대용량 플래시 메모리를 위한 효율적인 플래시 변환 계층 시스템 소프트웨어

정 태 선† · 박 동 주†† · 조 세 형†††

요 약

플래시 메모리는 비휘발성(non-volatility), 빠른 접근 속도, 저전력 소비, 그리고 간편한 휴대성 등의 장점을 가지므로 최근에 다양한 임베디드 시스템에서 많이 사용되고 있다. 그런데 플래시 메모리는 그 하드웨어 특성상 플래시 변환 계층(FTL: Flash Translation ayer)이라는 시스템 소프트웨어를 필요로 한다. 본 논문에서는 LSTAFF(Large Sate Transition Applied Fast Flash Translation Layer)라 명명된 대블록 플래시 메모리를 위한 새로운 FTL 알고리즘을 제안한다. LSTAFF는 운영체제가 다루는 데이터 섹터 크기 보다 큰 플래시 메모리의 페이지를 고려한 FTL 알고리즘이며, 기존 FTL 알고리즘과 제안될 LSTAFF를 구현하여 플래시 시뮬레이터를 이용하여 성능을 비교하였다.

키워드 : 플래시 메모리, 임베디드 시스템, 파일 시스템

An Efficient System Software of Flash Translation Layer for Large Block Flash Memory

Tae-Sun Chung† · Dong-Joo Park†† · Sehyeong Cho†††

ABSTRACT

Recently, flash memory is widely used in various embedded applications since it has many advantages in terms of non-volatility, fast access speed, shock resistance, and low power consumption. However, it requires a software layer called FTL(Flash Translation Layer) due to its hardware characteristics. We present a new FTL algorithm named LSTAFF(Large State Transition Applied Fast Flash Translation Layer) which is designed for large block flash memory. The presented LSTAFF is adjusted to flash memory with pages which are larger than operating system data sector sizes and we provide performance results based on our implementation of LSTAFF and previous FTL algorithms using a flash simulator.

Key Words : Flash Memory, Embedded System, File System

1. 서 론

플래시 메모리는 비휘발성 (non-volatility), 빠른 접근 속도, 저전력 소비, 그리고 간편한 휴대성 등의 장점을 가지므로 최근에 다양한 임베디드 시스템에서 많이 사용되고 있다. 그런데, 기존의 메모리와는 달리 반도체 기반의 플래시 메모리는 그 하드웨어적 성질 때문에 올바르게 사용되기 위해서는 소프트웨어적 해법을 필요로 한다. 플래시 메모리의 주요 하드웨어적 특징은 "쓰기 전 지우기" 성질이다. 즉, 플래시 메모리에서 데이터 갱신 연산을 수행하기 위해서는 반드시 그 데이터를 포함한 영역이 지워져 있어야 하는 성질을 가진다. 따라서 플래시 메모리 기반의 시스템은 이와 같은 플래시 메모리의 본질적인 하드웨어적인 한계점을 극복하기 위하여 시스템 소프트웨어가 필요한데 이 소프트웨어를 플래시 변환 계층(FTL : Flash Translation Layer)이라고 한다[1-7]. 기존에 제안된 FTL 알고리즘은 플래시 메모리의 물리 페이지의 크기가 운영체제의 데이터 섹터 크기와 같은 플래시 메모리(이후에 이와 같은 플래시 메모리를 소블록 플래시 메모리라고 한다)에 맞추어서 디자인된 알고리즘이다.

그런데, 최근에 주요 플래시 메모리 생산 회사는 대블록 플래시 메모리를 생산하고 있다. 대블록 플래시 메모리는 읽고 쓰는 기본 단위가 운영체제의 데이터 섹터 크기보다 큰 특징을 가진다. 즉, 대블록 플래시 메모리는 데이터를 읽거나 쓸때 그 기본 단위가 2KB 혹은 1KB임에 비하여 대부분의 운영체제에서 데이터 섹터 크기는 512B이다. 따라서 대블록 플래시 메모리를 위한 새로운 FTL 알고리즘이 필요

※ 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.
† 정 회 원 : 아주대학교 정보및컴퓨터공학부 조교수
†† 정 회 원 : 숭실대학교 컴퓨터학부 전임강사
††† 정 회 원 : 명지대학교 컴퓨터소프트웨어학과 부교수
논문접수 : 2005년 2월 19일, 심사완료 : 2005년 10월 14일

하며, 이에 본 논문에서는 LSTAFF(Large State Transition Applied Fast Flash Translation Layer)라 불리는 FTL 알고리즘을 제안한다. LSTAFF는 소블록 플래시 메모리를 위해 제안된 STAFF[3]의 대블록 플래시 메모리로의 확장이다.

논문의 구성은 다음과 같다. 2장에서 문제 정의와 기존 연구를 다루고, 3장에서 LSTAFF 알고리즘을 다루고, 4장에서 성능 평가 결과를 보인다. 마지막으로 5장에서 결론을 맺는다.

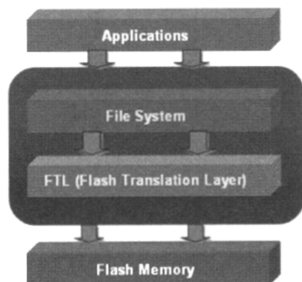
2 기존 연구

본 논문에서의 대블록 플래시 메모리 관련 용어의 의미는 다음과 같다.

- 섹터 : 파일 시스템에 의하여 읽거나 쓰여지는 가장 작은 데이터의 단위이다.
- 페이지 : 페이지는 플래시 메모리에 의하여 읽거나 쓰여지는 가장 작은 단위이다.
- 블록 : 플래시 메모리의 삭제 단위이다. 블록의 크기는 페이지 크기의 배수이다.

(그림 1)은 플래시 기반 임베디드 시스템의 메모리 기본 구조를 나타낸다. 가장 상위 레벨에 응용 프로그램이 있으며, 다음으로 파일 시스템, 플래시 메모리를 위한 디바이스 드라이버, 마지막으로 플래시 메모리가 존재한다. 파일 시스템은 플래시 메모리에 있는 데이터를 읽거나 플래시 메모리에 데이터를 저장하기 위하여 논리 섹터에 대한 읽기 혹은 쓰기 요청을 연속적으로 하게되며, FTL 알고리즘에 의하여 이러한 논리 주소는 실제 플래시 메모리의 섹터 주소로 변경된다.

따라서 대블록 플래시 메모리에서의 FTL 문제를 정형화하면 다음과 같다. 먼저 플래시 메모리는 l개의 물리 페이지로 구성되고 각 페이지는 m개의 물리 섹터로 구성된다고 하고, 파일 시스템은 플래시 메모리를 n 개의 논리 섹터로 간주한다고 하자. 이때 n은 l*m보다 작거나 같다. 즉, 플래시 메모리는 여러 개의 블록으로 구성되고, 각 블록은 여러 개의 페이지로 구성되며, 각 페이지는 여러 개의 섹터로 구성된다. 이때 플래시 메모리는 다음의 성질을 가진다. 플래시 메모리의 물리 페이지 안에 섹터 주소가 이미 데이터가 쓰여져 있으면 새로운 데이터가 그 위치에 다시 쓰여지기 위해서는 그 물리 섹터를 포함하는 블록이 지워져야만 한다.



(그림 1) 플래시 메모리 시스템의 아키텍쳐

대블록에서의 FTL 알고리즘은 파일 시스템이 보내는 논리 섹터 번호에 대하여 플래시 메모리의 물리 페이지 안의 물리 섹터 오프셋을 생성하는 알고리즘이다.

대블록 플래시 메모리는 최근에 생산되었기 때문에 대블록 플래시 메모리에 대한 FTL 알고리즘에 대한 연구는 현재까지 미미한 상황이다. 소블록 플래시 메모리의 경우에는 기존 연구를 섹터 사상, 블록 사상, 그리고 혼합 사상으로 분류할 수 있다. 이 밖에 플래시 메모리를 위한 파일시스템을 새로이 구현한 연구[8-10]도 있으나 FAT 호환성 문제 때문에 실제 상용 제품에 적용되기 힘든 상황이다.

2.1 섹터 사상

섹터 사상 기법은 읽기 쓰기 단위인 섹터 단위로 논리 물리 사상 테이블이 존재하는 방법이다. 즉, 파일 시스템 관점에서 m개의 논리 섹터가 존재 한다고 하면 논리-물리 사상 테이블의 행의 크기가 m이 된다[1]. 만일 같은 논리 섹터에 대한 갱신 연산이 요청되면 사상 테이블의 정보만 변경한 후 플래시 메모리의 비어있는 물리 주소에 데이터를 쓸 수 있으므로 삭제 연산을 최소화 할 수 있다.

2.2 블록 사상

블록 사상 기법은 섹터 사상 기법이 사상 테이블의 크기가 커지는 단점을 극복하기 위하여 제안된 방법으로 플래시 메모리의 삭제 단위인 블록 단위로 논리 물리 사상 테이블이 존재하는 방법이다. 만일 m개의 논리 블록이 파일 시스템에 의하여 보여지면 사상 테이블의 행 사이즈는 m이 된다[2, 4, 6]. 블록 사상 기법은 사상 테이블의 크기가 현격히 줄어 들었지만 블록 단위로 사상 정보를 일치시켜야 하기 때문에 같은 논리 주소에 쓰기 연산이 많이 요청되는 경우 성능 저하를 발생시킨다.

2.3 혼합 사상

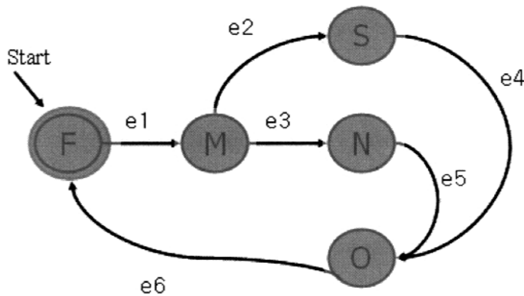
혼합 사상 기법은 섹터 사상 기법과 블록 사상 기법을 혼합한 기법이다. 즉, 일단 블록 사상 기법과 같이 블록 단위의 사상 테이블이 존재하고, 블록 내에서는 섹터 사상을 하는 방법이다[5, 7]. 플래시 메모리로부터 데이터를 읽을 때는 먼저 해당 논리 블록에 대한 물리 블록을 사상 테이블로부터 구한다. 다음으로 그 블록에 존재하는 논리 섹터 번호를 탐색함으로써 해당 데이타를 읽을 수 있다.

3. LSTAFF(Large STAFF)

LSTAFF는 대블록 플래시 메모리를 위한 FTL 알고리즘이다. LSTAFF는 STAFF 알고리즘[3]을 확장하여 대블록 플래시 메모리에 적용하였다.

3.1 LSTAFF 알고리즘

LSTAFF는 플래시 메모리의 블록에 다음과 같이 상태를 정의한다.



(그림 2) 블록 상태 오토마타

- F 상태 블록: F 상태 블록은 그 블록이 삭제되어 데이타가 한번도 쓰여 지지 않은 블록을 의미한다.
- O 상태 블록: O 상태 블록은 블록에 있는 데이터가 더 이상 유효하지 않음을 나타낸다.
- M 상태 블록: M 상태 블록은 논리 섹터의 오프셋과 물리 섹터의 오프셋이 같은 블록으로 그 블록의 일부는 데이터가 쓰여져 있고, 나머지는 비어있는 블록이다.
- S 상태 블록: M 상태 블록과 같이 논리 섹터의 오프셋과 물리 섹터의 오프셋이 같은 블록이면서 블록의 모든 섹터에 데이터가 쓰여져 있는 블록이다. S 블록은 교체 합병 (Swap Merging) 연산에 의하여 M 블록에서 전이된다.
- N 상태 블록: N 상태 블록은 논리 섹터의 오프셋과 물리 섹터의 오프셋이 다를 수 있는 블록으로 M 상태 블록에서 전이된다.

본 논문에서는 앞서 정의한 플래시 메모리 블록의 상태와 FTL 연산에 대하여 다음과 같이 오토마타를 정의한다. 오토마타 관련 기호는 [11]을 사용한다. 오토마타는 (Q,Σ,δ,q_0,F)으로 표현되고 각 기호의 의미는 다음과 같다.

- Q는 블록 상태의 집합으로 Q = {F, O, M, S, N}가 된다.
- Σ은 오토마타의 입력 기호로 본 논문에서는 FTL 연산 중의 여러 이벤트에 대응된다.
- δ는 전이 함수로 $Q\times\Sigma$에서 Q로의 함수이다.
- q_0는 시작 상태로 F 블록에서 시작한다.
- F는 종료 상태의 집합이다.

(그림 2)는 블록 상태 오토마타를 나타낸다. 시작 상태는 F 블록 상태가 되는데 첫번째 쓰기 요청에 의하여 그 F 블록은 M 블록으로 전이된다. M 블록은 FTL 연산에 의하여 S블록이나 N블록으로 전이된다. S블록과 N블록은 이벤트 e4 와 e5에 의하여 O블록으로 전이되고, O블록은 이벤트 e6에 의하여 F 상태로 변경된다. 구체적인 이벤트는 [3]을 참조하기 바란다.

3.2 대블록 확장

대블록 플래시 메모리에서는 한 페이지가 하나 이상의 섹터로 구성되므로 보다 복잡한 논리 물리 사상 방법을 고려해야 한다. 즉, 소블록 플래시 메모리의 경우에는 사상의 레벨이 크게 섹터 사상과 블록 사상이 재하지만 대블록 플래시 메모리의 경우에는 사상 레벨이 섹터, 페이지, 블록이 존재한다.

FTL 문제를 풀기 위해서 입력 논리 섹터 번호에 대하여 논리 페이지 번호, 논리 블록 번호, 그리고 논리 섹터 오프셋을 구하여야 한다. 논리 섹터 번호 (lsn)가 주어지면 논리 블록 번호 (lbn), 논리 페이지 번호 (lpn), 논리 섹터 오프셋 (lso)은 다음과 같이 계산된다. 단, np는 블록 당 페이지 수이고, ns는 한 블록 안의 한 페이지 안의 섹터 수이다.

$$lbn = lsn/(np * ns) \quad (1)$$

$$tmp = lsn\%(np * ns) \quad (2)$$

$$lpn = tmp/ns \quad (3)$$

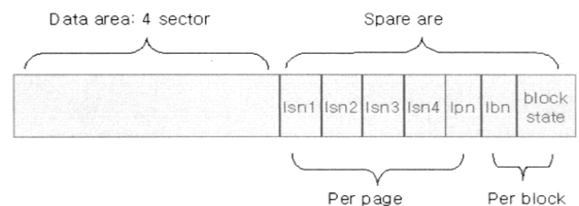
$$lso = tmp\%ns \quad (4)$$

예를 들어 플래시 메모리가 1024 블록으로 구성되고, 각 블록은 64 페이지, 각 페이지는 4 섹터로 이루어진다고 하면 플래시 메모리의 용량은 128MB가 된다. 논리 섹터 번호가 101이라면 lbn은 0, lpn은 25, 그리고 lso는 1이 된다.

사상 정보를 플래시 메모리에 저장하기 위하여 페이지 포멧을 (그림 3)과 같이 디자인하였으며, (그림 3)에서 한 블록의 한 페이지는 네 개의 섹터로 구성된다고 가정한다. 앞서 언급한대로 대블록 플래시 메모리의 논리 물리 사상에는 섹터, 페이지, 블록의 세 단계의 사상이 존재하기 때문에, 논리 섹터 번호 (lsn1, lsn2, lsn3, lsn4), 논리 페이지 번호 (lpn), 논리 블록 번호가 플래시 메모리의 각 페이지에 존재하여 주로 메타

정보를 저장하는 보조 영역 (spare area)에 저장된다. 여기서 lsn 과 lpn은 각 페이지 당 저장되어야 하지만, lbn은 블록 당 하나의 페이지에만 저장되면 된다. 이 보조 영역에 저장된 사상 정보를 스캔함으로써 논리 물리 사상 테이블이 생성되어 RAM에 저장될 수 있다. 블록에 대한 상태도 블록 당 한 곳에 저장될 수 있다.

LSTAFF에서 F, S, O와 M 블록은 논리 페이지 (섹터) 오프셋과 물리 페이지 (섹터) 오프셋이 일치하기 때문에 lsn과 lpn을 필요로하지 않는다. N 블록 만이 lsn과 lpn을 필요로 한다. N 블록에 쓰기 연산을 하는 방법에는 다음과 같은 선택 옵션이 존재한다.



(그림 3) 페이지 포멧

physical page number	data1	data2	data3	data4	lsn1	lsn2	lsn3	lsn4	lpn
0	D0	D1	D2	D3					0
1	D0	D1	D2	D3					0
2	D8	D9							2
3		D9							2

physical page number	data1	data2	data3	data4	lsn1	lsn2	lsn3	lsn4	lpn
0	D0	D1	D2	D3	0	1	2	3	
1	D0	D2	D2	D3	0	1	2	3	
2	D8	D9	D9		8	9	9		
3									

lsn sequence: 0,1,2,3,0,1,2,3,8,9,9

(그림 4) 사상 예제

- 페이지 사상 : (그림 3)의 lpn을 이용하여 논리 페이지 번호를 물리 페이지 번호에 사상할 수 있다. 이 경우 논리 섹터 번호 (lsn1, lsn2, lsn3, lsn4)는 플래시 메모리에 쓸 필요가 없다.
- 섹터 사상 : (그림 3)의 lsn을 이용하여 논리 섹터 번호를 물리 섹터 번호에 사상할 수 있다. 이 경우 lpn은 플래시 메모리에 쓸 필요가 없다.
- 1:n 사상 : 위의 페이지와 섹터 사상에서 한 논리 페이지 (섹터)가 둘 이상의 물리 페이지 (섹터)에 사상될 수 있다.

예제 1 (그림 4)는 페이지와 섹터 사상의 예제를 보인다. 입력된 연속된 lsn이 (0,1,2,3,0,1,2,3,8,9,9)라고 하면 (그림 4)의 윗부분은 페이지 사상을 보이고, (그림 4)의 아랫 부분은 섹터 사상을 보인다. 만일 1:1 페이지 사상을 가정하면 (그림 4)의 윗부분의 데이터 D8은 물리 페이지 번호 3번에 복사된다. 플래시 메모리에서 읽을 때는 가장 최근에 쓰여진 데이터가 유효한 데이터이다. 예를 들어 (그림 4)의 윗 부분에서 lsn 9에 대한 요효한 데이터는 물리 페이지 번호 3에 있는 데이터이다.

4. 성능 평가

4.1 비용 함수

LSTAFF의 비용 함수는 플래시 메모리의 입출력의 경우 대블록 플래시 메모리나 소블록 플래시 메모리 모두 페이지 단위의 입출력을 수행하므로 STAFF에서의 비용 함수와 동일하다. 단, 대블록 플래시 메모리에서는 페이지의 하나 이상의 섹터가 동시에 읽거나 쓰여질 수 있으므로 전체 시스템 성능은 LSTAFF가 STAFF보다 우수하다.

플래시 I/O 성능에 있어서는 읽기 쓰기 비용은 다음 식으로 계산될 수 있다.

$$C_{read} = p_M T_r + p_N k_1 T_r + p_S T_r \ (where\ p_M + p_N + p_S = 1) \quad (5)$$

$$\begin{aligned} C_{write} = {} & p_{first}[(T_f + T_w)] + (1 - p_{first})[p_{merge}\{T_m + \\ & p_{e_1} T_w + (1 - p_{e_1})(k_2 T_r + T_w)\} + \\ & (1 - p_{merge})\{p_{e_2}(T_r + T_w) + (1 - p_{e_2})(k_3 T_r + T_w) + \\ & T_r + p_{MN} T_w\}] \end{aligned} \quad (6)$$

여기서 $1 \le k_1, k_2, k_3 \le n$ 을 만족하며, n은 한 록에 존재하는 페이지 수를 의미한다. 식 (5)에서 p_M, p_N, p_S 는 각각 데이터가 M, N, S 블록에 저장될 확률을 의미한다.

식 (6)에서 1) p_{first} 는 입력 논리 블록에 대하여 쓰기 연산이 첫번째로 일어났을 확률을 2) p_{merge} 는 쓰기 연산이 합병 연산을 필요로 할 확률을 3) p_{e_1} 과 4) p_{e_2} 는 각각 합병 연산을 동반하고 혹은 동반하지 않고 입력 논리 섹터 번호에 대한 쓰기 연산이 플래시 메모리의 섹터 오프셋이 일치하는 위치에 쓰여질 확률을 5) T_f 는 F 블록을 할당하기 위해 필요한 비용을 6) p_{MN} 은 쓰기 연산이 M 블록을 N 블록으로 전이시킬 확률을 의미한다. 쓰기 연산이 M 블록을 N 블록으로 전이시킬 때는 상태를 표시하기 위하여 한번의 플래시 쓰기 연산이 더 필요하게 된다.

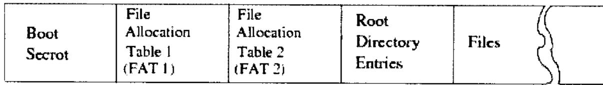
비용 함수는 M이나 S 블록에 대한 읽기 쓰기 연산 보다 N 블록에 대한 읽기 쓰기 연산이 더 많은 읽기 연산을 필요로 함을 알 수 있다. 하지만 플래시 메모리에 대한 읽기 연산은 쓰기나 지우기 연산에 비하여 매우 작은 비용이 드는 연산이다. 따라서 T_f 나 T_m 이 플래시 삭제 연산을 필요로 할 수 있기 때문에 전체 시스템 성능을 결정하는 주요한 요소가 된다. LSTAFF는 삭제 연산을 야기하는 T_f 나 T_m 의 값을 최소로 할 수 있도록 디자인되었다.

4.2 실험 결과

4.2.1 시뮬레이션 환경

(그림 1)의 플래시 시스템 아키텍쳐에서 LSTAFF 알고리즘과 2장에서 소개된 블록 사상 알고리즘을 구현하였다. 블록 사상 알고리즘에서 하나의 논리 블록은 두개의 물리 블록에 사상될 수 있도록 하였는데 이 방법이 좀더 효율적인 방법으로 LSTAFF 방법과 보다 공정한 비교를 하기 위해서이다. (그림 1)의 플래시 메모리 단계는 실제 플래시 메모리와 같은 성질을 가지는 플래시 에뮬레이터를 구현하여 실험하였다.

(그림 1)에서 파일시스템 단계는 임베디드 시스템에서 많이 사용되는 FAT 파일 시스템 [12]을 가정한다. (그림 5)는 FAT 파일 시스템의 디스크 포멧을 보인다. 먼저 부트 섹터와 하나 혹은 그 이상의 파일 할당 테이블, 루트 디렉토리, 그리고 볼륨 파일이 있다. 여기서 부트 섹터와, 파일 할당 테이블, 루트 디렉토리가 볼륨 파일에 비하여 자주 접근되는 것을 예측할 수 있다. 본 논문에서는 Symbian [13] 운영체제가 1MB의 파일 쓰기 요청에 대하여 블록 디바이스 드라이버 단으로 보내는 접근 패턴을 실험에 이용하였다.이 접근 패턴은 실제 임베디드 응용의 패턴과 유사하다.



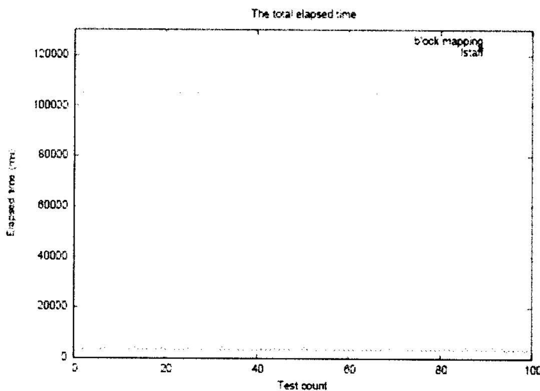
(그림 5) FAT 파일 시스템

4.2.2 결과

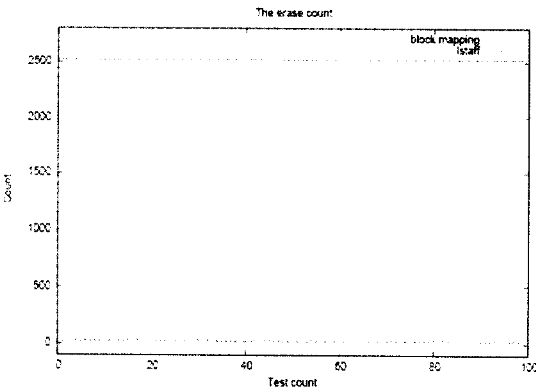
(그림 6)-(a)는 총 수행 시간을 보인다. X축은 실험한 횟수를 의미하고, y축은 총 수행시간을 나타낸다. 처음에 플래시 메모리는 비어 있으며 실험 횟수가 증가할수록 플래시 메모리에 데이터가 쌓이게 된다. 실험 결과는 LSTAFF가 블록 사상에 비하여 훨씬 좋은 성능을 보임을 알 수 있다. 그 이유는 LSTAFF는 상태 블록 개념을 이용하여 갱신 연산이 일어나더라도 성능 저하가 최소화되기 때문이다.

(그림 6)-(b)는 삭제 횟수를 나타낸다. 삭제 횟수는 총 수행 시간과 비슷한 결과를 나타낸다. 이것은 삭제 횟수가 전체 시스템 성능의 주요 요소가 되기 때문이다. [14]에 의하면 읽기 (1 페이지), 쓰기 (1 페이지), 지우기 (1 블록) 연산의 시간 비율이 약 1:4:20이다. 또한 LSTAFF는 플래시 메모리가 거의 다 차 있을 때도 일관된 성능을 보임을 알 수 있다.

(그림 7)-(a)와 (그림 7)-(b)는 읽기와 쓰기 횟수를 보인다. LSTAFF는 훨씬 적은 읽기와 쓰기 연산을 필요로 함을 알 수 있다. 이것은 블록 사상에서 병합 연산이 많이 일어나기 때문이다.

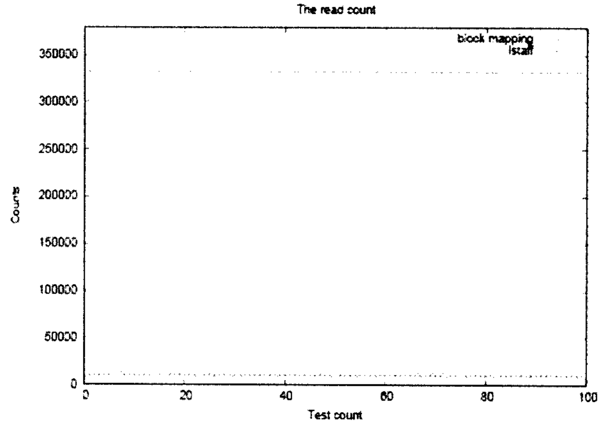


(a) 총 수행 시간

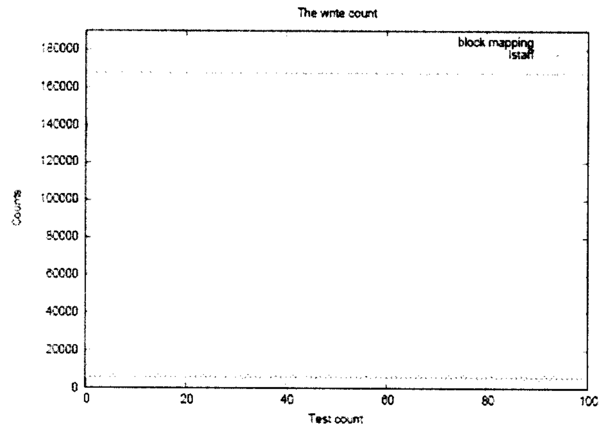


(b) 삭제 연산 수

(그림 6) 총 수행 시간 및 삭제 연산 수



(a) 읽기 연산 수



(b) 쓰기 연산 수

(그림 7) 읽기와 쓰기 연산 횟수

시뮬레이션 결과를 표로 요약하면 <표 1>과 같다. LSTAFF 알고리즘은 블록 사상 기법에 비하여 읽기, 쓰기, 삭제 연산을 각각 3.15%, 3.61%, 1.31%만을 수행하여 수행 시간이 3.33%만 소요됨을 알 수 있다.

〈표 1〉 시뮬레이션 요약

	읽기	쓰기	삭제	총수행시간 (ms)
블록 사상	332152	167340	2518	105187
LSTAFF	10470	6037	33	3506
	3.15%	3.61%	1.31%	3.33%

5. 결 론

본 논문에서 LSTAFF라 불리는 대블록 플래시 메모리를 위한 FTL 알고리즘을 제안하였다. LSTAFF는 파일 시스템의 데이터 섹터 크기에 비하여 큰 페이지를 가지는 대블록 플래시 메모리에 대하여 가장 좋은 성능을 내도록 디자인되었다. LSTAFF는 STAFF와 같이 플래시 메모리의 삭제 단위인 블록에 대하여 상태 전이 개념을 도입하여 입력 패턴에 따라서 블록의 상태가 전이됨으로써 삭제 연산을 최소화한다. 또한, 대블록 플래시 메모리에 데이터를 저장할 때 필

요한 몇가지 휴리스틱을 제안하였다.시뮬레이션 결과에 의하면 LSTAFF는 블록 사상에 비하여 총 수행 시간에 있어서 블록 사상의 3.33%만이 소요되었다.

참 고 문 헌

[1] Amir Ban, "Flash file system," United States Patent, No.5,404,485, 1995.

[2] Amir Ban, "Flash file system optimized for page-mode flash technologies," United States Patent, No.5,937,425, 1999.

[3] Tae-Sun Chung, Stein Park, Myung-Jun Jung, and Bumsoo Kim, "STAFF: State Transition Applied Fast Flash Translation Layer," In 17th International Conference on Architecture of Computing Systems, 2004.

[4] Petro Estakhri and Berhanu Iman, "Moving sequential sectors within a block of information in a flash memory mass storage architecture," United States Patent, No.5,930,815, 1999.

[5] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho, "A space-efficient flash translation layer for compactflash systems," IEEE Transactions on Consumer Electronics, Vol.48, No.2, pp.366-375, 2002.

[6] Takayuki Shinohara, "Flash memory card with block memory address arrangement," United States Patent, No.5,905,993, 1999.

[7] Bum soo Kim and Gui young Lee, "Method of driving remapping in flash memory and flash memory architecture suitable therefore," United States Patent, No.6,381,176, 2002.

[8] M. Resenblum and J. Ousterhout, "The Design and Implementation of a Log-structured File System," ACM Transactions on Computer Systems, Vol.10, No.1, pp.26-52, 1992.

[9] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," In International Conference on Architectural Support for Programming Languages and Operating Systems, 1994.

[10] A. Kawaguchi, S. Nishioka, and H. Motoda, "Flash Memory based File System," In USENIX 1995 Winter Technical Conference, 1995.

[11] John E. Hopcroft and Jeffrey D. Ullman, "Introduction to automata theory, languages, and computation," Addison-Wesley Publishing Company, 1979.

[12] Microsoft Corporation, "Fat32 file system specification," Technical report, Microsoft Corporation, 2000.

[13] Symbian, http://www.symbian.com, 2003.

[14] Samsung Electronics, "Nand flash memory & smartmedia data book," 2004.

정 태 선

e-mail : tschung@ajou.ac.kr

1995년 KAIST 전산학과(학사)
1997년 서울대학교 전산과학과(석사)
2002년 서울대학교 전기컴퓨터공학부(박사)
2002년～2004년 삼성전자 소프트웨어센터 책임연구원
2004년 3월～2005년 8월 명지대학교 컴퓨터소프트웨어학과 조교수
2005년 9월～현재 아주대학교 정보및컴퓨터공학부 조교수
관심분야 : 플래시 메모리, 데이터베이스 등

박 동 주

e-mail : djpark@ssu.ac.kr

1995년 서울대학교 컴퓨터공학과(학사)
1997년 서울대학교 컴퓨터공학과 (공학석사)
2001년 서울대학교 컴퓨터공학부 (공학박사)
2001년～2003년 삼성전자 책임연구원
2004년～현재 숭실대학교 컴퓨터학부 전임강사
관심분야 : 플래시 메모리, 내장형 소프트웨어, 멀티미디어 데이터베이스 등

조 세 형

e-mail : shcho@mju.ac.kr

1981년 서울대학교(공학사)
1983년 서울대학교 계산통계학(이학석사)
1992년 펜실베니아 주립대학 전산학(박사)
1983년 금성통신(주)
1984년～1999년 한국전자통신연구원 책임연구원
2000년～현재 명지대학교 컴퓨터소프트웨어학과 부교수
관심분야 : 인공지능, 온톨로지 및 시맨틱웹, 자연어처리