차세대 비휘발성 메모리를 이용한 플래시 메모리 파일 시스템의 개선*

김기홍*° 안성준** 최종무* 이동희† 노삼혁‡
*단국대학교 정보컴퓨터학부
**서울대학교 전기.컴퓨터공학부
† 서울시립대학교 컴퓨터과학부
‡ 홍익대학교 정보컴퓨터공학부
puresay@dankook.ac.kr, sjahn@ssrnet.snu.ac.kr, choijm@dandook.ac.kr,
dhlee@venus.uos.ac.kr, samhnoh@hongik.ac.kr

Enhancing Flash Memory File System Using Non-Volatile RAM

Kihong Kim*°, Seongjun Ahn**, Jongmoo Choi*, Donghee Lee†, Sam H. Noh‡
*Division of Information and Computer Science, Dankook University
**Department of Electrical Engineering and Computer Sciences, Seoul National University
† Department of Computer Science, University of Seoul
‡ Department of Computer Engineering, Hong-Ik University

요 약

본 논문에서는 차세대 비휘발성 메모리를 이용하여 기존 플래시 메모리 파일시스템의 마운트 시간을 단축시키고 메인 메모리 사용량과 전력소모량을 감소시킬 수 있는 기법을 제시한다. 제안된 기법은 소량의 비휘발성 메모리를 사용하여 블록의 상태 정보와 파일 메타데이터 및 데이터의 주소를 저장한다. 제안된 기법은 내장형 보드 상에서 구현되었으며, 실험 및 분석 결과는 마운트 시간을 크게 단축시키고 메인 메모리 사용량을 현저히 감소시켰음을 검증한다. 본 연구는 소량의 차세대 비휘발성 메모리를 내장형 시스템에서 어떻게 활용할 수 있는지에 대한 실용적인 방안을 제시하였으며, 그 효과를 실제 구현과 실험을 통해 정량적으로 검증하였다

1. 서론

최근 내장형 시스템이 다양한 분야에서 널리 사용되고 있으며 특히 휴대전화, MP3 재생기, PDA 등 모바일 장치의 수요가 급증하고 있다. 이러한 내장형 시스템은 다음과 같은 요구 조건을 가지고 있다. 첫째, 한정된 자원으로 동작이 가능해야 한다. 최근 디지털 통합이 이루어지면서 단일 기기가 다양한 기능을 수행하는 경우가 많아짐에 따라 한정된 자원을 보다 효율적으로 사용할 수 있는 방법이 요구된다. 둘째, 제한된 전력을 공급하는 배터리를 사용하므로 전력 소모량이 적어야 한다. 셋째, 장치의 기동시간이 짧아야 한다.

일반적으로 내장형 시스템의 저장장치로는 플래시 메모리가 많이 사용되고 있다. 플래시 메모리는 가볍고 크기가 작으며 내구성이 뛰어난 특징을 가지고 있다. 그러나 덮어쓰기가 불가능하다는 제약을 가지고 있어서 이를 극복하기 위해 플래시 메모리에 특화된 파일시스템이 사용되고 있다. 그러나 현재의 플래시 메모리 파일시스템은 앞서 언급한 내장형 시스템의 요구조건을 만족시키지 못하고 있다. 가장 큰 문제점은 마운트 시에 소요되는 시간이 길기 때문에 장치의 기동시간이 길어진다는 것과 메인 메모리 사용량이 크다는 것이다.

본 논문에서는 차세대 비휘발성 메모리를 이용하여 기존 플래시 메모리 파일시스템의 두 가지 단점을 보완할 수 있는 기법을 제시한다. 차세대 비휘발성 메모리(이하 NVRAM)에는 FeRAM, PRAM, MRAM 등이 있으며, 이러한 메모리들은 DRAM과 유사한 성능을 가지고 있고 전원이 공급되지 않아도 저장된 데이터를 유지하는 특성을 가지고 있으며 플래시 메모리와는 다르게 별도의 관리 기법을 필요로 하지 않는다[1]. 과거에는 높은 가격으로 인해 보편적으로 사용되지 않았으나, 현재는 반도체 기술의 발전으로 가격이 낮아지고 있는 추세이며 추후 플래시 메모리 및 기타 저장장치를 대체할 매체로 주목받고 있다.

본 연구의 접근 방법은 소량의 NVRAM에 플래시 메모리 파일시스템의 관리에 필수적인 정보를 저장함으로써 낮은 가격으로 자원 사용의 효율성을 증대하는 것이다. 제안된 기법의 구체적인 장점은 다음과 같다. 첫째, 파일시스템의 마운트 시 소요되는 시간이 감소한다. 특히, 기존의 플래시 메모리 파일시스템의 경우 저장된 파일의 개수 및 파일의 크기에 따라 마운트 시간이 길어지는 반면 제안된 기법의 경우 파일의 개수 및 크기에 관계없이 마운트 시간이 일정하다. 따라서 시스템의 기동시간을 크게 단축시킨다. 둘째, 메인 메모리에 유지해야 하는 정보의 양이 감소한다. 따라서 작은 용량의 메인 메모리만으로 시스템의 구성이 가능하다. 셋째, 시스템에 장착해야 하는 메인 메모리 용량을 감소시킴으로써 전체적인 전력 소모량을 감소시킬 수 있다.

본 연구에서는 리눅스와 윈도CE에서 사용되는 플래시 메모리 파일시스템인 YAFFS[2]의 코드를 수정하여 제안된 기법을 구현하였다. 구현된 파일시스템을 사용하여 수행한 실험 결과는 마운트 시 소요 시간이 실제로 크게 감소하였으며, 파일의 개수 및 용량에 관계없이 일정함을 검증하였다. 또한 앤드류 벤치마크를 수행할 때 사용된 메인 메모리를 측정한 결과 기존의 YAFFS에 비해

*본 연구는 한국과학재단특정기초연구(R01-2004-000-10188) 지원으로 수행되었음

83% 감소하였다. 본 연구의 의의는 소량의 차세대 비휘발성 메모리를 내장형 시스템에서 어떻게 활용할 수 있는지에 대한 실용적인 방안을 제시하였으며, 그 효과를 실제 구현과 실험을 통해 정량적으로 검증하였다는 데에 있다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 YAFFS에 대한 분석을, 3장에서는 수정된 YAFFS의 설계 및 구현을 설명한다. 4장에서는 성능평가 결과를 보여주며, 향후 연구 내용을 5장에서 정리한다.

2. 기존 YAFFS 분석

NAND 유형의 플래시 메모리는 여러 개의 블록으로 구성되어 있으며 블록은 다시 여러 개의 페이지로 구성된다. 각 페이지는 데이터 영역과 스페어 영역으로 나누어져 있다. YAFFS는 각 페이지의 데이터 영역에 파일의 메타데이터인 파일 헤더와 파일의 데이터를 저장한다. 파일 헤더는 파일이 속한 디렉터리를 나타내는 정보를 포함하고 있어서 이를 통해 계층적 디렉터리 구조를 표현하고 있다. 각 페이지의 스페어 영역에는 해당 데이터 영역에 저장된 정보의 유형, 데이터가 속한 파일의 식별자, 데이터가 파일의 어느 부분에 해당하는지를 나타내는 정보 등이 저장되어 있다. 스페어 영역에 저장된 정보를 태그라고 부르는데 이 태그는 YAFFS에서 파일의 메타데이터와 파일 데이터 간의 연관 관계를 나타내는 유일한 정보이다.

YAFFS는 모든 개체 간의 연관 관계를 역사상(inverse mapping)의 형태로 표현하고 있다. 예를 들어 디렉터리 구조를 표현함에 있어서 파일이 속한 디렉터리에 대한 정보를 파일의 메타데이터에 유지하고 있다. 또한 파일의 데이터가 자신이 속한 파일의 메타데이터를 가리키고 있다. 이는 일반적인 디스크 파일시스템과는 다른 구조이다. 그림 1은 리눅스에서 사용되는 디스크용 파일시스템인 Ext3에서 파일의 메타데이터와 데이터 사이의 관계를 표현하는 방법을 YAFFS와 비교하여 보이고 있다.

YAFFS에서 역사상의 형태로 개체 간의 관계를 표현할 수 있는 것은 NAND 유형의 플래시 메모리가 스페어 영역을 가지고 있기 때문이다. 역사상의 장점은 플래시 메모리에 유지해야 하는 정보의 양이 적다는 것과 파일의 데이터가 갱신되거나 파일과 디렉터리 간의 관계가 변경되었을 때 플래시 메모리 상에서 갱신해야 하는 정보의 양이 적다는 것이다. 반면에 직접 사상의 형태로 관계를 표현하는 경우, 플래시 메모리에서는 덮어쓰기가 불가능하므로 갱신해야 하는 정보의 양이 많아질 수 있다. 파일 데이터가 변경될 때마다 데이터는 플래시 메모리의 새로운 페이지에 저장된다. 만일 파일 헤더에 파일 데이터의 주소가 포함된다고 가정한다면 파일 헤더에 저장된 파일 데이터의 주소가 갱신되어야 하며 이를 위해 새로운 페이지에 갱신된 파일 헤더를 저장해야 한다. 그러나 역사상을 사용하면 파일 헤더를 갱신할 필요가 없다.

역사상의 단점은 원하는 정보를 찾기 위해 플래시 메모리의 모든 영역을 검색해야 한다는 것이다. 그러나 파일의 데이터나 파일의 메타데이터에 접근할 때마다 플래시 메모리의 모든 영역을 검색하는 것은 비효율적이다. 이를 해결하기 위해 YAFFS는 파일시스템을 마운트하는 시점에 플래시 메모리의 전체 영역을 검색하여 모든 파일의 헤더를 메인 메모리에 유지한다. 메인 메모리에 적재된 파일의 헤더들은 계층적 디렉터리 구조를 반영하기 위해 트리 형태를 유지한다. 이렇게 메인 메모리에 적재된 파일 헤더를 오브젝트라고 부른다. 또한 마운트 시에 모든 페이지의 태그를 검사하여 모든 파일 데이터의 주소를 메인 메모리에 유지한다. 메인 메모리에 구성된 파일 데이터의 주소 정보를 Tnode라고 부른다. 그림 2는 오브젝트와 Tnode가 구성된 예를 도시하고 있다.

이러한 기존 YAFFS의 접근 방법은 다음과 같은 문제를 야기한다. 첫째, 파일시스템을 마운트 할 때 모든 페이지를 스캔하여 오브젝트 및 Tnode를 구성해야 한다. 따라서 마운트에 소요되는 시간이 길다. 둘째, 모든 파일에 대한 오브젝트와 Tnode를 유지해야 하므로 파일의 개수가 많아질수록 많은 양의 메인 메모리를 필요로 한다.

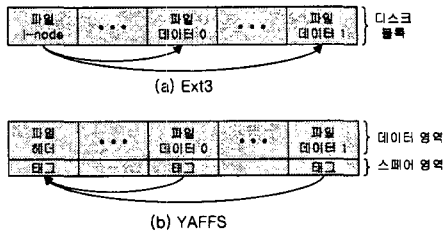

그림 1. 파일 메타데이터와 데이터 간의 관계

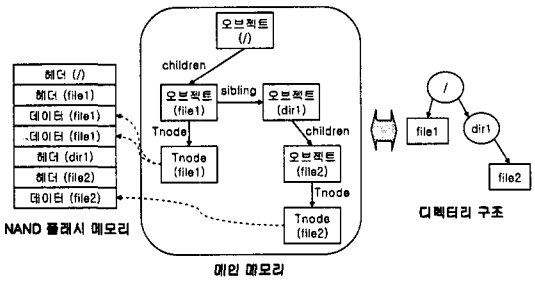

그림 2. 기존 YAFFS의 메인 메모리 자료구조

3. 수정된 YAFFS의 설계 및 구현

기존 YAFFS에서 역사상을 사용한 것은 플래시 메모리 상에 최소의 사상 정보를 유지함으로써 이를 갱신하기 위한 오버헤드를 최소화하려는 목적이다. 그러나 NVRAM은 덮어쓰기가 가능하므로 갱신의 오버헤드가 크지 않다. 따라서 NVRAM에 직접 사상의 형태로 정보를 유지할 경우 기존 YAFFS의 문제점을 개선할 수 있다. 본 연구에서는 기존 YAFFS의 코드를 수정하여 NVRAM에 부가 정보를 저장하고 파일 연산 시 이를 활용하도록 하였다. 플래시 메모리에 저장되는 데이터의 구성은 기존의 YAFFS와 동일하다. 따라서 기존 YAFFS가 구성된 플래시 메모리를 그대로 수정된 YAFFS에서 사용할 수 있다. 수정된 YAFFS에서 NVRAM에 저장하는 정보는 다음과 같다.

1) 플래시 메모리의 각 블록의 상태 정보: 기존 YAFFS에서는 마운트 시에 모든 페이지의 스페어 영역을 검사하여 메인 메모리에 블록의 상태 정보를 구성한다. 그러나 수정된 YAFFS는 마운트 시에 페이지를 검사하지 않으므로 블록의 상태 정보를 NVRAM에 유지해야 한다. 2) 헤더 정보: 파일 헤더의 물리적 위치와 디렉터리 구조를 저장한다. 또한 파일 데이터의 물리적 주소를 나타내는 Tnode에 대한 메모리 주소를 유지한다. 루트 디렉터리에 대한 헤더 정보는 고정된 위치에 저장된다. 헤더 정보를 저장하기 위해 사용되는 자료구조를 그림 3에 나타내었다. 3) Tnode: 기존 YAFFS에서 메인 메모리에 구성되었던 Tnode를 NVRAM에 저장한다.

이와 같이 수정된 YAFFS에서는 모든 파일 헤더와 파일 데이터가 저장된 물리 주소를 NVRAM에 저장함으로써 마운트 시에 플래시 메모리의 모든 페이지를 검사할 필요성을 없앴다. 또한, 파일이 접근되는 시점에 해당 파일에 대한 오브젝트를 생성할 수 있으므로 메인 메모리에는 현재 접근되고 있는 파일의 오브젝트만을 유지하면 파일 연산을 수행할 수 있으며, 메인 메모리에는 Tnode에 대한 별도의 정보를 유지할 필요가 없다. 따라서 수정된 YAFFS는 기존 YAFFS에 비해 메인 메모리 사용량을 감소시킨다.

4. 성능 평가

본 연구에서는 인텔의 PXA-255 프로세서를 사용하는 EZ-X5 보드를 사용하여 구현된 파일시스템의 성능을 평가하였다. 이 보드에는 64MB의 NAND 플래시 메모리가 장착되어 있다. NVRAM의 동작은 SDRAM을 사용하여 에뮬레이트 하였다.

```
struct header_info {
    int flash_memory_address;
    struct header_info *parent;
    struct header_info *child;
    struct header_info *sibling;
    struct Tnode *Tnode;
}
```

그림 3. 헤더 정보를 저장하기 위한 자료구조

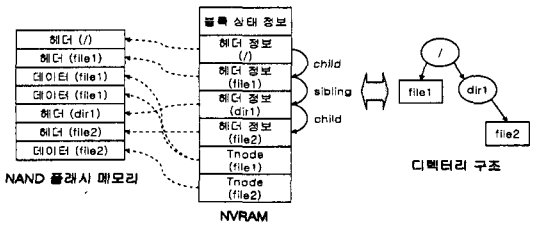

그림 4. 수정된 YAFFS의 NVRAM 자료구조

4.1 NVRAM 용량 분석

NVRAM에 저장되는 정보는 블록 상태, 헤더 정보, Tnode이다. 한 블록의 상태를 표시하기 위해서는 8 바이트의 메모리가 필요하다. 따라서 4096개의 블록으로 이루어진 64MB의 플래시 메모리를 사용할 경우 필요한 NVRAM 용량은 32KB이다. 그리고 헤더 정보 저장을 위한 자료구조는 20바이트로 이루어졌다. 64MB의 플래시 메모리를 사용할 경우 생성할 수 있는 최대 파일의 개수는 6만4천개이다(각 파일의 데이터가 한 개의 페이지에 저장되는 경우를 가정). 6만4천개 파일의 헤더 정보를 저장하기 위해 필요한 NVRAM의 용량은 약 1.28MB이다. 또한 64MB의 파일 데이터를 위해 필요한 Tnode의 용량은 약 300KB이다. 따라서 64MB의 플래시 메모리를 가정했을 때 필요한 NVRAM의 용량은 2MB이하로 플래시 메모리 용량의 약 3%에 해당한다.

4.2 마운트 시 소요 시간

그림 5는 파일 시스템에 저장된 파일의 개수에 따른 마운트 시간의 변화를 보이고 있다. 각 파일의 크기는 512KB이다. 결과에서 확인할 수 있듯이 기존 YAFFS의 마운트 시간은 파일의 개수 및 파일의 크기와 비례하여 증가한다. 이는 메인 메모리에 오브젝트와 Tnode를 생성할 때 소요되는 시간을 반영하고 있다. 반면 수정된 YAFFS의 마운트 시간은 일정하다. 파일이 저장되지 않은 경우에도 기존 YAFFS는 블록의 상태 정보를 구성하기 위해 모든 블록의 첫번째 페이지의 스페어 영역을 검사해야 하므로 수정된 YAFFS 보다 많은 시간이 소요된다. 이 경우 기존 YAFFS의 마운트 시간은 377ms, 수정된 YAFFS의 마운트 시간은 220ms로, 수정된 YAFFS의 마운트 시간이 약 42% 감소하였다.

4.3 메인 메모리 사용량

그림 6은 512KB의 크기를 가지는 64개의 파일을 생성한 상태에서 오픈된 파일의 개수가 변화될 때 각 파일시스템이 사용하는 메인 메모리 사용량을 보이고 있다. 기존 YAFFS는 마운트 시에 모든 파일의 오브젝트와 Tnode를 메인 메모리에 생성하므로 오픈된 파일의 개수와 관계없이 일정한 메모리 사용량을 보인다. 수정된 YAFFS에서는 파일이 접근되는 시점에서 오브젝트를 생성하므로 오픈된 파일의 개수가 증가함에 따라 메인 메모리 사용량도 증가한다. 두 결과의 차이가 큰 것은 수정된 YAFFS에서 Tnode 정보가 메인 메모리에 저장되지 않기 때문이다.

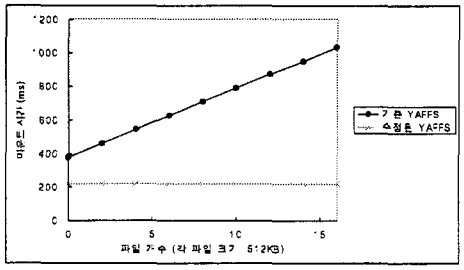

그림 5. 마운트 시간 비교

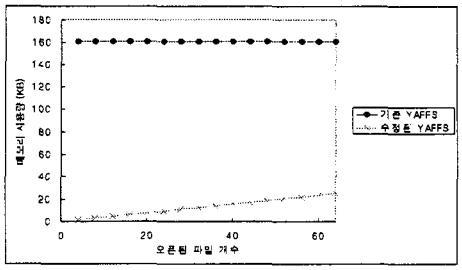

그림 6. 메인 메모리 사용량 비교

표 1. 앤드류 벤치마크 수행시간 (단위: ms)

	기존 YAFFS	수정된 YAFFS
Phase 1	67	69
Phase 2	2704	2736
합계	2771	2805

4.4 앤드류 벤치마크 (Andrew benchmark) 수행 시간

수정된 YAFFS는 파일 연산 시 기존 YAFFS에 비해 부가적인 작업을 수행한다. 파일 생성 및 갱신 시에는 NVRAM에 저장된 헤더 정보를 갱신해야 하며, 파일 오픈 시에는 헤더 정보를 검색하여 파일 헤더의 위치를 찾고 플래시 메모리에서 헤더를 읽어서 오브젝트를 생성해야 한다. 따라서 파일 연산에 소요되는 시간이 기존 YAFFS에 비해 길어진다. 이에 따른 오버헤드 분석을 위해 앤드류 벤치마크의 1단계와 2단계를 수행하였다. 1단계에서는 20개의 디렉터리를 생성하는 작업을 수행하며, 2단계에서는 70개의 파일을 복사하는 작업을 수행한다. 복사되는 데이터의 양은 총 355KB이다. 표 1은 측정된 수행시간을 보이고 있다. 수정된 YAFFS의 경우 기존 YAFFS에 비해 벤치마크 수행시간이 약 1.2% 증가되었다. 그러나 메인 메모리 사용량은 기존 YAFFS가 12.3KB, 수정된 YAFFS가 2.1KB로 약 83% 감소하였다.

5. 결론

본 논문에서는 NVRAM을 사용하여 플래시 메모리 파일시스템의 마운트 시간을 단축하고 메인 메모리 사용량을 감소시킬 수 있는 기법을 제안하였다. 향후 제안된 기법으로 감소시킬 수 있는 전력 사용량을 정량적으로 분석하고, NVRAM을 캐쉬로 사용하는 기능을 추가하여 성능을 보안하는 기법을 연구할 계획이다.

참고문헌

[1] 유병곤, 류상욱, 윤성민 유비쿼터스용 유니버셜 메모리 기술, 전자통신경향분석, 제 20권, 1호, 130~138, 2005
[2] http://www.alephl.co.uk/yaffs/yaffs.html, YAFFS Spec.