NAND 플래시 파일 시스템을 위한 효율적인 복구 기법

이승환°, 이태훈, 정기동
부산대학교 컴퓨터공학과
cat°@melon.cs.pusan.ac.kr, {withsoul, kdchung}@pusan.ac.kr

An Efficient Recovering Method for A NAND Flash File System

Seunghwan Lee, Teahoon Lee, Kidong Chung
Dept of Computer Engineering Pusan Nat'l Univ

요 약

본 논문은 NAND 플래시 메모리를 기반으로 한 임베디드 시스템에서 예기치 않은 오류에 대해 데이터 일관성 지원하는 파일 시스템을 제안 한다. 플래시 메모리는 하드디스크에 비해 작고, 내구성, 저 전력, 읽기속도 등 많은 부분에서 장점을 지니고 있어 임베디드 기기에 유리하다. 하지만 제자리 덮어쓰기가 되지 않고 추가적인 연산을 통해 지움 연산을 해야 하는 단점이 있다. 본 논문에서는 이미지 로그를 사용하여 시스템의 비정상적인 종료를 판단하고 플래시 메모리의 외부 갱신 쓰기 특징을 이용하여 파일 연산 전후 메타데이터의 타입을 다르게 하여 추가적인 로그 쓰기 연산 없이 파일 연산 중 오류를 판단하고 이전의 데이터로 복구를 할 수 있는 파일 시스템을 제안 한다. 또한 빠른 마운트를 지원 하는 파일 시스템에 복구 기법을 추가하고 마운트 시간을 실험 하였다. 실험 결과 정상적인 종료 시 YAFFS에 비해 76%~85% 마운트 시간을 감소 시켰고 비정상 적인 종료로 인해 오류 복구를 해야 할 때 마운트 시간은 YAFFS에 비해 40%~60%감소 시켰다. 그리고 파일에 대한 연산 시간도 YAFFS 와 차이가 없었다.

1. 서 론

최근 컴퓨터 산업이 빠르게 발전하고 있으며 그로인해 이동전화기, PDA, MP3 플레이어, PMP 등 다양한 개인용 멀티미디어 정보기기가 개발되어 널리 보급 되고 있다. 이러한 이동정보기기들에서는 대용량의 멀티미디어 데이터를 저장하고 처리하기 위한 효율적인 저장 장치가 필요하다. 일반적인 시스템에서의 하드 디스크 저장장치는 저비용으로 고용량의 데이터를 저장 할 수 있다. 하지만 내구성이 취약하고, 부피가 크고, 소비전력과 응답시간이 크다는 단점이 있어 소형의 개인용 멀티미디어 기기들에는 적합하지 않다. NAND형 플래시 메모리는 데이터 저장매체로서 하드디스크의 단점을 해결하기 때문에 소형의 이동 정보 기기들에 매우 적합한 특성을 가지고 있다. 하지만 제자리 갱신(in-place-update)이 불가능 하고 읽기에 비해 쓰기속도가 느리다는 단점이 있기 때문에 이러한 특성을 잘 고려해야 한다.NAND형 플래시 메모리를 사용하는 임베디드 시스템 개발 시 고려해야 할 사항 중 가장 대표적이라 말할 수 있는 사항은 다음과 같다. 첫째, 빠른 마운트를 지원해야 한다. 둘째, 데이터의 신뢰성을 보장하여야한다. 기존의 플래시 파일 시스템 중 JFFS2[1]와 YAFFS[2] 는 LFS(Log-Structure File System)에 기반을 두고 있다. YAFFS는 데이터 신뢰성을 위한 복구기법을 지원하고 있지 않으며 JFFS2는 복구기법을 지원하고 있다. YAFFS와 JFFS2에서는 마운트 과정에서 가장 최신의 데이터를 얻기 위해 전체 플래시 메모리 영역을 읽어야 하고 이는 플래시 메모리 마운트 시간이 길어지는 결과를 초래한다. 이는 빠른 부팅시간을 요구하는 임베디드 시스템에는 적합하지 않다. 이러한 문제점을 해결하기 위해 "임베디드 기기를 위한 NAND 플래시 파일 시스템의 설계" [3] 라는 논문에서 빠른 마운트를 지원하는 기법을 제안 했다. 하지만 복구기법을 통한 데이터의 일관성 유지를 보장하지는 않는다.

본 논문에서는 복구기법을 지원하여 빠르고 일관성 있는 파일 시스템을 위한 파일 시스템을 제안한다. 제안하는 기법은 플래시 이미지 영역과 데이터 영역을 나누어 빠른 마운트를 지원하며 무효 메타(invalid meta) 블록 타입을 두어 파일에 대한 연산중 비정상적인 종료가 발생 할 시 복구를 하여 플래시 메모리의 일관성을 유지한다. 본 논문의 구성은 다음과 같다. 2장에서는 기존에

"이 논문은 2단계 두뇌한국21사업에 의하여 지원되었음"

개발된 플래시 파일 시스템에 대한 설명을 하고. 3장에서는 본 논문에서 제안하는 플래시 파일 시스템에 대한 설명이 있겠다. 그리고 4장에서는 제안된 파일 시스템의 성능을 평가하고 5장에서 이 논문의 결론과 향후 과제에 대해 정리를 한다.

2.관련연구

2.1 JFFS2

JFFS는 1999년 개발된 LFS에 기반 한 플래시 파일 시스템이다. RedHat 에서는 여기에 NAND 플래시 지원 기능과 압축 기능을 더해서 JFFS2 를 발표 했다. JFFS2의 저널링 기법은 수정 시 기존파일을 수정하는 것이 아니라 수정된 내용을 로그형태로 저장하고 로그의 버전 정보를 이용해 최신 데이터를 구별 한다. 이러한 로그 구조를 이용하여 JFFS2는 전원 중단 오류가 발생해도 저널링 기법을 이용하여 복구할 수 있다. 과정을 살펴보면 JFFS2는 마운트 과정에서 플래시 메모리의 처음부터 마지막까지 순차적으로 로그 정보를 살펴보면서 파일 시스템의 구조를 파악한다. 오류회복은 마운트 과정에서 파일 시스템의 순차 접근으로 이루어지며 흩어져 있는 로그를 모두 읽어야 하므로 전체 플래시 메모리 영역을 다시 읽어야 하므로 마운트 시간이 길어진다.

이와 같은 단점을 보완해서 마운트 시간과 오류 회복 시간을 최소화 할 필요가 있다.

2.2 YAFFS

YAFFS는 2002년 개발된 NAND 플래시 메모리를 위한 대표적인 파일 시스템으로 JFFS2의 높은 메모리 사용률과 느린 마운트 속도를 개선하기 위해 개발된 NAND 플래시 전용 파일 시스템이다. YAFFS역시 LFS를 기반으로 하여 플래시 메모리에 대한 갱신 연산을 추가 연산으로 변형하여 처리하는 외부 갱신 기법을 사용한다. 이를 통해 플래시 메모리의 제자리 덮어쓰기(in-place-update)가 되지 않는 운제점을 해결하였다. 파일에 대한 갱신된 데이터가 다른 곳에 기록됨으로써 기존의 데이터가 플래시 메모리 공간에 그대로 남아있게 된다.

마운트시 블록의 첫 chunk를 읽어 빈 블록인지를 판단하고 데이터가 있는 경우만 블록내의 32개의 chunk를 읽어준다. 그러므로 JFFS2 보다 빠른 마운트를 지원 한다. 하지만 플래시 메모리 용량이 커짐에 비례하여 마운트 시간이 증가하고 복구를 위한 기법이 없기 때문에 파일 쓰기 연산도중 전력 차단으로 인한 비정상적인 종료가 발생하면 수정중인 파일에 대한 일관성이 유지 되지 않는다.

2.3 빠른 마운팅을 지원하는 파일 시스템

임베디드 시스템에서 빠른 부팅시간을 제공하기 위한 것으로 "임베디드 기기를 위한 NAND 플래시 파일 시스템의 설계" 에서 제안하는 파일 시스템이 있다. 플래시 메모리의 전체 공간을 플래시 이미지 영역과 데이터 영역으로 나누고 블록에 대한 타입을 메타 데이터 타입, 데이터 타입, 비어있는 타입 3가지로 나누어 고정된 크기의 플래시 이미지 블록에 전체 플래시 메모리 이미지를 기록한다. 마운트 시 플래시 메모리 이미지 영역을 읽고 메타 데이터 타입을 가지는 블록만을 읽어서 플래시 메모리 전체를 읽지 않고 마운트를 할 수 있어 YAFFS 비해 더욱 빠른 마운트 시간을 지원한다.

플래시 메모리의 전체 이미지는 언 마운트 시 기록을 하여 플래시 이미지 영역에 기록을 하고 있어 정상적인 언 마운트 후 재 마운트 시에는 빠른 마운트를 지원 하며 전체 파일의 일관성을 보장 한다. 하지만 비정상적인 종료가 있으면 최신의 플래시 전체 이미지가 기록이 되지 않아 재 마운트를 하면 이미지 영역의 데이터와 전체 블록의 데이터가 일치 하지 않아 전체 파일의 일관성을 보장 할 수 없다. 또한 플래시 메모리 전체영역을 읽어서 마운트를 한다고 해도 플래시 영역의 쓰기 작업 중 비정상적인 종료가 발생하면 파일에 대한 데이터가 깨어지게 되어 파일에 대한 일관성을 보장 할 수 없다.

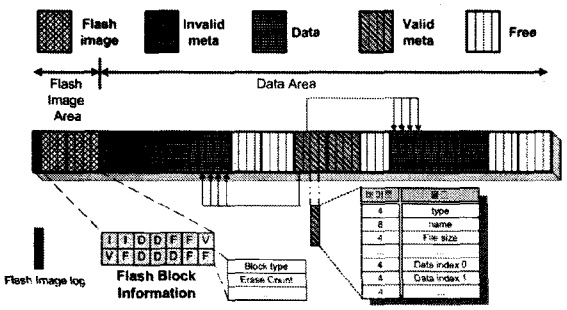
3.제안하는 기법

임베디드 시스템은 빠른 부팅 시간을 제공함과 동시에 예기치 않은 오류에 대한 복구기능을 가지고 있어야 한다. 본 논문에서는 플래시 메모리의 빠른 마운트 시간을 제공하면서 복구기능을 지원하는 NAND 플래시 파일 시스템을 제안 한다.

3.1 제안하는 파일 시스템 구조

본 논문에서 제안하는 파일 시스템 구조는 [그림 1]과 같다. 플래시 메모리 전체를 플래시 이미지 영역과 데이터 영역으로 나누고 데이터 영역에는 유효 메타(Valid meta) 타입, 무효 메타(Invalid meta) 타입, 데

이터(Data) 타입, 그리고 비어있는(Free) 타입의 블록들로 나누고 플래시 이미지 블록에 전체 블록 에 대한 이미지를 저장한다.



[그림 1] 파일 시스템 전체 구조

3.1.1 플래시 이미지 영역 (Flash image Area)

플래시 이미지 영역에는 전체 플래시 메모리에 대한 이미지를 압축하여 기록한다. 플래시 이미지는 블록의 타입 블록의 상태 및 유효한 페이지 수 등 마운트 과정에서 구축 할 정보를 포함 하고 있다. 최초 마운트 시 최신의 플래시 이미지가 RAM상에 저장 하고 마운트 과정에서 사용한다. 그 밖에도 블록 할당, 가비지 컬렉션(garbage collection) 에도 사용한다.

3.1.1.1 이미지 로그 (Flash Image log)

플래시 이미지 영역이 최신의 것인지를 판단하기 위해 정상적으로 언 마운트가 되었는지를 판단하기 위해서 이미지 로그를 이용한다. 최초 마운트가 끝나면 플래시 이미지 영역에 플래시 이미지 로그를 쓰고 언 마운트 시 로그를 삭제한다. 다시 마운트를 할 때 플래시 이미지 영역에 이미지 로그가 있는지 없는지 판단하여 시스템의 정상적인 종료 유무를 판단한다.

3.1.2 데이터 영역 (Data Area)

데이터 영역에의 각 블록은 유효 메타 타입, 무효 메타 타입, 데이터 타입, 비어 있는 타입의 블록들로 나뉜다. 유효 메타 타입, 무효 메타 타입의 블록에는 메타 데이터가 저장이 된다. 데이터 타입의 블록에는 파일의 데이터가 저장이 되고 비어 있는 타입의 블록은 쓰이지 않은 빈 블록 이다.

3.1.2.1 무효 메타 타입 & 유효 메타 타입 (Invalid meta & Valid meta)

메타 데이터는 파일에 대한 접근 시간, 수정 시간, 이름, 사이즈 등 파일의 정보와 실제 파일 데이터 위치 리스트를 저장 하고 있다. 파일에 대한 연산이 있을시 파일에 대한 정보 및 데이터 위치 수정으로 인해 메타데이터를 갱신 한다. 최초 파일의 접근시간 변경으로 인해 메타 데이터를 수정하여 저장 하고 파일의 연산이 끝나면 메타 데이터를 최신의 것으로 수정해서 플래시 메모리에 저장을 하는 것이 기본 이다. 결국 파일에 대한 생성 및 수정 작업 시 최초에 저장 되는 메타 데이터는 연산 후 새로 저장되는 최신의 메타데이터를 저장하고 난 뒤 삭제 처리가 된다.

본 논문에서는 파일 연산 시에 플래시 메모리에 메타 데이터가 수정, 삭제되는 특징을 이용하여 메타데이터가 저장될 블록을 무효 메타 블록 타입과 유효 메타 블록 타입으로 나누고 메타데이터를 관리 한다.

메타의 위치를 달리 하여 추가 적인 로그 쓰기를 하지 않고도 파일 연산에 대한 원자성을 보장하여 추가 적인 쓰기에 대한 부하를 늘리지 않고 파일 시스템의 일관성 유지를 지원한다.

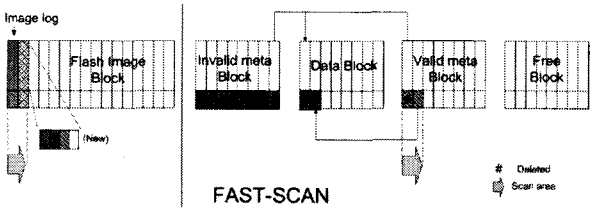
파일에 대한 최초 접근으로 수정 된 메타 데이터는 무효 메타 블록에 저장하고 파일에 대한 연산이 끝난 후 파일에 대한 최신정보를 유지 하고 있는 메타 데이터는 유효 메타 블록에 저장한다. 즉 정상적인 파일 정상적 연산 종료 후 무효 메타 블록에 있는 메타 데이터는 파일 연산이 끝나면 삭제가 되고 유효한 메타 블록의 메타 데이터만 유지를 한다. 이후 무효 메타 타입의 블록에 저장된 메타 데이터를 이용해 오류상황을 판단하고 복구를 할 수 있다. 추가 적으로 무효 메타 타입의 블록은 삭제 된 메타 데이터만 존재 하므로 이후 가비지 컬렉션을 하지 않고 바로 지움 연산을 할 수 있어 효율이 좋아진다.

3.2 스캔 및 복구

파일 연산중 예기치 않은 오류로 인해 시스템 종료 후 재시작을 시도 하였을 때 다음과 같이 두 가지 상황에 대한 고려를 해야 한다. 첫째는 최신의 플래시 메모리 이미지가 저장 되지 않은 상황이고, 둘째는 파일 데이터 저장 중 중단된 상황이다.

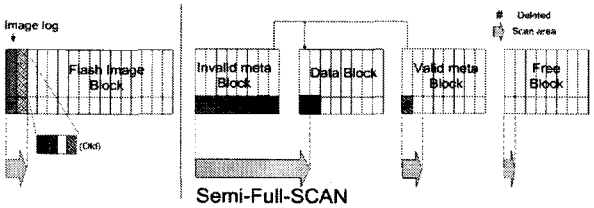
시스템이 다운되면 첫 번째 상황은 반드시 나타난다. 즉 플래시 메모리 이미지에 대한 최신 업데이트는 절대로 보장 할 수 없다. 그러므로 주기적으로 최신의 이미지를 저장 할 수 있도록 해야 한다. 하지만 전체 플래시 이미지에 대한 갱신은 큰 오버헤드를 발생 시키므로 갱신 횟수는 최소화해야 한다. 본 논문에서는 정상 적인

종료 시에만 최신의 플래시 이미지에 대한 저장을 한다.



[그림 2] 최신 플래시 이미지 기반 스캔영역

그러므로 정상적인 종료 후 마운트 시에는 [그림 2]와 같이 플래시 이미지를 바탕으로 빠른 마운트를 할 수 있다. 비정상 적인 종료 후 마운트 시에는 최신의 플래시 이미지가 아니므로 많은 연산으로 인해 플래시 이미지와 실제 플래시 메모리의 정보가 차이가 있다.



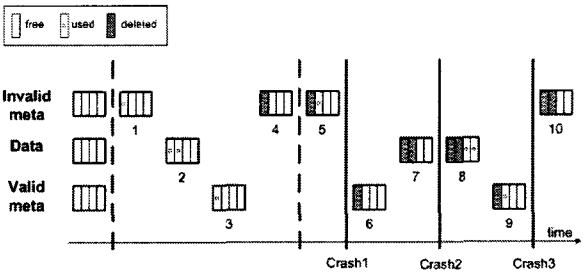
[그림 3] 비정상 종료 후 스캔 영역

신뢰성 있는 데이터 정보를 얻기 위해서는 [그림 3]과 같이 메모리 전체 블록에 대한 스캔이 필요 하다. 스캔 시 블록의 첫 번째 스페어 영역을 읽어 블록의 타입을 판단한다. 메타 데이터 블록과 무효 메타 데이터 블록에서는 블록 내부 페이지를 모두 읽어 자료 구조를 구축을 해야 하고 데이터 영역과 빈 블록은 첫 번째 스페어 영역만 읽어 최대한 빠르게 복구를 한다. 그리고 스캔시 이전 플래시 이미지를 활용해 블록에 대한 삭제횟수 정보 들은 유지한다.

파일 연산 시 종료 되어 무효 메타 타입의 블록에 삭제되지 않은 메타 데이터가 존재 할 경우 파일에 대한 연산 중 비정상적인 종료가 있었음을 판단하고 복구 기법을 적용해 파일 시스템의 일관성을 유지 할 수 있다.

이러한 복구 순서는 결국 전체 블록에 대한 접근이 있으므로 마운트 시간은 길어지게 된다. 하지만 데이터 블록에 대한 접근을 최소한으로 하였으므로 YAFFS 에 비해 빠른 마운트 시간을 보장 할 수 있다.

두 번째 상황으로 파일 데이터를 수정 또는 저장 시에 중단이 되었을 때 무효 메타 영역에 삭제하지 않은 메타 데이터가 존재한다. 이에 대한 상황 및 복구 대안은 다음과 같다.



[그림 4] 파일 연산중 오류 발생 위치

[그림 4] 에서 1~4까지는 정상적인 파일 쓰기를 할 때 각 블록 상태를 보여준다.

1. 최초 파일을 만들 때 파일에 대한 기본적인 정보를 가진 메타 데이터를 무효 메타 블록에 저장
2. 파일에 대한 데이터를 데이터 블록에 저장
3. 데이터의 위치 정보 및 파일에 대한 기본적인 정보를 가진 메타 데이터를 유효 메타 블록에 저장
4. 무효 메타 블록의 메타 삭제

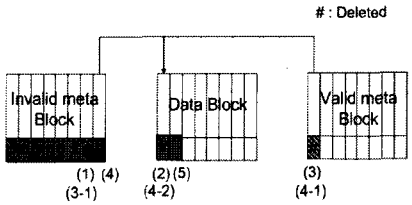
같은 파일에 대한 연산 중 예기치 않은 오류는 그림에서 나타난 Crash1~3 부분에서 일어 날 수 있다.

Crash1. 무효 메타 영역에 메타 데이터만 쓰고 이전 유효 메타 영역의 메타 데이터 삭제 전

Crash2. 무효 메타 영역에 메타 데이터 쓰고 이전 데이터 삭제 후 새로운 데이터 기록 전

Crash3. 무효 메타 영역에 메타 데이터 쓰고 데이터 영역에 새로운 데이터 기록 후 유효 메타 영역에 데이터 쓰고 이전 무효 메타영역 데이터 삭제 전

Crash 1, 3 번 경우 같은 파일에 대한 meta가 두 개 존재하므로 최초 마운트 시 플래시 이미지 영역을 읽고 유효 메타 블록에서 메타 데이터들로 파일 데이터 페이지에 대한 위치를 파악 할 수 있다. 그러므로 무효 메타 블록의 메타데이터를 삭제하면 복구가 완료 된다.



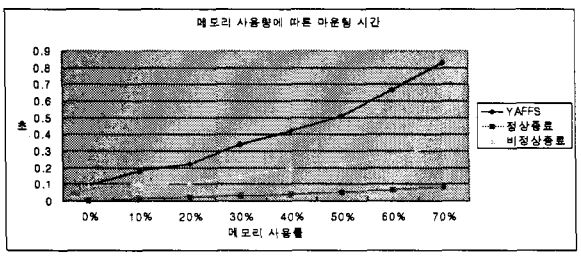
[그림 5] 파일 연산 시 플래시 메모리 동작

Crash 2 번 경우는 파일에 대한 수정 된 데이터가 써지기 이전 또는 도중에 중단이 되었으므로 수정 전 상태로 데이터를 복구 시킨다.

[그림 5]에서 (1),(2),(3),(3-1)까지의 연산은 정상적으로 파일연산이 되었을 때 플래시 메모리에 데이터를 쓰는 동작 순서를 보이고 있다. 이 파일에 대해 다시연산을 한다면 (4)에서 무효 메타 영역에 이전 파일에 대한 위치 및 파일 접근 정보를 갱신하고 (4-1)에서 이전 메타 데이터 지운다. 이후 (4-2)에서 이전 데이터를 삭제하고 (5)에서 새로이 수정 된 데이터를 기록하고 (6)에서 메타 데이터를 갱신 한다. Crash 2 는 [그림 5]에서 (5)이전에 오류가 발생 한 상황 이다. 이 때 무효 메타 블록에서 삭제되지 않은 메타 데이터의 수정 전 데이터 위치를 이용해 수정 전 파일을 복구 할 수 있다. 새로 생성 중인 파일에 대한 연산이라면 무효 메타 영역의 메타 데이터에 데이터 위치에 대한 정보가 없으므로 파일을 삭제 한다. 쓰다가 중지된 데이터는 속한 블록에 대한 포인터가 없으면 블록에 대한 지움 연산을 실시하고 삭제한다.

4.성능 평가

본 논문에서는 복구 기법에 대한 제안을 하고 빠른 마운트를 지원 하는 파일 시스템에서 적용을 하여 마운트 시간에 영향을 미치지 않고 복구를 지원 하는지에 대한 실험을 하였다.



[그림 6] 메모리 사용량에 따른 마운팅 시간

실험은 YAFFS 그리고 제안하는 기법에 대한 정상적인 종료시의 마운트 시간과 비정상적인 종료 시 제안하는 기법의 마운트 시간을 플래시 메모리 사용량에 따라 측정하여 [그림 6]과 같은 결과를 얻었다. YAFFS의 경우, 마운트 시점에서 전체 플래시 공간을 읽어 파일 시스템을 구축한다. 제안된 기법의 경우 최초 비정상 종료에 대한 유무를 판단하고 정상 종료를 감지하면 플래시 이미지를 이용하여 메타 데이터 블록만을 읽어 YAFFS 에 비해 75%~85%에 가까운 마운트 시간이 감소되었다. 비정상 종료를 감지하면 전체 블록에 대한 접근을 하고 유효 메타 영역과 무효 메타 영역을 읽어 파일 시스템을 구축하고 파일 연산 중 오류에 대해서 복구를 하여 정상 종료 보다는 느리지만 YAFFS에 비해 50%에 가까운 마운트 시간이 감소되었다.

또한 파일 연산 시의 시간을 측정 하여 추가적인 쓰기 시간에 대한 성능을 측정 해보았다. 측정은 10k 사이즈의 파일을 200개 만들어 10회 연산을 하였고 연산 시간은 다음 과 같다.

<표 1> 10K 사이즈 파일 200개 생성 시에 따른 시간 (단위:sec)

	최소값	최대값
YAFFS	5.996279	6.005941
Propose	5.996582	6.006588

<표 1> 에서 파일 생성시의 연산 시간은 YAFFS와 제안하는 기법은 큰 차이 가 없다고 할 수 있다. 그러므로 제안하는 기법은 따로 로그를 쓰지 않으면서 복구를 할 수 있어 동일한 파일 연산 시간을 가지는 YAFFS 보다 효율 적 이라고 할 수 있다.

5.결론 및 향후 과제

본 논문에서는 임베디드 기기를 위한 복구 기법 기반의 NAND 플래시 파일 시스템을 제안하였다. 이미지 로그를 이용하여 시스템의 비정상적인 종료 유무를 판단하고 메타 데이터를 저장 하는 블록의 타입을 무효 메타 블록 타입과 유효 메타 블록 타입으로 나누어 파일에 대한 연산 중 비정상 적인 종료를 판단하여 복구 하여 신뢰성 있는 파일 시스템을 지원한다.

정상 적인 종료 후 마운트 시에는 플래시 이미지 영역을 기반으로 빠른 마운트를 지원한다. 비정상적인 종료 후 플래시 이미지 데이터와 플래시 메모리간의 일관성을 유지 할 수 없을 시 메모리 전체 블록에 대한 접근을 해야 한다. 하지만 데이터 블록에 대한 전체 읽기를 하지 않으며 복구 기법을 지원하여 YAFFS 보다 빠른 결과를 내고 있다.

앞으로 경량화 된 플래시 이미지 갱신 및 할당 될 블록에 대한 예측으로 더욱 빠른 복구를 지원하는 기법이 연구 되어야 할 것이다.

참고 문헌

[1] D. Woodhouse, "JFFS: The Journaling Flash File System," *In proceedings of the Ottawa Linux Symposium (OLS)*, RedHat Inc., 2001.

[2] Aleph One Company, "The Yet Another Flash Filing System (YAFFS)," http://www.yaffs.net/

[3] 박송화, 이태훈, 정기동, "임베디드 기기를 위한 NAND 플래시 파일 시스템의 설계"