FlaSim: 리눅스 커널 모듈을 이용한 FTL 에뮬레이터

(FlaSim: A FTL Emulator using Linux Kernel Modules)

최 화 영 † 김 상 현 †
(Hwayoung Choe) (Sanghyun Kim)
이 승 원 †† 박 상 원 †††
(Seoungwon Lee) (Sangwon Park)

요 약 플래시 메모리의 성능평가 실험 환경 구성은 플래시 메모리가 장착된 제품들이 동작하는 시스템으로 이뤄진다. 이와 같은 방법은 물리적이고 비용적인 제약이 따르게 된다. 또한 실험에 쓰이는 입력 데이터와 FTL 알고리즘의 성능평가를 위한 결과 데이터인 트레이스의 추출 방법이 까다롭고 힘들다. Oracle의 경우 트레이스 추출이 불가능하고, MySQL, SQLite는 트레이스 추출이 가능하더라도 결과의 정확성이 보장되지 않는 문제점이 있다. 따라서 본 논문에서는 디바이스 드라이버를 통해 물리적 제약을 없애고 트레이스 추출을 쉽고 간편하게 하여 정확한 실험 결과 분석이 용이하도록 FTL 에뮬레이터를 설계하고 구현한 내용에 대해 다룬다. 본 논문에서 제안한 FTL 에뮬레이터인 FlaSim은 플래시 메모리의 데이터 저장 매커니즘과 동일한 동작을 하도록 구현하고 리눅스 커널 모듈을 사용하여 필요한 기능을 추가할 수 있다. FlaSim은 모듈 적재 방식을 사용하기 때문에 FTL 알고리즘 및 플래시 메모리에 대한 실험의 확장성을 향상시킨다. 또한 다양한 응용프로그램에 적용이 쉽고, 플래시 메모리에 대한 실험의 제약이 되는 물리적인 비용을 줄일 수 있다. 게다가 트레이스 추출하는 데 쉽고 효율적인 방법을 제공하여 결과 도출 및 분석 시 시간적, 시스템적 제약을 받지 않아 효율성이 큰 장점이 있다. 추후 많은 FTL 알고리즘 및 플래시 메모리에 대한 실험과 연구에 도움이 될 것으로 예상된다.

키워드 : 플래시 메모리, 플래시 변환 계층, 데이터베이스, 리눅스 모듈 프로그램

Abstract Many researchers have studied flash memory in order to replace hard disk storages. Many FTL algorithms have been proposed to overcome physical constraints of flash memory such as erase-before-write, wear leveling, and poor write performance. Therefore, these constraints should be considered for testing FTL algorithms and the performance evaluation of flash memory. As doing the experiments, we suffer from several problems with costs and settings in experimental configuration. When we, for example, replay the traces of Oracle to evaluate the I/O performance with flash memory, it is hard to extract exact traces of I/O operations in Oracle. Since there are only write operations in the log, it is impossible to gather read operations. In MySQL and SQLite, we can gather the read operations by changing I/O functions in the source codes. But it is not easy to search for the exact points about I/O and even if we can find out the points, we might get wrong results depending on how we modify source codes to get I/O traces. The FlaSim proposed in this paper removes the difficulties when we evaluate the performance of FTL algorithms and flash memory. Our Linux drivers emulate the flash memory as a hard disk. And we can easily obtain the usage statistics of flash memory such as the number of write, read, and erase operations. The FlaSim can be gracefully extended to support the additional modules implemented by novel algorithms and ideas. In this paper, we describe the structure of FTL emulator, development tools and operating methods. We expect this emulator to be helpful for many experiments and research with flash memory.

Key words : Flash Memory, Flash Translation Layer, Database, Linux Module Program

1. 서 론

플래시 메모리는 최근 대중화되고 있는 임베디드 장치에 많이 쓰이며 성능 향상을 위한 알고리즘과 기술에 대해서 연구가 이루어지고 있다[1]. 플래시 메모리에 대한 실험에 쓰이는 장치는 다분히 비싸고 다루기 힘든 단점이 있어 실험 비용이 증가하고 실험 환경 구성에 제약이 따른다. 또한 FTL 알고리즘 실험에 사용되는 I/O 패턴인 트레이스 추출이 정확하게 이루어지지 않는 문제점이 있다. Oracle의 경우 로그파일을 사용하여 쓰

· 이 연구는 2009학년도 한국외국어대학교 교내학술연구비의 지원에 의하여 이루어진 것임
· 이 논문은 2009 한국컴퓨터종합학술대회에서 'FlaSim: 리눅스 커널 모듈을 이용한 FTL 에뮬레이터'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 한국외국어대학교 컴퓨터 및 정보통신공학과
hychoe@dislab.hufs.ac.kr
shkim@dislab.hufs.ac.kr
†† 학생회원 : 한국외국어대학교 정보통신공학과
swlee@dislab.hufs.ac.kr
††† 종신회원 : 한국외국어대학교 정보통신공학과 교수
swpark@hufs.ac.kr

논문접수 : 2009년 8월 13일
심사완료 : 2009년 9월 30일

기 연산에 대한 분석만 가능하기 때문에 읽기 연산에 대한 분석이 불가능하다[2]. 이와 같이 쓰기 연산을 제외한 다른 연산의 성능 측정이 불가능한 문제점이 발생한다. MySQL의 경우, 소스 코드를 분석하여 I/O 발생하는 지점을 찾아 트레이스를 추출할 수 있지만 이것은 간단하지 않고, 운영체제의 버퍼를 거치기 때문에 직접적인 I/O 트레이스가 아닌 한계가 있다. 또한 수정된 부분에서 발생하는 추가적인 연산 오버헤드로 인해 결과의 정확성이 떨어지는 문제점이 발생한다. 이와 같은 기존 실험에서 사용되는 트레이스 추출 방법은 버퍼의 크기에 영향을 받으며, PC 사양과 다양한 실험 환경에 따라 추출이 어렵고 까다롭다. 따라서 플래시 메모리를 이용한 실험의 물리적 제약, 트레이스 추출의 어려움, 다양하지 못한 실험 조건 등의 문제점을 해결하기 위해 FTL 에뮬레이터인 FlaSim을 개발하였다.

기존 FTL 시뮬레이터[3,4]는 추출한 I/O 트레이스를 이용하여 I/O 횟수만 연산하는 것이었다. 이와 달리 FlaSim은 파일 시스템에서 요청하는 I/O 연산을 FTL 알고리즘에 따라 플래시 메모리 에뮬레이터를 통하여 실제 저장하고 읽도록 하고 있다. I/O 트레이스를 추출할 수 없는 경우에 실험할 수 없었던 기존 FTL 시뮬레이터와는 달리 어떠한 경우에도 실험을 수행할 수 있기 때문에 FlaSim은 다양한 환경에서 정확한 실험 결과를 얻을 수 있는 장점이 있다. 본 논문에서는 FlaSim의 전체적인 구조, 특징, 동작방법에 대해 설명하고 플래시 메모리에 대한 실험 및 향후 개선책에 대해 알아본다.

2. 플래시 메모리

그림 1은 NAND 플래시 메모리의 구조를 나타낸 것이다.

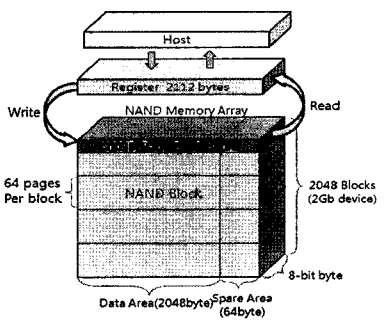

그림 1 NAND 플래시 메모리 구조

NAND 플래시 메모리는 일정한 크기의 섹터 단위로 읽기, 쓰기 연산을 수행하고 블록 단위로 소거 연산을 수행한다. 또한 덮어쓰기 연산이 불가능하고 특정 블록에 쓰기 연산이 발생하면 쓰기 전 소거 연산이 일어나는 특징이 있다[1]. NAND 플래시 메모리는 물리적으로 8비트 또는 16비트 양방향 데이터 버스를 통해 명령과 주소를 전송한다. 모든 NAND 플래시 동작은 명령 사이클로 시작한다. 명령 사이클에 사용되는 NAND 플래시 메모리 명령 인터페이스는 6개의 주요 제어 신호로 이루어진다. 제어 신호는 chip enable, write enable, read enable, command latch enable, address latch enable, ready/busy로 구성되고, 이 6개의 주요 제어 신호의 조합으로 이루어진 물리적인 명령 사이클을 통해 표 1과 같은 13가지 동작을 제어한다.

표 1 NAND 플래시 메모리 명령

NAND 플래시 메모리 명령	
PAGE READ	READ ID
PROGRAM PAGE	READ STATUS
RANDOM DATA READ	BLOCK ERASE
RANDOM DATA INPUT	RESET
PAGE READ CACHE MODE START	
PAGE READ CACHE MODE START LAST	
READ for INTERNAL DATA MOVE	
PROGRAM PAGE CACHE MODE	
PROGRAM for INTERNAL DATA MOVE	

기본적인 읽기, 쓰기 연산이 요구될 때, 플래시 메모리는 물리적으로 내부에서 처리되는 읽기, 쓰기 연산에 해당되는 명령과 소거 연산에 해당되는 블록 소거 명령, 리셋 명령으로 요구되는 동작을 수행한다[5]. NAND 플래시 메모리는 읽기, 쓰기, 소거 연산의 속도가 서로 다르며, 각 연산의 속도 차이와 부가적인 소거 연산으로 인한 성능 저하를 막기 위해 FTL(Flash Translation Layer)이라는 소프트웨어 계층을 둔다. FTL은 파일 시스템에서 플래시 메모리를 논리적 주소로 바라볼 수 있게 해주는 매개체 역할을 한다. 매핑 방법들과 읽기, 쓰기, 소거 연산 외에 할당, 교환, 합병 연산 등을 이용한 여러 FTL 알고리즘들이 있다[3].

3. FlaSim의 설계

3.1 FlaSim의 구조

그림 2는 본 논문에서 다루는 FTL 에뮬레이터인 FlaSim의 전체적인 구조를 그림으로 나타낸 것이다. FlaSim은 커널의 파일 시스템에 의해 nfdd(NAND Flash Device Driver)를 거쳐 플래시 메모리 역할을 하는 볼륨에 접근한다. nfdd는 블록 디바이스를 위한 NAND 플래시이다. nfdd는 읽기, 쓰기 연산이 발생하면 FTL 계층의 **read_sector**, **write_sector** 함수를 호출한다. FTL 계층은 파일 시스템의 읽기, 쓰기 요청을 플래시 메모리의 연산인 읽기, 쓰기, 소거 연산으로 변환하는 역할을 담당한다. 각 계층은 독립적으로 동작하기 때문에 FTL 계층을 모듈로 따로 두어 다양한 FTL 알고리

즘을 적용시킬 수 있다. NAND 플래시 에뮬레이터는 하드 디스크나 파일을 NAND 플래시 장치로 사용 가능하도록 하는 플래시 디바이스 드라이버이다. 하드 디스크에서 원하는 플래시 메모리의 크기만큼 할당받아 소블록과 대블록으로 설정할 수 있다. 각 계층의 디바이스 드라이버는 여러 가지 설정이 가능하여 실험 시 여러 가지 조합으로 동작할 수 있다.

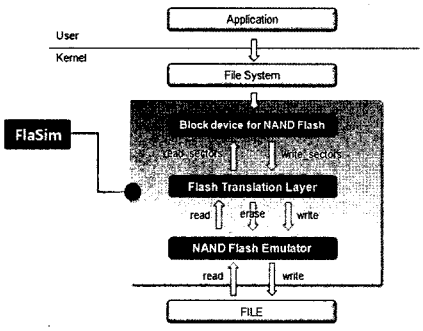

그림 2 FlaSim의 전체적인 구조

3.2 NAND 플래시를 위한 블록 디바이스

사용자의 읽기, 쓰기 연산 요청이 발생하면 파일 시스템에서 nfdd를 통해 FTL로 입력, 출력 연산을 요청한다. 다음 그림 3은 nfdd의 내부 동작과정을 나타낸 것이다.

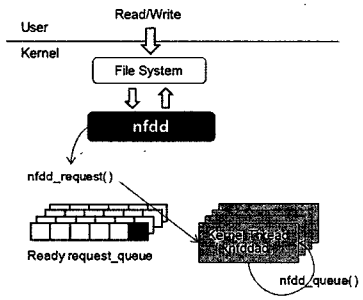

그림 3 NAND 플래시 디바이스 드라이버 내부 동작

nfdd는 커널 스레드의 상태 변환에 관여한다. 커널 스레드는 추상화된 플래시 메모리인 볼륨마다 하나씩 생성된다. 먼저 커널은 nfdd의 요청이 발생하면 knfddad 커널 스레드가 깨어나도록 하는 역할을 한다. 볼륨마다 생성된 커널 스레드는 자신의 요구 큐에 있는 작업을 처리하고 이 작업이 모두 끝나면 잠자기(sleep) 상태로 들어간다. 이러한 작업 스케줄링은 커널 내부에서 자동적으로 이뤄지지만 nfdd의 요청이 있어야만 요청 연산이 요구 큐의 대기상태로 변한다. nfdd에서는 FTL 모듈의 **read_sector, write_sector** 함수를 호출한 뒤 요청 연산을 처리하도록 요구한다.

3.3 NAND 플래시 에뮬레이터

FlaSim은 플래시 메모리의 데이터 저장 메커니즘을 반영한 디바이스가 필요하다. 이때 하드 디스크와 같은 특정 디바이스를 플래시 메모리처럼 사용하기 위해 NAND 플래시 에뮬레이터를 이용한다.

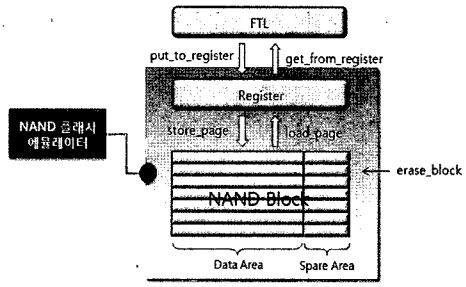

그림 4 NAND 플래시 에뮬레이터

그림 4는 NAND 플래시 에뮬레이터의 내부 동작 과정을 나타낸 것이다. NAND 플래시 메모리는 레지스터와 NAND 블록으로 데이터를 처리한다. 데이터의 읽기, 쓰기 연산이 발생하면 레지스터에서 플래시 메모리 데이터 영역으로 옮기는 연산이 발생한다. 실제 NAND 플래시 메모리에서는 레지스터로 인한 추가 연산을 명령 사이클을 통해서 제어하고 있다. 제어 사이클의 조합으로 이루어진 읽기, 쓰기, 소거 연산을 명령으로 정의해서 사용한다. NAND 플래시 에뮬레이터는 NAND 플래시 메모리의 데이터 저장 메커니즘을 반영하기 위해 5가지의 주요 함수로 구성하여 같은 기능을 하도록 구현하였다. NAND 플래시 에뮬레이터는 상위 계층인 FTL에서 요구하는 읽기, 쓰기 연산이 발생하면 주요 함수인 **put_to_register, get_from_register, store_page, load_page, erase_block**을 호출하여 데이터 연산을 처리한다. NAND 플래시 에뮬레이터는 레지스터와 블록사이의 데이터를 교환하는 **load_page, store_page** 함수를 사용하고, **put_to_register, get_from_register** 함수로 레지스터와 FTL 사이에서 데이터 교환을 담당한다. 블록 소거 연산은 **erase_block** 함수로 수행한다. 주요 5가지 함수를 이용하면 NAND 플래시 메모리와 같은 특징을 가진 디바이스로 사용할 수 있다. NAND 플래시 에뮬레이터는 기존 FTL 시뮬레이터[3,4]에는 없는 것으로서 플래시 메모리의 특징을 반영한 구성 요소이다. 이것을 통하여 FTL 알고리즘은 플래시 메모리의 특성과 동일한 형태로 데이터를 저장하거나 읽을 수 있기 때문에 더욱 정확한 실험 결과를 얻을 수 있다.

3.4.1 NAND 플래시 메모리 에뮬레이터 API

그림 5는 NAND 플래시 에뮬레이터의 주요 5가지 API를 기술한 것이다.

put_to_register	int put_to_register(struct flash *flashp, int col_addr1, int col_addr2, char *host_addr)
get_from_register	int get_from_register(struct flash *flashp, int col_addr1, int col_addr2, char *host_addr)
store_page	int store_page(struct flash *flashp, int page_addr)
load_page	int load_page(struct flash *flashp, int page_addr)
erase_block	int erase_block(struct flash *flashp, int block_no)

그림 5 NAND 플래시 에뮬레이터의 주요 5가지 API

주요 API를 사용하여 새로운 FTL 알고리즘 구현이 가능하다. put_to_register는 호스트 버퍼의 내용을 데이터 레지스터의 주어진 범위에 저장하는 역할을 한다. 인자로 사용된 flashp은 플래시 디바이스를 가리키는 포인터이고, col_addr1과 col_addr2은 각각 페이지의 시작 오프셋(offset)과 끝 오프셋을 담는 인자이다. host_addr는 호스트 버퍼의 주소를 가리킨다. col_addr1과 col_addr2를 이용해서 오프셋을 구하고 그 위치에 데이터를 저장한다. get_from_register는 데이터 레지스터의 일부 내용을 호스트 버퍼로 전달한다. 호스트 버퍼에 레지스터의 데이터를 메모리 복사하는 방식으로 동작한다.

store_page는 데이터 레지스터의 내용을 플래시에 저장한다. flashp 포인터와 페이지 번호를 저장하는 page_addr가 있다. 오프셋은 페이지 주소와 소블록 물리 페이지 크기를 곱한 뒤 데이터의 시작 오프셋을 더해서 계산한다. 실제 FTL에서 쓰기 연산이 요청되면 put_to_register를 수행하여 레지스터에 저장한 뒤 다시 레지스터에서 플래시에 저장하는 store_page를 호출한다. load_page는 페이지를 플래시에서 읽어 플래시 장치의 데이터 레지스터로 적재한다. erase_block은 블록을 소거한다. 인자는 flashp 포인터와 블록 정보를 저장하는 block_no가 있다.

4. FlaSim을 이용한 실험과 동작 결과

4.1 FlaSim의 동작

FlaSim은 디바이스 드라이버를 모듈로 적재하고 적재된 볼륨을 표준화된 파일 시스템으로 포맷한 뒤 사용이 가능하다. 수행되는 메커니즘은 플래시 메모리의 데이터 저장 메커니즘과 같다. FlaSim은 읽기, 쓰기 연산의 여러 항목의 통계치를 proc 파일 시스템으로 확인할 수 있다. proc 파일 시스템은 리눅스에서 디바이스 드라이버의 정보를 보여주는 가상 파일 시스템이다.

```
Volume name : /home/flasim/vol/nfdda.vol
   Log name : /home/flasim/vol/nfdda.log
   Operations : e0bff568   Company :    Model:

The number of operation calls
load_page     store_page     erase_block
     3791            257               1
get_from_register    put_to_register
             4355                514

The number of bytes transferred
dev2reg      reg2dev      reg2host      host2reg
2001648       135696        632463        132051
```

그림 6 플래시 메모리 정보

그림 6은 플래시 메모리 정보를 나타낸다. 그림 4의 NAND 플래시 에뮬레이터의 정보를 담고 있다. /proc/nfdd/nfdda/flash 경로로 proc 파일 시스템을 통해 확인할 수 있다. 볼륨의 이름 및 플래시 메모리에 읽기, 쓰기, 소거 연산을 하기 위해 호출된 오퍼레이션의 횟수가 확인이 가능하다. 플래시 메모리 정보는 NAND 플래시 에뮬레이터에서 사용한 주요 5개 API의 호출 횟수로 나타난다.

dev2reg와 reg2dev는 실제적으로 디바이스에서 레지스터로, 레지스터에서 디바이스로 전송된 바이트 수를 나타내고 reg2host와 host2reg는 레지스터에서 호스트로, 호스트에서 레지스터로 전송된 바이트 수를 나타낸다. 시스템 호출과 전송된 바이트 수를 통계적 수치로 나타냄으로써 정확한 결과를 추출할 수 있다. 또한 여러 종류의 FTL 모듈을 동시에 적재시켜 사용 가능하다.

4.2 FlaSim을 이용한 FTL 성능 평가 실험

FlaSim을 이용한 FTL 성능 평가 실험을 진행하는 것을 가정한다. FTL 알고리즘의 성능 평가 실험에 FlaSim을 사용했을 때 어떠한 결과를 도출해 낼 수 있는지 여부를 확인하고, FlaSim의 장점을 확인하기 위한 간단한 실험이다. 이 실험은 SQLite 데이터베이스를 이용한 것으로, 생성한 볼륨에 튜플을 삽입한 후 그 결과를 proc 파일 시스템을 이용해서 확인했다. 쓰기 연산에 사용된 릴레이션 스키마는 다음과 같다. test(col1, col2, col3)으로 test라는 이름의 테이블을 생성하고 각 속성의 도메인 타입은 정수로 설정하였다. 생성한 test테이블에 튜플 10만개를 삽입하였다. 실험한 FTL 알고리즘은 복사블록기법[6]과 로그블록기법[7]이다. 본 실험에서 사용한 플래시 메모리는 모두 소블록이며, 물리적 블록 사이즈를 1000개로 설정하였다. 볼륨 생성 후 해당 볼륨에 튜플을 삽입하였을 때, 복사블록기법과 로그블록기법 알고리즘이 어떤 차이가 있는지 분석하여 FlaSim의 유용성을 검증하였다.

4.3 FlaSim을 이용한 실험 결과 분석

표 2는 4.2절의 실험 조건으로 FlaSim을 사용한 실험을 한 후, NAND 플래시 에뮬레이터의 5가지 주요 함수의 호출 횟수를 추출한 표이다. 로그블록기법은 호스트로 데이터를 전송하는 get_from_register의 호출이 한 번도 일어나지 않는다. 이는 로그블록기법이 로그블록이 가득 차 합병(merge)연산이 일어나서 load_page가 호출되어도 데이터 레지스터에서 호스트로 섹터의 내용을

복사하지 않고 내부복사를 하기 때문에 튜플을 삽입하는 연산만 수행한다. 복사블록기법 또한 쓰기 연산에 대한 실험이기 때문에 호스트로 읽어오는 읽기 연산은 그 빈도가 낮아야 한다. 하지만 복사블록기법은 복사블록이 가득차서 합병연산이 일어날 때, 복사블록의 여유영역을 호스트로 복사해야 한다. 그 이유는 복사블록이 변동섹터(out-of-place)방식으로 기록되어있어 여유영역을 읽어 섹터의 논리적 번호를 구해야 하기 때문이다. 블록 삭제 연산의 경우 로그블록기법이 훨씬 많이 일어나는 것을 확인할 수 있으며 이것은 [4]의 결과와 유사하였다.

표 2 10만 번 쓰기 연산 결과

(단위 : 회)

실험 횟수 / 호출된 함수	복사블록기법	로그블록기법
load_page	3690662	1426827
store_page	651753	1665423
erase_block	14254	89178
get_from_register	3469762	0
put_to_register	1082600	1904019

로그블록기법은 블록 소거연산이 많이 일어나서 성능이 저하된다. 그 이유는 랜덤한 위치의 블록에 작은 쓰기 연산을 수행하는 경향이 있는 데이터베이스와 달리 로그블록은 서너 개의 로그블록에 데이터를 기록하여 로그블록이 랜덤한 위치의 쓰기 연산을 감당하지 못하기 때문이다. 랜덤한 쓰기 연산은 로그블록의 가용성(utilization)을 떨어뜨려 블록에 쓰기를 할 섹터가 많이 남았음에도 블록이 합병되기 때문이다. 로그블록의 개수를 증가시켜도 이와 같은 현상은 계속 발생하여 소거연산의 횟수가 증가한다. 하지만 복사블록기법은 사용자에게 보이는 메모리의 용량이 실제 메모리의 반 밖에 되지 않는 문제점이 있어 로그블록보다 좋은 방법이라 보기 어렵다. 실험에서 사용한 복사블록기법과 로그블록기법 이외의 FTL 알고리즘도 적용시켜 실험할 수 있다. FTL 알고리즘의 성능 측정뿐만 아니라 FlaSim을 통해 얻은 통계적 함수 수치를 가지고 특정 플래시 메모리의 데이터시트에서 제공하는 각 연산속도와 계산을 통해 처리율(throughput) 성능 측정에도 사용할 수가 있다. 따라서 같은 FTL 알고리즘을 적용시킨 뒤 플래시 메모리 종류마다 처리율의 성능 측정을 할 수 있다.

5. 결론 및 향후 과제

FlaSim은 다양한 FTL 알고리즘을 적용할 때, 필요한 경우 언제라도 주요 API를 사용하여 모듈로 적재가 가능하다. 적재와 제거가 매우 편리하고 볼륨 옵션을 사용하여 다양한 조건에서 플래시 메모리와 관련된 성능 평가 실험을 할 수 있다. FlaSim을 사용함으로써 실험의 확장성이 향상되며 기존의 플래시 메모리에 대한 실험의 제약을 없애고 다양한 성능평가가 가능하다. Oracle 데이터베이스를 사용한 플래시 메모리에 대한 성능평가 실험에서도 쓰기 연산뿐만 아니라 읽기, 삭제연산에 대해서 평가가 가능하고 소스를 수정해야하는 MySQL에서 추출한 트레이스의 정확성이 떨어지는 문제점을 해결하여 정확한 성능평가가 가능하다. 또한 proc 파일 시스템을 이용하기 때문에 실험 결과 분석이 쉽고 간편한 장점이 있다. FlaSim은 리눅스에서 디스크와 동일하게 사용할 수 있으므로 벤치마크를 이용한 성능 측정뿐만 아니라 Oracle, MySQL, SQLite와 같은 다양한 데이터베이스를 적용한 FTL 알고리즘의 실험이 가능하다. 실험과 관련된 다양한 모듈들의 적용이 쉽고 간단하기 때문에 향후 다양한 FTL과 플래시 메모리 실험에 많은 도움이 될 것이다.

참 고 문 헌

[1] S. W. Park, "Flash memory and Database," *Journal of KIISE*, vol.25, no.6, pp.40-47, June. 2007. (in Korean)

[2] W. Lee, B. K. Moon, "Design of Flash-Based DBMS: An In-Page Logging Approach" *ACM SIGMOD International Conference on Management of Data : Database technology for novel applications*, pp.55-66, June. 2007. Beijing, China.

[3] W. J. Park, S. H. Park, S. W. Park, "Performance Analysis of Flash Translation Layer Algorithms for Windows-based Flash Memory Storage Device," *Journal of KIISE : Computing Practices and Letters*, vol.13, no.4, pp.213-225, Aug. 2007. (in Korean)

[4] H. Park, J. Y. Jang, Y. J. Seo, W. J. Park, S. W. Park, "Performance Analysis of Flash Translation Layer using TPC-C Benchmark," *Journal of KIISE : Computing Practices and Letters*, vol.14, no.2, Apr. 2008. (in Korean)

[5] C. Jim, "NAND Flash 101: An Introduction to NAND Flash and How to Design It In to Your Next Product," Micron Technology Inc., Technical Note, TN-29-19, Nov. 2006.

[6] A. Ban "Flash file system optimized for page-mode flash technologies," United States Patent, patent no.5937425, August 10. 1999.

[7] J. S. Kim, J. M. Kim, S. H. Noh, S. L. Min, Y. K. Cho, "A space-efficient flash translation layer for Compact Flash systems," *IEEE Transactions on Consumer Electronics*, vol.48, no.2, May. 2002.