# 재구성 가능한 FAT 호환 통합 플래시 메모리 소프트웨어 구조

## (Reconfigurable Integrated Flash Memory Software Architecture with FAT Compatibility)

김 유 미 [†] 최 용 석 [††] 백 승 재 [†††] 최 종 무 [††††]
(Yumi Kim) (Yongsuk Choi) (Seungjae Baek) (Jongmoo Choi)

**요 약** 소형 USB 저장장치에서부터 대용량 데이터베이스 서버에 이르기까지 플래시 메모리의 활용범위가 더욱 확장되어 감에 따라 저장된 데이터의 호환성은 플래시 메모리 관리 소프트웨어의 중요한 고려사항이다. 이를 위해 FTL(Flash Translation Layer)과 FAT 파일시스템이 플래시 메모리 관리를 위한 사실상 표준 소프트웨어로써 사용되고 있다. 그러나 동일한 FTL과 FAT 파일시스템을 다양한 하드웨어로 구성된 시스템에서 구동하는 경우 각각의 요구조건을 만족할 수 없는 문제가 발생한다. 따라서 본 논문에서는 재구성 가능하며 FAT 표준 데이터의 호환성 및 향상된 기능을 제공하는 통합 플래시 메모리 관리 소프트웨어인 INFLAWARE(INtegrated FLAsh softWARE)를 제안한다. 제안된 기법은 실제 플래시 메모리가 장착된 시스템에 구현되었으며, 실험을 통해 본 논문에서 제안한 기법이 기존 기법 대비 최대 27%, 평균 19%의 메모리 사용량 감소 효과를 가져 올 수 있으며 또한 map_destroy 기법의 적용을 통해 최대 21%, 평균 10%의 성능 향상이 있음을 보인다.

**키워드** : 플래시 메모리, FAT, FTL, 호환성, 재구성, 성능 향상

***Abstract*** As deployments of Flash memory are spreading out rapidly from tiny USB storages to large DB servers, interoperability become an indispensable requirement for Flash memory software architecture. For the purpose, many systems make use of the conventional FAT file system and FTL (Flash Translation Layer) software as a de facto standard. However, the tactless combination of the FAT file system and FTL does not satisfy diverse other requirements of a variety of systems. In this paper, we propose a novel reconfigurable integrated Flash memory software architecture, named INFLAWARE (INtegrated FLAsh softWARE) that supports not only interoperability but also reconfigurability and performance enhancement. Real implementation based experimental results have shown that INFLAWARE can achieve improvements of memory footprint up to 27% with an average of 19%, compared with the conventional FAT and FTL combination. Also, by using map_destroy technique, it can reduce response times of various applications up to 21% with an average of 10%.

**Key words** : Flash memory, FAT, FTL, Interoperability, Reconfigurability, Performance Enhancement

· 이 연구는 단국대학교 대학원 연구보조장학금의 지원으로 이루어진 것임

† 정 회 원 : 하이닉스 반도체 F-Solution 개발팀 연구원
raspberi@nate.com
†† 정 회 원 : (주)Irumtek 선임연구원
hicys@hotmail.com
††† 정 회 원 : 단국대학교 컴퓨터공학과
baeksj@dankook.ac.kr
†††† 종신회원 : 단국대학교 컴퓨터공학과 교수
choijm@dankook.ac.kr

논문접수 : 2009년 9월 29일
심사완료 : 2009년 10월 27일



정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제1호(2010.1)

## 1. 서 론

저전력, 충격에 대한 내구성, 소형, 빠른 속도 등을 특징으로 하는 플래시 메모리는 생산 기술 발전으로 인해 지속적인 대용량화 및 용량 대비 가격 하락이 가속화되고 있다. 이에 따라 플래시 메모리는 휴대폰, MP3, PMP 등 소형 전자제품의 저장장치로 활발히 사용되고 있으며[1,2], 서버나 노트북 등에 사용되는 SSD(Solid-State Disk)[3,4], 혹은 센서 노드(sensor node)를 위한 저장장치[5], 대용량 데이터베이스 서버의 트랜잭션 처리를 위한 저장소[6]에 이르기까지 그 활용 범위 또한 더욱 넓어지고 있다.

USB 플래시 드라이브나 SD(secure digital) 카드와 같이 다양한 이동 시스템에서 사용될 수 있는 플래시 메모리 기반 저장 장치의 특성상 플래시 메모리에 기록된 데이터의 호환성은 이를 관리하는 소프트웨어 설계의 중요한 고려 사항이다. 이를 위해 그림 1에 나타난 것처럼 단순한 구조를 가지고 있는 FAT 파일시스템이 플래시 메모리의 사실상 산업계 표준으로 사용되고 있다. 또한 플래시 메모리 고유의 특성을 감추고 상위 수준에 하드디스크와 같은 일반적인 블록 장치로 보여줌으로써 기존의 파일시스템을 사용할 수 있도록 해주는 소프트웨어인 FTL(Flash Translation Layer)이 FAT 파일 시스템과 함께 사용되고 있다.

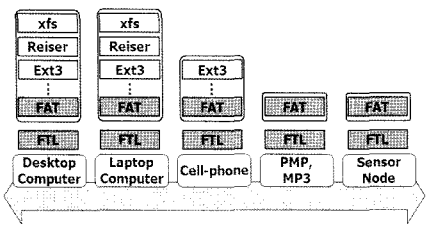


그림 1 플래시 메모리의 사용과 관리 소프트웨어

그러나 데이터의 호환성을 위해 범용 컴퓨터 기반으로 개발된 FTL과 FAT 파일시스템을 낮은 성능의 CPU와 최소한의 메모리로 구성되는 센서노드와 같은 열악한 하드웨어 환경에서 운용하기에는 많은 제약이 따른다. 반대로 상대적으로 풍부한 컴퓨팅 자원을 가지고 있는 컴퓨터나 스마트 폰과 같은 환경에서는 기존 FAT 파일시스템 보다 높은 수준의 성능과 신뢰성을 요구하고 있다. 이를 위해 호환성이 필요할 경우에는 FAT 파일 시스템을 사용하고, 추가 특성이 필요할 경우 Ext3나 xfs 같은 파일 시스템을 사용하는 경우도 발생한다.

본 논문에서는 재구성 가능한 통합 플래시 메모리 관리 소프트웨어(INtegrated FLAsh softWARE, 이하 INFLAWARE)를 제안한다. INFLAWARE는 FAT 표준과 호환되는 데이터의 상호 호환성을 제공함과 동시에 기존 시스템에서 제공하기 어려웠던 부가적인 기능 제공과 다양한 시스템에 적합하게 재구성이 가능하다는 특징을 가진다. 구체적으로 FAT와 FTL을 통합하여 서로의 정보를 활용하는 층간 최적화(cross-layer optimization)을 제공하며, 통합할 때 각 기능을 컴포넌트 기반으로 명확히 구분하고 재구성 도구를 지원하여 다양한 시스템의 자원 특성에 맞게 재구성할 수 있다.

제안된 기법은 400MHz로 동작하는 Xscale CPU, 64MB SDRAM, 64MB NAND 플래시 메모리 등이 장착되어 있는 임베디드 보드 상에서[8], 리눅스의 VFAT [9] 파일시스템과, NFTL[10]을 기반으로 구현되었다. 또한 실험을 통해 제안된 INFLAWARE가 기존의 VFAT과 NFTL 구성에 비해 27% 적은 용량의 메모리 자원을 필요로 하며, 최대 21%, 평균 10%의 향상된 성능을 제공함을 확인할 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대해 살펴보며, 3장에서는 INFLAWARE의 설계 및 구현에 대해 설명한다. 4장에서는 실험결과를 보이며, 끝으로 5장에서는 결론 및 향후 연구계획에 대해 기술한다.

## 2. 관련 연구

플래시 메모리는 덮어쓰기 제약, 삭제연산의 필요성, 연산 수행 단위와 시간의 비대칭성, 제한된 횟수의 삭제 연산 가능, 베드 블록(bad block)의 발생 가능 등 기존 저장장치와는 다른 특징을 보인다. 따라서 하드디스크 등의 기존 저장장치 기반으로 개발된 파일시스템을 플래시 메모리 기반 저장 장치에 그대로 적용하는 것은 많은 문제점을 가지고 있다.

플래시 메모리 기반 저장장치를 효율적으로 관리하기 위한 기법은 크게 둘로 나뉜다. 첫째, 플래시 메모리의 특성을 고려한 전용 파일시스템을 사용하는 방법이다. LFS(Log-Structured File System)[11]의 동작 구조를 이용한 JFFS[12], YAFFS[13], CFFS[14]등의 파일시스템이 대표적인 예이다.

둘째, 기존 하드디스크와 다른 플래시 메모리의 다양한 특성을 숨기고 일반적인 블록 장치로 에뮬레이션 해주는 FTL과 FTL 상위에 전통적인 파일시스템을 구축하여 사용하는 방법이다. TrueFFS, RFS[15], KFAT [16]과 ZFS[17] 등이 이러한 구조를 채택하고 있다. FTL이 논리적인 페이지 번호와 물리적인 페이지 번호를 연결하는 단위에 따라 블록 맵핑 기법 FTL[10]과 페이지 맵핑 기법 FTL [18]로 나뉠 수 있다. 또한 이들의 장점을 취한 로그블록 기반 맵핑 FTL이 존재한다[19].

본 연구는 FAT 호환성을 제공한다는 측면에서는 두 번째 기법들의 접근 방법을 따른다. 반면, 파일 시스템

과 FTL의 기능을 통합하고 성능 향상을 시도한다는 측면에서는 두 번째 기법들의 접근 방법과 유사하다. 본 연구와 동일한 목표로 진행된 기존 연구에는 [7]이 있다. 이 연구에서는 FTL과 FAT 파일시스템을 사용하는 플래시 메모리 기반 시스템의 성능 향상을 위한 기법으로 파일시스템 수준에서 삭제된 데이터들에 대한 정보를 FTL에 알려줌으로써 FTL의 가비지 컬렉션 수행 시 불필요한 복사 오버헤드를 최소화 시킬 수 있는 map_destroy 기법을 소개하였으며 이를 통한 성능 향상이 가능함을 보였다.

## 3. INFLAWARE 설계 및 구현

본 논문에서 제안하는 INFLAWARE는 상호 호환성 제공, 부가적인 확장 기능 제공, 재구성 기능 제공이라는 세 가지 목표를 두고 설계되었다. 각 목표에 대한 구체적인 내용은 다음과 같다.

첫째, 저장된 데이터의 상호 호환성이 제공되어야 한다. 이를 위해 플래시 메모리 기반 저장장치에서 사실상 산업계 표준으로 사용되는 FAT 파일시스템을 기반으로 하여 INFLAWARE를 설계하였으며, FAT 파일시스템을 플래시 메모리 위에서 동작시키기 위해 NFTL을 사용하였다. 사실 FAT 파일시스템과 FTL은 서로 독립적이며 따라서 NFTL이 아닌 다른 FTL을 적용하는 것도 가능하다.

둘째, 기존 기법에서 제공하지 못했던 다양한 부가 기능이 제공되어야 한다. 이를 위해 본 연구진은 FAT 파일시스템과 FTL을 단일 소프트웨어 모듈로 통합하였다. 이를 통해 map_destroy[7]와 같이 기존 소프트웨어 구조에서 작용시키기 어려웠던 다양한 속도 향상 기법이나 신뢰성 향상 기법을 적용하는 것이 가능하다.

셋째, 운용될 시스템의 환경에 적절하게 소프트웨어의 재구성이 가능해야 한다. 이를 위해 본 연구진은 doxygen[20]을 이용하여 INFLAWARE의 모든 함수간의 종속성을 파악하였고 이를 주요 기능 별로 분류하였다. 또한 각 기능 별로 사용 유무를 결정하여 재구성할 수 있도록 설계하였다.

그림 2는 INFLAWARE의 주요 구성 요소를 보여준다. 열악한 컴퓨팅 자원을 가지고 있는 시스템을 위해서는 데이터 호환성을 제공하기 위한 최소한의 파일 연산과 FAT 테이블 관리 등의 코드만이 포함되는 'Core configuration'을 선택할 수 있다. 또한, 'Directory configuration'을 선택함으로써 mkdir(), readdir()과 같은 디렉터리 관련 연산을 지원할 수 있다. 또한 비교적 풍부한 컴퓨팅 자원을 가지고 있는 시스템에서는 'Enhanced configuration'을 선택함으로써 보다 높은 수준의 성능과 신뢰성을 제공받는 것이 가능하다.

본 연구진은 제안한 INFLAWARE의 기능을 리눅스

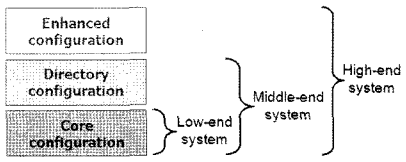


그림 2 INFLAWARE의 재구성 기능

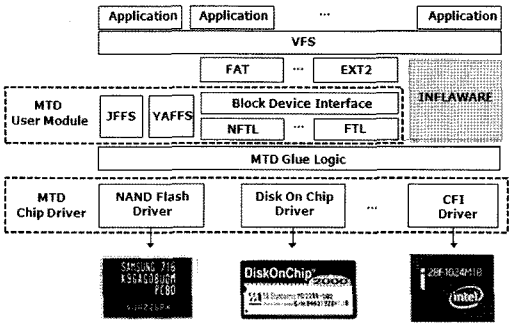


그림 3 리눅스의 플래시 메모리 소프트웨어 구조

환경에 구현하여 검증하였다. 그림 3은 리눅스 운영체제의 플래시 메모리 관리 소프트웨어 구조를 보여준다. 그림 3에서 볼 수 있듯이 리눅스의 플래시 메모리 관리 소프트웨어 구조는 크게 VFS(Virtual File System)계층, 파일시스템 계층(예: Ext2, VFAT 등), MTD(Memory Technology Device)계층의 3단계의 계층 구조를 이루고 있다.

MTD계층은 다시 MTD 사용자 모듈 계층과 MTD 글루 로직(glue logic) 계층, MTD칩 드라이버 계층으로 나누어진다. MTD 사용자 모듈 계층은 블록 수준 맵핑을 제공하는 NFTL, 페이지 수준 맵핑을 하는 FTL 등의 오픈 소스 FTL 소프트웨어들을 포함한다.

즉, 리눅스 상에서 FAT 파일시스템을 통해 플래시 메모리를 저장장치로 사용하기 위해서는 총 세 개의 모듈(VFAT, FAT, NFTL)을 필요로 한다. NFTL은 상위 수준에 일반적인 블록 읽기/쓰기 인터페이스를 제공하며, VFAT은 FAT 파일시스템이 제공하는 기본적인 FAT 파일시스템 관련 인터페이스를 이용하여 VFAT 고유의 기능을 제공한다.

본 논문에서 제안하는 INFLAWARE는 MTD 글루 로직 상위에, FTL과 파일시스템의 기능을 동시에 수행할 수 있도록 구현되었다. 즉, INFLAWARE는 단일 소프트웨어 모듈을 통해 FTL의 기능과 파일시스템의 기능을 동시에 제공한다.

그림 4는 구현된 INFLAWARE의 내부 소프트웨어 컴포넌트를 보여준다. 가장 하단 부에는 주소변환(Address mapping), 가비지 컬렉션(Garbage collection), 마모 균등화(Wear-leveling)등 FTL의 기능을 하는 컴포

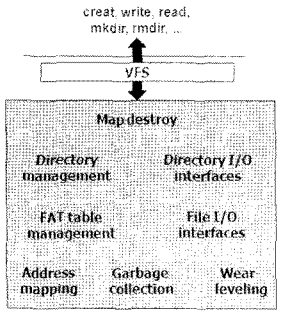


그림 4 INFLAWARE의 논리적인 구조

넌트가 존재한다. 이들 컴포넌트 상위에는 FAT 테이블 관리 및 기본적인 파일 입출력 인터페이스 기능을 하는 컴포넌트가 존재한다. 이 두 개의 컴포넌트가 그림 2에서 설명된 Core configuration에 해당된다.

디렉토리 관리 메커니즘 및 입출력 인터페이스 컴포넌트가 다음 계층에 존재하며 하위 컴포넌트와 함께 그림 2의 Directory configuration을 구성하게 된다. 가장 상단부에는 부가 기능을 제공하는 컴포넌트가 위치한다. 현재는 map_destroy 기능이 구현되어 있으며, 역시 하위 컴포넌트와 함께 그림 3의 Enhanced configuration을 구성하게 된다.

그림 4에 소개된 각 컴포넌트들은 사전 정의된 인터페이스를 통해서만 상호작용한다. 따라서 각 컴포넌트는 새로운 컴포넌트로 대체 하는 것이 가능하다. 또한 각 컴포넌트 간에 종속성이 없도록 구현되었다. 구체적으로, doxygen을 통한 분석 결과를 바탕으로 컴파일 되는 시점에 사용자의 정의에 따라 필요한 컴포넌트만 컴파일에 포함될 수 있도록 'define'과 'ifdef' 지시어를 활용하여 구현하였다.

그림 5는 리눅스의 커널 설정(configuration)기능을 통해 구현한 INFLAWARE의 재구성 과정을 보여준다. 그림 5에서 볼 수 있듯이 INFLAWARE는 그림 2에서 설명된 세 가지 구성과, 각 구성에 속해있는 함수의 지원 여부를 선택할 수 있도록 구현되어 있다. 이를 통해 일반 사용자들이 특정 시스템의 환경에 걸 맞는 INFLAWARE를 손쉽게 구성할 수 있다.

## 4. 실험 결과

구현된 INFLAWARE는 400MHz로 동작하는 Intel PXA255 CPU와 64MB DRAM, 64MB NAND 플래시 메모리 등의 장치가 부착되어 있는 임베디드 개발 보드 상에서 실험되었으며, 실험을 위해 Linux 2.6.21을 동작시켰다. 비교대상으로는 Linux의 VFAT[9] 파일시스템과 MTD(Memory Technology Device)내에 존재하는 NFTL[10]을 사용하였다.

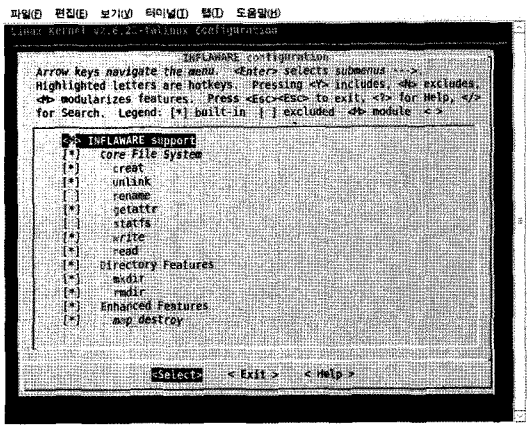


그림 5 재구성 기능 지원

### 4.1 재구성 기능 확인

그림 6은 INFLAWARE 각 구성의 코드 사이즈와 기존(FAT+NFTL)의 코드 사이즈를 보여준다. 그림 6에서 볼 수 있듯이 기존(FAT+NFTL)이 93.71KB의 코드 사이즈를 보이는 반면, INFLAWARE의 경우 Core configuration, Directory configuration, Enhanced configuration이 각각 68.13KB, 75.62KB, 84.14KB의 코드사이즈를 보이고 있다. 이들 결과를 통해 INFLAWARE가 기존 기법 대비 최대 27%(Core configuration의 경우), 평균 19% 적은 메모리 사용량을 필요로 함을 알 수 있다.

그림 6의 결과는 스택(stack)이나 힙(heap)과 같은 동적 메모리 사용량은 배제하고, 컴파일 된 이미지의 크기를 통해 정적인 메모리 사용량만을 비교하고 있다. 그러나 FAT 파일시스템은 내부적으로 kmalloc()등 커널 내부의 메모리 할당 함수를 호출하는 것이 드물며, 고정된 크기의(예: 8KB) 스택을 활용하여 동작하기 때문에 실제의 메모리 사용량과 큰 차이가 없을 것으로 예상된다. 한편, FAT 파일 시스템에서 가장 공간을 많이 사용하는 기능이 FAT 테이블 관리이며 이 기능이 Core

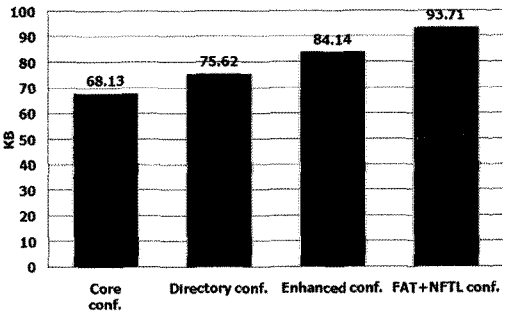


그림 6 INFLAWARE의 구성 별 메모리 사용량

configuration에 포함되기 때문에 본 연구진이 처음 기대했던 것 보다는 메모리 사용량 절약이 적었다. 이 부분에 대해서는 향후 더욱 연구할 것이다.

### 4.2 데이터 상호 호환성 실험

그림 7은 INFLAWARE의 실제 동작 과정을 통해 FAT 호환성 지원 여부를 보여준다. 그림 7에서는 우선 fat.ko, vfat.ko, nftl.ko 세 개의 모듈을 'insmod' 명령을 이용하여 커널에 적재한다. 그런 뒤 mount 명령을 사용하여 블록 장치를 마운트 하고, a.txt 파일을 생성하여 test_string 이라는 테스트 문자열을 파일에 기록하였다. 이후 같은 블록 장치를 inflaware.ko 모듈을 통해 마운트 하여 a.txt 파일에 대한 연산을 수행함으로써 INFLAWARE 완벽한 FAT 호환 파일시스템 기능을 제공함을 확인할 수 있었다.

```
[root@falinux ~]$ insmod fat.ko
[root@falinux ~]$ insmod vfat.ko
[root@falinux ~]$ insmod nftl.ko
                      ⋮
[root@falinux ~]$ mount /dev/nftla1 /mnt/temp
[root@falinux ~]$ echo "test string" >> /mnt/temp/a.txt
[root@falinux ~]$ ls -l /mnt/temp
-rwxr-xr-x    1 root     root            12 Jan  1 00:16 a.txt
[root@falinux ~]$ umount /mnt/temp
[root@falinux ~]$
[root@falinux ~]$ insmod inflaware.ko
[root@falinux ~]$ mount /dev/nftla1 /mnt/temp
[root@falinux ~]$ ls -l /mnt/temp
-rwxr-xr-x    1 root     root            12 Jan  1 00:16 a.txt
[root@falinux ~]$ cat /mnt/temp/a.txt
test string
[root@falinux ~]$
```

그림 7 데이터 상호 호환성 확인

### 4.3 성능 비교 결과

그림 8은 map_destroy 기능이 추가된 INFLAWARE와 기존 시스템 간의 성능 비교를 위해 Postmark 벤치마크[21]와 [22]의 연구에서 소개된 Phone, Fax machine 벤치마크의 수행 결과를 보여준다. 실험을 위해 각 벤치마크 수행 전에 저장장치의 이용률(utilization)을 50%로 만든 뒤, 각 벤치마크 수행 시간을 측정하였다. 그림 8에서 볼 수 있듯이 map_destroy기능이 추가되어 있지 않은 Core configuration은 기존(FAT+NFTL)과 유사한 성능을 보이고 있으며, map_destroy기능이 추가된 Enhanced configuration은 기존 기법 대비 최대 21.75% (Postmark의 경우), 평균 10%의 성능 향상을 보이고 있다.

그림 9는 다양한 이용률에서의 Postmark 벤치마크의 수행결과를 보여준다. 결과를 통해 map_destroy기능이 추가된 Enhanced configuration은 이용률이 높은 때뿐만 아니라 모든 이용률에서 고르게 성능 향상을 보임을 알 수 있다.

낮은 이용률에서도 기존 기법대비 성능 향상을 보이는 이유는 다음과 같다. 첫째, 파일시스템 수준에서 모

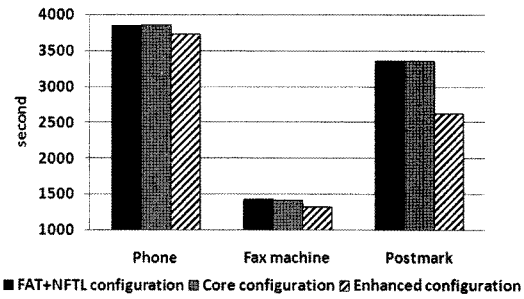


그림 8 벤치마크 성능 실험 결과(이용률: 50%)

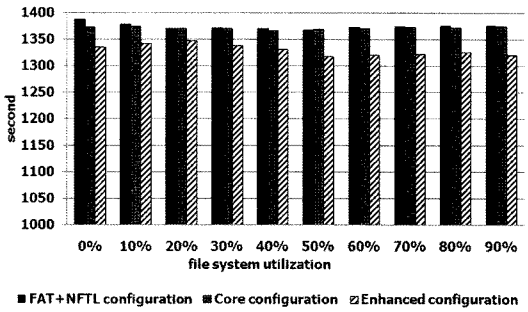


그림 9 다양한 이용률에서 벤치마크 실험 결과

든 파일이 삭제되었다 하더라도 FTL 수준에서는 100%의 이용률이 유지되고 있을 수 있기 때문에 map_destroy 기법이 성능 향상을 유발한다. 둘째, 이용률이 낮은 경우라 할지라도 map_destroy기법이 불필요한 가비지 컬렉션을 최소화시킴으로 인해 성능이 향상된다.

## 5. 결론 및 향후 연구 계획

본 논문에서는 저장된 데이터의 상호 호환성과 성능 향상을 제공하는 플래시 메모리를 위한 통합된 소프트웨어인 INFLAWARE를 제안하였다. 제안된 기법은 리눅스 2.6.21에 실제 구현되었으며, 또한 실험을 통해 제안된 기법이 평균 19% 메모리 사용량을 절약할 수 있으며, map_destroy 기법을 도입함으로써 평균 10%의 성능 향상을 가져옴을 확인할 수 있었다. 향후 본 연구는 고성능 시스템을 위해 신뢰성 강화 기능 도입, map_destroy외의 다양한 성능 향상 기법 개발 등 다양한 부가 기능 추가와 센서 노드와 같은 열악한 하드웨어 환경에서도 원활히 동작할 수 있는 구성을 제공하도록 확장해 나갈 것이다.

## 참 고 문 헌

[1] F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, and J. Tauber, "Storage Alternatives for Mobile Computers," in *Proceedings of the First Symposium on Operating Systems Design and*

*Implementation (OSDI)*, pp.25-37, 1994.
[2] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings," in *Proceeding of the 14th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 229-240, 2009.
[3] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating Server Storage to SSDs: Analysis of Tradeoffs," in *Proceedings of the fourth ACM European conference on Computer systems (EUROSYS)*, pp.145-159, 2009.
[4] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," *Proceedings of the 2008 USENIX Annual Technical Conference*, pp.57-70, 2008.
[5] H. Dai, M. Neufeld, and R. Han, "ELF: An Efficient Log-Structured Flash File System For Micro Sensor Nodes," in *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (SenSys)*, pp.176-187, 2004.
[6] S. Lee, B. Moon, C. Park, J. Kim, and S. Kim, "A case for flash memory SSD in enterprise database applications," in *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pp.1075-1086, Vancouver, BC, June 2008.
[7] 김성관, 이동희, 민상렬, "FAT 호환 플래시 메모리 파일시스템을 위한 성능 최적화 기법", *한국 컴퓨터 종합 학술대회 논문집*, vol.32, no.1(A), pp.796-798.
[8] EZ-X5 Evaluation Board, "http://www.falinux.com"
[9] Linux VFAT File System, "http://bmrc.berkeley.edu/people/chaffee/vfat.html"
[10] M-Systems, "Flash-Memory translation layer for NAND flash(NFTL)," 1998.
[11] 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, pp.26-52, 1992.
[12] D. Woodhouse, "JFFS: The journaling Flash file system," Ottawa Linux Symposium, 2001.
[13] Aleph One, "YAFFS: Yet another flash file system," www.yaffs.net.
[14] S. Lim and K. Park, "An Efficient NAND Flash File System for Flash Memory Storage," *IEEE Transactions on Computers*, vol.55, no.7, July, 2006.
[15] Samsung RFS, "Robust FAT File System," www.samsung.com/global/business/semiconductor/products/flash/Products_FlashSoftware.html.
[16] M. S. Kwon, S. H. Bae, S. S. Jung, D. Y. Seo, and C. K. Kim, "KFAT: Log-based Transactional FAT Filesystem for Embedded Mobile Systems," in *2005 US-Korea Conference*, ICTS-142, 2005.
[17] ZFS, "Zeen File System," Zeen Information Technologies, Inc., http://zeen.snu.ac.kr/.
[18] Intel Corporation, "Understanding the Flash Translation Layer (FTL) Specification," 1998.
[19] J. M. Kim. J. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A Space-efficient Flash Translation Layer for CompactFlash Systems," *IEEE Transactions on Consumer Electronics*, vol.28, pp.366-375, 2002.
[20] Doxygen, "Source Code Documentation Generator Tool," www.stack.nl/~dimitri/doxygen/index.html.
[21] J. Katcher, "PostMark: A New File System Benchmark," Technical Report TR3022, Network Appliance Inc., 1997.
[22] E. Gal and S. Toledo, "A transactions Flash file system for microcontrollers," *Proceedings of the 2005 USENIX Annual Technical Conference*, pp. 89-104, 2005.

김 유 미
2008년 단국대학교 전기전자컴퓨터공학부 졸업(공학사). 2009년 단국대학교 대학원 컴퓨터학과 컴퓨터과학 졸업(공학석사). 2009년~현재 (주)하이닉스반도체 연구원. 관심분야는 운영체제, 임베디드 시스템

최 용 석
2007년 단국대학교 컴퓨터과학과 졸업(이학사). 2009년 단국대학교 대학원 정보컴퓨터과학과(이학석사). 2009년~현재 단국대학교 대학원 컴퓨터학과 박사과정 2009년~현재 (주)Irumtek 선임연구원 관심분야는 운영체제, 차세대 저장장치 등

백 승 재
2005년 단국대학교 컴퓨터 공학과 졸업(공학사). 2004년~현재 비트 컴퓨터 강사. 2007년 단국대학교 대학원 정보컴퓨터 과학과(이학석사). 2007년~현재 단국대학교 대학원 컴퓨터학과 박사과정. 관심분야는 운영체제, 임베디드 시스템, 차세대 저장장치 등

최 종 무
1993년 서울대학교 해양학과 졸업(이학사). 1995년 서울대학교 대학원 컴퓨터공학과(공학석사). 2001년 서울대학교 대학원 컴퓨터 공학과(공학박사). 2001년~2003년 유비쿼스 주식회사 책임 연구원. 2003년~현재 단국대학교 공과대학 컴퓨터학부 컴퓨터공학 전공 부교수 2005년~2006년 UC Santa Cruz 방문 교수. 관심분야는 운영체제, 임베디드 시스템, 차세대 저장장치 등