플래시 메모리 파일 시스템의 지움정책 개선에 관한 연구

전승진[0] 공기석 황달연
한국 산업기술 대학교 에너지 대학원
blue_m45[0]@hotmail.com, {kskong, dyhwang}@kpu.ac.kr

An Enhanced Cleaning Policy Of Flash Memory File System

SeungJin Jeon[0], Ki-Sok Kong, DalYeon Hwnag
Dept. of Graduate School of Energy, Korea Polytechnic University.

요 약

최근 들어 임베디드 시스템이 많은 발전을 하고 있고, 공간적인 장점으로 인해서 임베디드 시스템에 플래시 메모리가 많이 사용되고 있다. 그러나 플래시 메모리는 일반 자기 저장 매체와는 다른 특성을 가지고 있어서 플래시 메모리에 알맞은 효율적인 파일 관리 시스템을 필요로 한다. 현재 임베디드 리눅스에서는 JFFS2파일 시스템을 많이 사용하고 있는데, 이 논문에서는 JFFS2의 wear-leveling 알고리즘을 개선한 빈도수를 이용한 지움정책을 제안하고, 시뮬레이션을 통해서 빈도수를 이용한 지움정책의 wear-leveling이 효과적인 성능을 나타냄을 보였다.

1. 서 론

플래시 메모리는 전원이 꺼져져도 데이터가 손실되지 않는 비휘발성 특성을 가지며, 전기적으로 데이터를 지우고, 쓰고, 읽을 수 있는 저장 매체로서 최근 급속히 발전하고 있는 임베디드 시스템에서 많이 사용하고 있다. 플래시 메모리는 다른 저장 매체에 비해 휴대가 용이하고, 전력소모가 적으며, 임의 접근 속도가 빠르고 실용 용량에서의 비트단가가 낮다는 장점을 가지고 디지털 카메라, MP3 player, 가전기기, 휴대용 통신 기기, 셋탑박스 등에서 그 사용이 급격하게 증가 하고 있다. 플래시 메모리는 기존의 자기 저장 매체와는 달리 데이터를 쓰기 위해서는 반드시 erase작업을 거쳐야 하며 반복된 기록 작업으로 한 셀만 수명이 다해도 나머지 셀들을 사용할 수 없는 특성을 가진다. 때문에 플래시 메모리를 많이 사용하는 리얼타임 시스템과 임베디드 시스템에서는 플래시 메모리에 적당한 효율적인 파일 시스템을 필요로 한다. 플래시 메모리 파일 시스템은 지움정책, 오류복구, 등의 쟁점이 있다. 하지만 JFFS2는 데이터를 메모리 공간에 순차적으로 저장하고 erase block을 연속적인 리스트로 관리하는 LFS방법을 기본으로 사용하고 있기 때문에 오류복구가 용이하고 wear-leveling 구현을 따로 할 필요가 없다.[4] 그러나 wear-leveling을 통해서 더욱 효과적으로 플래시 메모리를 관리할 필요가 있다.

이 논문에서는 JFFS2의 garbage collection이 어떻게 동작하는가를 알아보고 효과적인 wear- leveling을 위한 시점과 방법을 제안 하며, 시뮬레이션을 통하여 기존의 방법에 비하여 얼마나 향상되었는지를 알아볼 것이다.

2. 관련연구

2.1 플래시 메모리의 지움정책

플래시 메모리는 한번 사용한 블록을 다시 사용하기 위해서 항상 erase과정을 거쳐야 한다. 플래시 메모리는 초기에는 논리1로 세팅 되어 있다가, data가 쓰여지면 논리 0 값으로 바뀌게 되는데 한번 사용된 블록을 재 사용하기 위해서는 논리 값을 다시 1로 세팅하는 과정을 거쳐야 한다. 이것이 erasing과정이다. 읽기, 쓰기 속도에 비해 지우는 속도가 상당히 느리며, 또한 상온에서의 대략 10만번의 사용 제한이 있기 때문에 특정 블록만 계속 사용하여 10만번 이상을 사용하면 나머지 다른 부분도 사용 할 수 없게 된다. 따라서 이러한 사건이 발생하지 않도록 하기 위해서 전체 블록에 고르게 data를 기록하는 관리가 필요하고 이것을 wear-leveling이라 한다.

2.2 기존 플래시 메모리 파일 시스템의 지움정책

지움정책을 설계할 때는 언제 지우는 작업을 해야 하는가? 한번에 얼만큼의 블록을 지우는가? 어떤 블럭을 지울 것인가? 지울 블록에 있는 유효한 데이터를 어떻게 처리할 것인가? 하는 4가지를 고려 하여야 한다. 첫 번째로 LFS[1]에서는 지움정책을 비교하기 위해 Write Cost라고 하는 계산값을 사용하는데, 지우기 위한 Over Head까지 포함해서 데이터를 쓰는데 소요되는 시간의 양이다. 그러나 원시적인 파일 시스템의 형태로서 현재 JFFS2에서 지원하는 다양한 기능에 비해 성능이 많이 떨어진다. 그리고 write Cost를 계산하기 위해서 많은 계산 과정이

들어가야 한다.

eNVy[2] 시스템에서는 블록에 있는 유효한 데이터를 다른 블록에 옮기는 3가지의 지움정책에 대해서 제시한다. Greedy는 LFS와 유사한 방법으로 유효한 방법으로 유효한 data가 가장 적은 블록으로 옮기는 방법이고, 두 번째는 높은 locality가 있는 경우에 인접한 블록으로 데이터를 옮기는 방법이고, 세 번째로 앞선 두 가지의 절충형이 있다. 그러나 느린 쓰기를 개선하기 위해 버퍼를 따로 사용해야 하고 매 클리닝 마다 비용과 빈도를 곱하여 전체 세그먼트와 비교해야 하는 계산과정이 많이 들어간다.

가와 구치의 방법에서는[3] Geedy와 Cost-Benefit 두 가지 정책을 사용하고 있다. Greedy 유용한 데이터가 가장 적은 블록을 지울 대상으로 선택하는 방법이고 Cost-benefit은 투자 비용에 의해 이익이 가장 많이 발생하는 블록을 지울 대상 블록으로 선정하는 방법이다. 마찬가지로 투자비용대비 이익을 계산하기 위해서 여러 번의 곱셈 과정이 들어가야 한다.

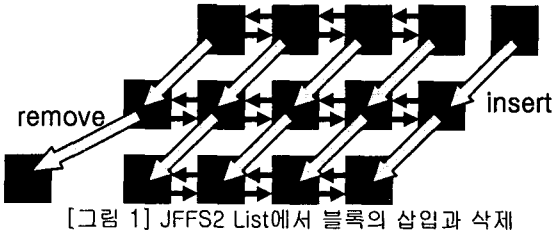
등급별 지움 정책은[4] erase block 내에서 erase block의 크기에 대한 각각의 Valid, Free, Invalid 블록 비율과 erase block의 전체 지움 가능 횟수에 대한 현재까지의 지움 횟수의 비율을 이용하여 계산된 등급 값을 가지고 지울 블록을 선택하는 정책이다. 이 정책도 등급 값을 정하기 위해 블록마다 몇 번의 곱셈과 나눗셈이 들어간다.

JFFS2는 현재 임베디드 리눅스에서 가장 많이 쓰이는 파일 시스템이다. 플래시의 공간을 순차적으로 보고 데이터를 순차적으로 저장하는 LFS형태의 파일 시스템으로 오류 복구가 용이한 장점이 있다. wear-leveling을 따로 할 필요가 없는 것처럼 인식되고 있으나 JFFS2에도 wear-leveling을 미약하지만 하고 있다. 그래서 좀더 효과적인 wear-leveling을 위한 지움정책이 필요하다.

3. 개선된 JFFS2 지움정책

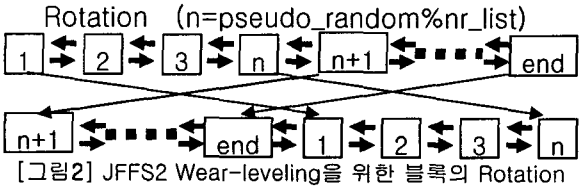
3.1 JFFS2의 지움정책

JFFS2는 파일 시스템을 초기화하는 과정에서 블럭들을 슈퍼 블록의 해당 리스트에 삽입하여 관리한다. 블록의 물리적인 특성은 고려되지 않으며, 해당 데이터의 종류에 따라 각각의 리스트로 분류되어 관리된다. 삭제는 리스트의 앞에서부터, 삽입은 리스트의 뒤에서부터 이루어진다. 다른 파일 시스템의 지움정책에서 말하던 블록의 수명이나 valid data의 크기에는 상관없이 bad_used_list, dirty_list, 혹은 clean_list의 최초의 블록을 지울 대상 블록으로 하고, 시스템이 시작한 때부터 지나간 틱수를 counting 하는 jiffies를 이용해서 대부분 dirty_list에서 지울 블록을 가져오도록 한다.(그림1) JFFS2에서는 파일 시스템을 초기화 하는 과정에서 Garbage Collection을 담당하는 스레드를 만들어 놓고 미리 예약되어 있는 남은 블록의 개수로 판단하여 지움 작업을 시작 한다.[5]



[그림 1] JFFS2 List에서 블록의 삽입과 삭제

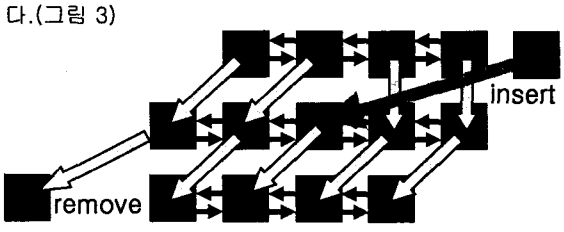
현재 JFFS2에서는 wear-leveling을 위해서 파일 시스템을 만들고 리스트를 구성한 후에 random넘버를 해당 리스트의 갯수로 나머지 연산한 만큼 리스트의 순서를 바꾸는 작업을 하지만 매번 초기화 마다 한번만 일어난다.



[그림2] JFFS2 Wear-leveling을 위한 블록의 Rotation

3.2 빈도수를 적용시킨 JFFS2 지움정책

JFFS2에서는 다른 파일 시스템처럼 wear-leveling을 위한 작업을 하지는 않고 jiffies에 의해서 결정된 리스트의 head에서 지울 블록을 가져왔다. 그러나 빈도수를 이용한 JFFS2에서는 효과적인 cleaning policy를 구현하기 위해 list를 구성하는 단계와 이후에 블록의 삽입 시점에서 각 erase block의 빈도수를 가지고 리스트를 재구성한다. JFFS2는 블록을 리스트로 관리하면서 리스트의 앞에서 제거하고 리스트의 꼬리 부분으로 삽입을 한다. 따라서 JFFS2의 뛰어난 파일 관리 능력을 그대로 사용하고, 동시에 효과적인 wear-leveling을 구현하기 위해서는 다른 파일 시스템에서 사용하던 것처럼 계산값을 가지고 블록을 비교하여 중간에서 하나씩 뽑아내는 방법으로는 할 수 없다. 즉 블록이 리스트에 삽입 되는 시점에서 관리를 해주어야 한다는 것이다. 블록에 빈도변수를 만들어서 지움 작업이 발생 할 때 마다 카운트를 하나씩 증가 시킬 것이다. 그리고 free리스트에서 대기 중이던 블록들은 쓰기 작업으로 dirty, clean list로 삽입되는 과정에서 해당 리스트 블록들의 빈도 변수와 자신의 빈도 변수를 비교하여 같거나 혹은 큰 블록의 이전으로 삽입이 된다. 이렇게 하면 리스트가 앞쪽에서부터 뒤쪽으로, 빈도 변수가 낮은 순서에서 빈도 변수가 높은 순서대로 재구성이 이루어 진다.(그림 3)

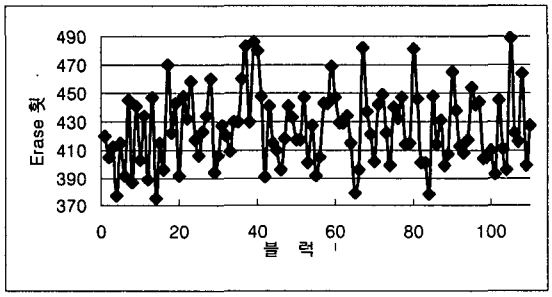


[그림 3] 빈도수 적용 리스트에서 블록의 삽입과 삭제

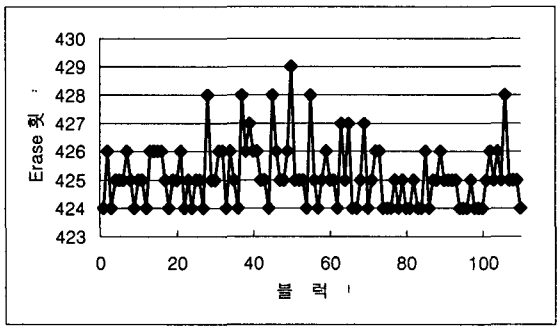
이 방법으로 기존의 JFFS2에서 제공하는 안정적인 파일 관리 능력을 그대로 제공 받으면서, wear-leveling을 위해 다른 파일 시스템처럼 복잡한 계산 과정을 하지 않아도 된다. 지울 대상 블록을 찾기 위한 복잡한 계산 과정을 하지 않고 블록의 빈도 변수만 비교하면 되기 때문에 기존의 방법보다 효과적이고, 간단하다. 이전의 JFFS2에 비해서는 리스트를 스캔 하여 위치선정 하는데 시간이 소요 되기는 하지만, 이 작업을 통해서 wear-leveling이 개선되는 효과를 가져 올 수 있다.

4. 실험 결과

실험은 JFFS2와 유사한 환경을 만들어 놓고 시뮬레이션을 통하여 수행 하였다. 16Mbyte 플래시 메모리를 사용 기준으로 블록의 개수는 110개 한 블록의 크기는 131072bit로 테스트 하였다. 10만회의 읽고 쓰기 작업을 한 결과 아래와 같은 결과를 얻었다.



[그림 4-1] JFFS2 블록당 지움횟수



[그림 4-1]빈도 적용 JFFS2 블록당 지움횟수

그림 4-1은 이전의 JFFS2의 결과로 각 블록당 지움 횟수가 최하 370부터 최고 490까지 120회의 차이를 보이고 있다. 반면 그림 4-2는 빈도 변수를 적용시킨 지움 정책의 결과로 최고424회부터 최하 430회에 걸쳐서 블록의 지움횟수 차이가 10회를 나타내고 있다. 즉 빈도변수 적용 시킨 경우에 블록이 전체적으로 고르게 사용되었다는 것을 말한다. 이는 리스트를 구성하면서 빈도수가 높은 블록들이 리스트 뒤쪽에 머물러 있으면서 재 사용될 기회가 별로 없고 앞쪽에 있던 빈도수가 적은 블록들이 계속해서 재사용 되면서 전체적인 균형을 맞추기 때문이다.

아래의 [표 1]은 두 가지 경우의 지움 횟수의 평균과 분산을 살펴본 것이다. 두 경우가 평균 값은 같지만 기존의 JFFS2에서는 669였는데 반해 빈도수를 적용시킨 새로운 wear-leveling을 사용한 경우에서는 1.2정도로 큰 분산의 차이를 보여준다. 분산은 유의 수준 5%에서 검정 절차를 거쳐서 충분히 작다는 검증 결과를 보였다. [6]

	JFFS2	빈도수 적용
평균	425.14	425.17
분산	669.67	1.19

[표 1] JFFS2와 빈도수 적용 지움 정책의 평균과 분산

이렇게 하여 빈도수를 이용한 지움 정책을 사용 함으로써 JFFS2의 뛰어난 파일 관리 능력과 기능을 그대로 유지하면서 wear-leveling을 개선 시켜서 블록의 지움 횟수를 메모리 전체에 골고루 분산 시킴으로써 플래시 메모리의 수명을 연장시킬 수 있다는 것을 확인 하였다.

5. 결론 및 향후 과제

시뮬레이션 단계의 실험이었지만 개선의 효과가 큰 것으로 보인다. 구현시켜서도 같은 효과를 얻을 것으로 기대된다. 리스트 구성시 마다 블록을 scan하기 때문에 이전의 JFFS2보다는 시간적인 문제가 생길 것으로 예상되나 얼마나 시간이 낭비될 것인가는 차후에 다시 검증 절차를 거친 후 스캔 방법을 향상 시켜서 보완 해야 할 것이다.

6. 참고 문헌

[1] Mendel Rosnblum and John K. Ousterhout, " The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems, Vol. 10, pp 26-52, 1992

[2] M. Wu and W. Zwaenepoel," eNVy:A Non-Volatile , Main Memory Storage System," Proc. Of the 6th International Conference if Architectural Support for Programming Languages and Operating Systems, pp 86-97, 1994

[3] Atsuo Kawaguchi, Shingo Nishioka and Hiroshi Motoada" A Flash_Memory Based File System" Proc. Of USENIX Technical Conference, pp. 155-164, 1995

[4] 김정기, 박승민, 김채규," 정보가전을 위한 플래시 파일 시스템에서 등급별 지움 정책과 오류 복구 방법" , 제5회 차세대 통신 소프트웨어 학술대회(NCS 2001), pp.938-941, 2001

[5] David Woodhouse, " JFFS: The Journaling Flash File System" , Technical Paper of RedHat Inc, 2001

[6] 강근석외 5명, " PC 통계학" , 자유아카데미, 1996