 영역이나 정적 데이터 영역은 새로운 프로그램으로 덮어 씌워지지 않는 한 결코 갱신되는 일이 없다. 이와 같은 쓰기 패턴은 대단히 일반적이지만, 디스크를 물리적 매체로 사용하는 일반적인 파일시스템에서는 그다지 중요한 의미를 가지지 못한다. 하지만 물리적인 매체가 플래시 메모리일 경우 이것은 대단히 중요하다. 왜냐하면 플래시 메모리는 덮어씌우는 갱신이 불가능하기 때문이다. 만약 디스크와 마찬가지로 갱신을 하고자 한다면, 원래의 데이터를 먼저 지운 뒤 새로운 데이터를 써야 하는데, 플래시 메모리의 최소 삭제 단위 블록은 최소 쓰기 단위 블록보다 훨씬 크다. 따라서 훨씬 큰 영역을 삭제해야 하므로 엄청난 부담 비용을 유발한다.

특히 저빈도 데이터와 고빈도 데이터가 섞여 있는 블록이 GC의 대상으로 선택 되었을 경우, 대부분의 고빈도 데이터들은 이미 무효화되어 있을 것이다. 왜냐하면 그것들은 빈번히 갱신되기 때문이다. 반면에 대부분의 저빈도 데이터들은 여전히 유효한 상태일 것이다. 마찬가지로 그것들은 거의 갱신되지 않기 때문이다.

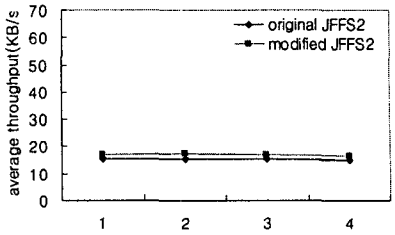


(a) 평균 쓰기 속도

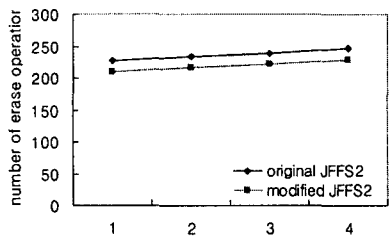


(b) 평균 블록 삭제 횟수

그림 1. 고빈도-저빈도 쓰기 성능



(a) 평균 쓰기 속도



(b) 평균 블록 삭제 횟수

그림 2. 무작위 쓰기 성능

이런 상황에서 GC는 모든 유효 블록을 다른 새로운 블록으로 복사할 수 밖에 없다. 이것은 순전히 부담 비용일 뿐이다. 사실 저빈도 데이터들을 복사하는 것은 전혀 필요치 않은 동작이기 때문이다. 뿐만 아니라, 저빈도 데이터가 복사된 새로운 블록에는 또다시 고빈도 데이터들이 갱신될 것이고, 금새 무효화될 것이며, 쓰기 작업이 진행되면서 이런 상황은 계속해서 반복 발생하게 될 것이다. 다시 말해, 고빈도 데이터와 섞여있는 한 저빈도 데이터는 계속 의미 없이 복사되어야 하는 것이다.

이와 같은 문제를 해결하기 위해 저빈도 데이터를 위한 블록과 고빈도 데이터를 위한 블록을 분리시키는 방법이 제안되었다[3]. 만약 저빈도 데이터가 있는 블록에는 저빈도 데이터만이 모여있고, 고빈도 데이터가 있는 블록에는 고빈도 데이터만 모여있다면 비용을 고려하는 GC는 거의 언제나 대상 블록을 고빈도 데이터가 모여있는 블록들 중에서 선택하게 될 것이다. 특히 선택된 대상 블록은 고빈도 데이터들만 모여 있고 고빈도 데이터들은 이미 무효화되어 있을 것이므로, 유효한 데이터들을 복사하기 위한 부담 비용은 대단히 낮아지게 된다.

따라서 중요한 것은 어떻게 데이터들을 쓰기빈도에 따라 나눌 것인가 하는 문제와 어떻게 적절한 대상 블록을 선택할 것인가 하는 문제인데, [2,7,8]에서는 이것들을 위해 각각 DAC(Dynamic dAta Clustering) 알고리즘과 CAT(Cost Age Times) 알고리즘을 제안하였고, 본 논문에서는 그 알고리즘들을 FTL이 아닌 JFFS2에 적용시켜 구현했다.

DAC 알고리즘은 데이터를 쓰기빈도에 따라 나누는 방법으로서 플래시 메모리를 개념적으로 몇 개의 영역(region)으로 나눈다. 최초의 쓰기는 모두 바닥영역에서 이루어지며 이후에 임의의 데이터에 갱신이 이루어질 경우 그 데이터가 보존된 시간이 임계 값보다 작으면 한 단계 높은 영역에 갱신할 데이터를 쓴다. 마찬가지로, 임의의 시간에 GC가 이루어질 경우 대상블록에 포함된 데이터가 보존된 시간이 임계 값보다 크면 한 단계 낮은 영역에, 작으면 한 단계 높은 영역에 데이터를 쓴다. 이런 방식을 취할 경우 높은 영역으로 갈수록 좀 더 고빈도의 데이터가 존재하고, 낮은 영역으로 갈수록 좀 더 저빈도의 데이터가 존재하게 된다. 한 편 CAT 알고리즘은 GC의 대상블록을 선택하는 방법으로서, 각 블록의 CAT값을 구해서 CAT값이 가장 작은 블록을 GC 대상블록으로 선택한다. 이 때, CAT값은 블록 안의 유효데이터/무효데이터로 구해지는 블록 삭제비용과 그 블록이 생성된 이후 현재까지 지난 시간의 역수, 그리고 그 블록의 삭제횟수를 모두 곱해 구한다.

3. 성 능 평 가

3.1. 고빈도-저빈도(Hot-and-Cold) 쓰기 성능

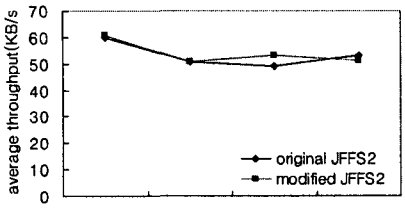
본 논문의 목적은 지역성(Locality)이 강한 데이터의 쓰기성능을 향상시키는 것이다. 따라서 고빈도-저빈도 쓰기성능은 가장 뚜렷한 성능향상을 관찰할 수 있을 것으로 기대했고, 결과는 예상에서 빗나가지 않았다.

이 성능평가를 위해 사용된 테스트는 640-116테스트 이다. 640-116은 60%의 쓰기는 초기 데이터의 1/8 영역에서 일어나고, 나머지 40%의 쓰기는 초기 데이터의 또 다른 1/8 영역에서 일어나며, 초기 데이터의 3/4 영역에는 쓰기가 이루어지지 않음을 의미한다. 이러한 쓰기비율은 실제의 파일시스템에서 67~78%의 쓰기가 메타데이터 영역에서 이루어진다는 것을 밝힌 [4]의 결과에 기초한 것이다.

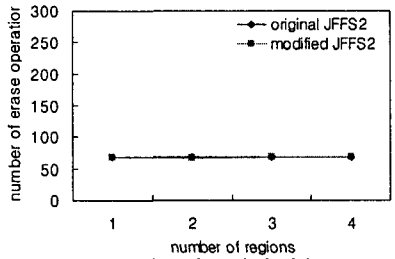
그림1은 결과를 보여준다. 우선 CAT의 적용은 뚜렷한 성능향상을 가져왔다. 평균 쓰기속도는 최고 3배 정도 빨라졌고, 평균 블록 삭제횟수 역시 반 이하로 줄어들었다. 하지만 DAC의 적용은 아무런 성능향상을 가져오지 못했을 뿐만 아니라 데이터를 빈도에 따라 몇 단계로 나눌 것인지를 의미하는 영역수가 증가할 경우 오히려 성능하락을 보였다.

3.2. 무작위 쓰기 성능

사실 고빈도-저빈도 쓰기 성능 평가를 제외한 성능 평가들은 새로운 GC 알고리즘 적용에 따르는 성능 저하가 없는지를 관찰하기 위함이다. 비록 고빈도-저빈도 쓰기 성능의 향상이 뚜렷하다고 해도 다른 쓰기 성능의 저하가 크다면 새로운 알고리즘을 JFFS2에 적용하지 않는 것이 좋을 것이다.



(a) 평균 쓰기 속도



(b) 평균 블록 삭제 횟수

그림 3. 순차 쓰기 성능

무작위 쓰기 성능 평가는 아무런 성능향상이나 성능저하가 없을 것으로 예상했다. 왜냐하면 쓰기 패턴에 지역성이 없는 경우 데이터 쓰기빈도 별로 분류하는 것이나 CAT에 의한 대상 블록 선택은 아무 의미 없는 것이기 때문이다. 앞선 실험과 마찬가지로 실험결과는 예상에서 벗어나지 않았다.

그림 2 는 성능저하가 없을 뿐 아니라 큰 의미는 없지만 오히려 약간의 성능향상이 존재한다는 실험결과를 보여준다. 성능향상은 평균 쓰기 속도를 나타내는 그림 2(a)에는 거의 드러나지 않지만, 평균 삭제 횟수를 나타내는 그림 2(b)에는 약간 드러난다. 이것은 무작위 쓰기에 의해 무효 데이터가 사방에 흩어져 있지만 그나마 그 중에서 비용이 가장 작은 것을 선택한 CAT 에 의한 것으로 생각된다. 그리고 DAC 는 여전히 성능에 아무런 영향을 끼치지 못하고 있는 것 역시 관찰할 수 있다.

3.3. 순차 쓰기 성능

순차 쓰기 성능 평가는 최악의 경우를 평가하기 위함이다. 많은 양의 데이터가 순차적으로 쓰여졌을 경우, 각 블록들은 순차적으로 무효화된다. 즉, 이런 상황에서는 GC을 위한 최고의 대상 블록이 순차적으로 나타나는 것이다. 따라서 대상 블록을 순차적으로 선택하는 JFFS2의 기존 GC 알고리즘만으로도 충분할 뿐 아니라, 가장 효율적이다. 이런 상황하에서 CAT 값을 계산하기 위한 비용은 완전히 부담일 뿐이다. 따라서 순차쓰기 성능평가는 분명히 성능저하가 있을 것으로 예상했다. 하지만 실험결과는 부담 비용이 무시해도 될만한 정도임을 보여줬다.

그림3은 평가 결과를 보여준다. 비록 평균 쓰기 속도에 약간의 차이가 보이긴 하지만, 그것은 쓰기속도에 영향을 끼치는 다른 요인에 의한 것으로 보인다. 왜냐하면 평균 삭제횟수를 나타내는 그림3(b)는 DAC와 CAT으로부터 직접적인 영향을 받는 부분인데, 그것이 서로 거의 일치하기 때문이다. 그밖에 관찰되는 특이한 점은 평균쓰기 속도가 영역수가 1일 때 뚜렷하게 좋다는 점인데, 이것은 DAC가 특히 순차 쓰기일 경우에는 단지 시간 부담일 뿐이라는 것을 의미한다.

4. 결 론

본 논문에서는 CAT 과 DAC 를 이용한 좀 더 효율적인 G.C 알고리즘이 JFFS 에 적용되어 구현되었고, 앞서 살펴본 바와 같이 성능평가가 이루어졌다.

실험 결과를 살펴 보면, 우선 CAT 알고리즘은 고빈도-저빈도 쓰기에서 뚜렷한 성능향상을 가져왔고, 성능향상 폭은 이전과 비교해 3 배 가까이에 이르렀다. 이것은 단지 CAT 알고리즘이 대단히 효율적이기 때문이라기 보다는 이전의 G.C 알고리즘이 고빈도-저빈도 쓰기에 있어서 대단히 취약했기 때문이다. 즉, 기존의 JFFS2 에 있어서 고빈도-저빈도 쓰기는 단지 무작위 쓰기의 극한적인 상황일 뿐인 것이다.

하지만 DAC 알고리즘은 모든 경우에 있어서 아무런 성능향상을 가져오지 못했다. 오히려 쓰기 속도는 몇몇 경우에 있어서 떨어지기도 했다. 이와 같은 결과의 가장 중요한 원인은 JFFS2 의 기존 G.C 알고리즘이 데이터를 분류해서 모으는 방법을 이미 사용하고 있기 때문인 것으로 생각된다. 그것은 바로 Clean list 이다. Clean list 는 유효 데이터만을 가지고 있고, 계속 Clean list 에 남아 있기 위해서는 갱신이 이루어지지 않아야만 한다. 즉, Clean list 는 갱신이 전혀 이루어 지지 않은 가장 낮은 빈도의 데이터만을 가지고 있기 때문에 Clean list 자체가 DAC 의 바닥영역과 마찬가지가 되어 결국 DAC 의 적용은 Dirty list 를 좀더 자세히 나누는 역할밖에 하지 못한 것이다.

결론적으로 CAT 는 지역성이 강한 데이터의 쓰기에서 큰 폭의 성능 향상을 가져올 뿐 아니라 어떤 방식의 쓰기에서도 거의 성능하락이 없기 때문에 당장 JFFS2 의 G.C 에 적용해도 효과적일 것이지만 DAC 를 효과적으로 적용하기 위해서는 추가적인 연구가 필요할 것으로 보인다.

향후 연구에서는 본 연구를 바탕으로 DAC 가 성능향상을 가져오지 못하는 정확한 이유 파악을 위한 데이터의 영역별 이동 추적 과 더불어 JFFS2 와 FTL 간의 성능비교를 수행할 수 있을 것이다. JFFS2 와 FTL 은 플래시 메모리의 파일시스템을 관리하는 방법이 많이 다르기 때문에 특정 방법이 우수하다기 보다는 상황에 따라 우수한 방법이 있을 것으로 예상되며, 두 방법의 비교는 좀 더 나은 파일시스템과 G.C 의 개발에 도움을 줄 수 있을 것으로 기대한다.

5. 참고문헌

[1] David Woodhouse, "JFFS: The Journalling Flash File System", *Ottawa Linux Symposium*, 2001.

[2] M. L. Chiang, Paul C. H. Lee, and R. C. Chang, "Data Management in a Flash Memory Based Storage Server", *International Computer Symposium (ICS'98)*, 1998.

[3] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda, "A Flash-Memory Based File System", *Usenix Technical Conference*, 1995.

[4] C. Ruemmler and J. Wilkes, "UNIX disk access patterns", *Proc. '93 Winter USENIX*, 1993.

[5] 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, 1992.

[6] M. Seltzer, K. Bostic, M. K. McKusick, and C. Staelin, "An Implementation of a Log-Structured File System for UNIX", *Proc. '93 Winter USENIX*, 1993.

[7] M. L. Chiang, Paul C. H. Lee, and R. C. Chang, "Managing Flash Memory in Personal Communication Devices", *Proceedings of the 1997 International Symposium on Consumer Electronics (ISCE'97)*, Singapore, 1997, pp. 177-182.

[8] M. L. Chiang and R. C. Chang, "Cleaning Policies in Mobile Computers Using Flash Memory", *Journal of Systems and Software*.