6

플래시메모리와 데이터베이스

한국외국어대학교 ▌박 상 원*

1. 서 론

최근 디지털 카메라, MP3 플레이어, 핸드폰과 같은 이동형 기기가 많이 등장하고 있다. 이러한 이동형 기기는 충격에 강하고 크기가 작으면서 전력 소모량이 적은 기억 장치가 필요하다. 플래시 메모리는 이러한 요구를 충족시킬 수 있는 좋은 소자이다. 특히 전원이 차단되어도 내용을 잃어버리지 않는 특징이 있어 디스크를 대체할 수 있는 소자로 각광받고 있다. 그 외에도 부피가 작으면서 용량이 크고, 전력 소모가 작으면서 속도가 빠르고, 충격에 강해야 하는 요구사항을 만족하는 휴대용 저장 장치로 플래시 메모리가 각광받고 있다. 매년 플래시 메모리의 용량이 두 배 이상 증가하고 있으므로, 많은 분야에서 플래시메모리는 점차 디스크를 대체하여 대표적인 대용량 저장장치로도 자리매김을 할 것이다.

플래시 메모리는 낸드(NAND) 플래시 메모리와 노어(NOR) 플래시 메모리로 나누어진다. 표 1에서 보는 바와 같이 노어 플래시 메모리는 저장된 데이터를 읽을 때 바이트 단위로 어드레싱이 가능하다. 그러므로 노어 플래시 메모리는 프로그램을 적재하여 실행할 수 있는 특징이 있어 휴대전화에서 많이 사용된다. 하지만 모든 소자를 어드레싱해야 하기 때문에 수율이 낮아 비트당 가격이 비싼 단점이 있다. 낸드 플래시 메모리는 디스크와 같이 섹터 단위로 어드레싱을 한다. 그러므로 플래시 메모리에서 직접 프로그램을 실행하지는 못하고 이를 DRAM으로 옮겨서 실행해야 한다. 하지만 한 비트를 저장하기 위한 셀(cell)의 크기가 작고 특정 블록에 문제가 있더라도 그 블록을 배드 블록(bad block)으로 처리한 후 사용하면 되므로 수율이 높아 싼 가격으로 고집적화를 할 수 있어 대용량 메모리를 만들기가 쉬운 장점이 있다. 요즈음은 낸드 플래시의 장점과 노어 플래시의 장점을 두루 갖춘 OneNAND와 같은 제품도 등장하고 있다. 이것은 내부에 낸드 플래시와 DRAM을 두어 대용량의 장점과 바이트 어드레싱의 장점을 모두 취할 수 있는 메모리이다. 데이터베이스는 대용량 데이터를 저장해야 하므로 낸드 플래시 메모리에 기록한다고 가정하고 본 논문에서는 이에 대해서만 논하기로 한다.

2. 플래시 메모리의 구조

플래시 메모리는 그림 1에서 보는 바와 같이 블록, 페이지, 섹터로 구성되어 있다. 플래시 메모리에서 데이터를 읽고 쓰는 단위는 페이지이다. 섹터는 보통 512 바이트로 구성되어 있으며 각 섹터의 메터 데이터를 기록하기 위한 여유 영역(spare area)가 있다. 메터 데

표 1 플래시 메모리의 특징

	NOR 플래시 메모리	NAND 플래시 메모리
장점	바이트 단위 어드레싱 빠른 읽기	블록 단위 어드레싱 상대적으로 빠른 소거, 쓰기 성능 작은 셀(cell) 크기 높은 수율(배드 블록(bad block) 처리)
단점	상대적으로 느린 소거, 쓰기 연산	느린 임의 접근
응용	부트 이미지, BIOS 휴대전화	SSD(solid state disk) 대용량 저장 용도

† 본 연구는 정보통신부 및 정보통신연구진흥원의 IT신성장동력 핵심기술개발사업의 일환으로 수행하였음.
[2006-S-040-01, Flash Memory 기반 임베디드 멀티미디어 소프트웨어 기술 개발]

* 종신회원

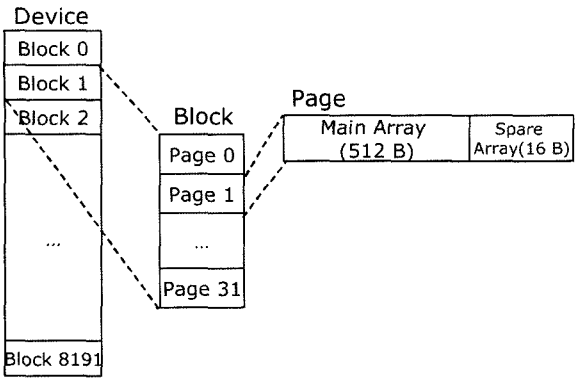

그림 1 소블록 (128 Mbytes)

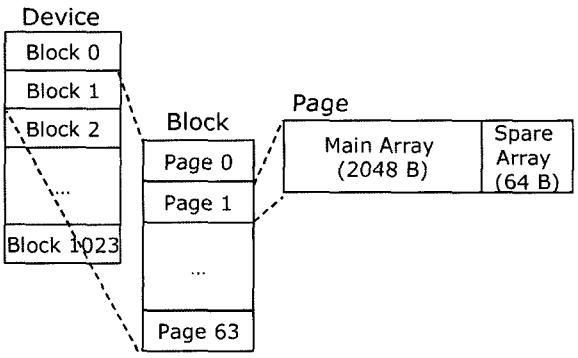

그림 2 대블록(128 Mbytes)

이터에 기록되는 정보로는 그 구조가 크게 두 가지로 구성되어 있다. 그림 1은 소블럭(small block) 구조로서 블록, 페이지, 섹터로 구성되어 있다. 한 섹터는 512 바이트로 구성되며 이러한 섹터 32개가 하나의 블록을 구성하고 있다. 대블럭의 경우는 페이지가 4개의 섹터로 구성되어 있다. 즉, 대블럭의 경우는 4개의 섹터를 동시에 읽거나 쓸 수 있다.

플래시 메모리는 디스크나 램과 다르게 덮어쓰기 연산을 하지 못한다. 예를 들어 특정 블록을 수정하기 위해서는 그 블록을 소거(erase)하고 쓰기 연산을 수행해야 한다. 이것을 쓰기 전 소거(erase-before-write) 연산이라 한다. 그러므로 블록 내의 한 섹터를 수정하기 위해서는 그 블록의 내용을 다른 곳에 복사해 두고, 해당 블록을 소거하여 지운 다음, 수정하고자 하는 섹터 및 회피해 두었던 다른 섹터의 내용을 해당 블록으로 복사해야 한다. 이것은 대단히 비효율적인 연산이라 할 수 있다.

표 2는 플래시 메모리의 연산 속도를 나타낸 것이다. 표에서 보는 바와 같이 읽기 연산에 비해서 쓰기 연산이, 쓰기 연산에 비해서 소거 연산에 걸리는 시간이 아주 크다는 것을 알 수 있다. 만약 특정 섹터를 수정하기 위하여 그 섹터가 속한 블럭을 소거한다면 전체 연산이 아주 비효율적으로 동작할 것이다.

표 2 낸드 플래시 메모리의 연산 속도

	Read	Write	Erase
Samsung K9WAG08U1A 16Gbits SLC NAND	80 μs (2 KB)	200 μs (2 KB)	1.5 ms (128 KB)

따라서 플래시 메모리는 성능 향상을 위하여 새로운 소프트웨어 계층인 FTL(flash translation layer)을 둔다. FTL은 어떠한 섹터에 대해 쓰기를 요구하면 FTL 내부의 매핑테이블을 이용하여 요구한 섹터와 실제 기록하는 섹터를 다른 위치에 쓰도록 한다. 이는 플래시메모리의 성능 저하의 원인인 소거연산이 줄여 성능을 높인다. 또한 플래시메모리를 운영체제에서 보았을 때 일반 디스크와 같은 장치로 볼 수 있게 한다.

3. 본 론

3.1 플래시 메모리와 FTL

그림 2는 플래시 메모리에서 FTL에 대한 것이다. FTL은 파일 시스템으로부터 전달받은 읽기, 쓰기에 대한 명령어와 섹터 번호, 길이를 인자로 받아 필요한 섹터를 반환하는 역할을 한다[1]. 이때 쓰기 연산의 속도를 증진시키기 위하여 물리적인 고정된 위치에 데이터를 기록하지 않고 미사용중인 빈 영역을 찾아 쓰기 연산을 수행한다. 그러므로 파일 시스템에서 요청한 섹터 번호와 실제 데이터가 기록된 물리적인 위치는 서로 다르게 된다. 이 정보를 유지하기 위하여 FTL은 램 테이블로 불리는 매핑 테이블을 가지고 있다. 이때 매핑을 어떻게 하느냐에 따라 FTL의 성능이 달라진다.

3.2 매핑 방법

FTL은 위의 플래시메모리의 특징을 잘 이용하여 플래시메모리의 성능 향상과 수명 연장을 주목적으로

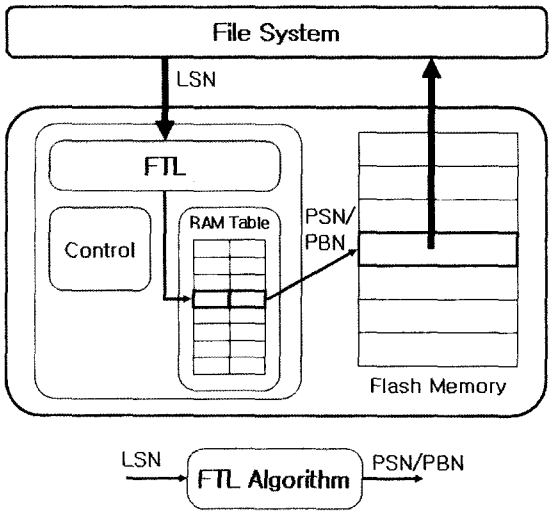

그림 3 FTL

한다. FTL은 주로 램 테이블과 실제 플래시 메모리에 저장된 여러 가지 정보를 이용하여 매핑 테이블을 유지, 관리하는 역할을 주로 수행한다.

FTL알고리즘에서 사용되는 매핑은 섹터매핑, 블록매핑, 하이브리드(hybrid)매핑으로 분류할 수 있다[2]. 섹터매핑은 섹터 단위로 매핑 테이블을 만들어 물리적인 섹터 번호와 논리적인 섹터 번호를 매핑하는 것이고, 블록 매핑은 물리적인 블록 번호와 논리적인 블록 번호를 매핑하는 것이다.

그림 2에서와 같이 섹터 매핑을 하면 한 번의 매핑 테이블의 검색을 통해 실제 플래시메모리의 물리적인 섹터 번호를 알 수 있으므로 매우 빠르게 동작 할 수 있다. 하지만 플래시 메모리의 용량이 증가하면 섹터의 개수는 늘어나게 되어 매핑할 정보가 많아지게 되므로 매핑 테이블을 유지하기 위한 램이 크게 증가하는 문제가 있다.

블록 매핑은 그림 3에서 보는 바와 같이 플래시 메모리의 블록 단위로 매핑 정보를 유지한다. 그리고 블럭 내의 섹터는 오프셋을 이용하여 필요한 섹터를 구하게 된다. 이것은 섹터 매핑에 비하여 상대적으로 적은 수의 램만으로 매핑 정보를 유지할 수 있는 장점이 있다. 하지만 동일 섹터에 대한 덮어 쓰기 연산이 계속적으로 수행하게 되면 블록을 계속적으로 합병해야 하는 부담이 있다.

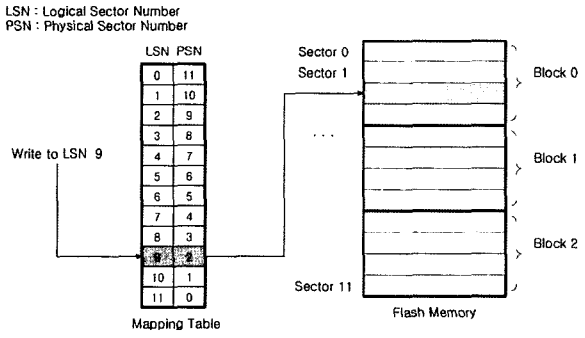

그림 4 섹터 매핑

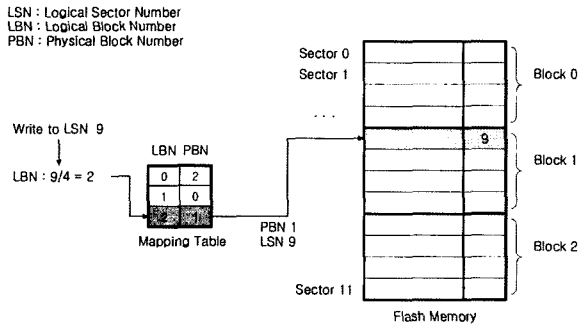

그림 5 블록 매핑

하이브리드 매핑은 블록 매핑을 이용하여 원하는 블록을 찾고 섹터는 블록 내에서 임의의 위치에 둘 수 있게 하는 방법이다. 이때 섹터를 구분하기 위하여 플래시 메모리의 여유 영역(spare area)에 섹터 번호를 기록하여 섹터를 구분할 수 있게 한다. 이것은 블록 매핑에서의 단점인 동일 섹터의 계속적인 쓰기 연산이 발생할 때 합병 연산이 발생하는 것을 블록 내의 다른 섹터에 쓰기 연산을 수행하는 것으로 해결할 수 있다.

플래시 메모리에서 한 개의 블록 내에 있는 섹터들에 대해 기록하는 방법을 두 가지로 나누는데 한 블록에 저장된 섹터들이 정해진 특정 위치에 고정되는 것을 고정 섹터방식(in-place)이라 부르며, 섹터들이 블록 내의 임의의 위치에 존재하는 것을 변동 섹터 방식(out-of-place)라 부른다. 블록 매핑은 고정 섹터 방식을 사용하고, 하이브리드 매핑은 변동 섹터 방식을 사용한다.

3.3 대표적인 FTL 알고리즘

기존의 FTL 알고리즘을 분류하면 크게 여유 영역 기법, 복사 블록 기법과 로그 블록 기법으로 구분할 수 있다.

3.3.1 여유 영역 기법

미쓰비시(Mitsubishi)의 여유 영역 알고리즘[3]은 각 블록이 데이터 영역과 여유 영역으로 나누어 덮어쓰기에 대한 해결책을 제시하고 있다. 이 알고리즘에서는 램 테이블에 모든 블록에 대한 블록 매핑 정보를 가지고 있다. 쓰기 연산은 주어진 논리적 섹터 번호를 이용하여 블록내의 여유영역을 제외한 데이터 영역 만을 논리적인 한 블록으로 보고 논리적 블록 번호를 계산하고 이 값을 이용하여 매핑 테이블에서 물리적 블록 번호를 찾아낸다. 그림 4는 알고리즘이 동작하는 예제를 나타낸 것으로 한 블록에 4개의 섹터

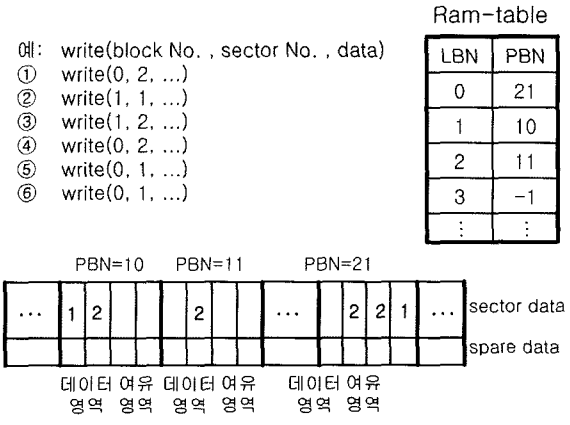

그림 6 여유 영역 기법

가 기록된다고 가정하여 그린 그림이다. 이때 데이터 영역으로는 2개의 섹터가 여유 영역으로 2개의 섹터가 배정되어 있다.

블록 디바이스 드라이버에서 FTL로 쓰기 요청이 들어오면 논리적 블록번호에 대해 해당하는 물리적 블록 번호가 있는 경우와 없는 경우로 나뉜다. 해당하는 물리적 블록 번호가 없는 경우 빈 블록을 찾아내어 논리적 블록 번호와 물리적 블록 번호에 대한 매핑 정보를 테이블에 기록한다. 이 빈 블록에 고정 섹터 방식으로 기록할 섹터의 위치를 찾아서 데이터를 기록한다.

논리적 블록 번호에 해당하는 매핑 정보가 있는 경우 해당되는 블록을 구하여 기록하게 된다. 이때 블록 내의 해당 섹터 위치가 비어 있는 경우에는 데이터를 그 곳에 기록한다. 만약 그 섹터에 이미 데이터가 기록되어 있으면 덮어 쓰기가 발생한다. 이 경우 덮어쓰기를 하지 않고 블록의 여유 영역에 변동 섹터 방식으로 기록하고 데이터 영역의 이전 내용을 사용 불가(invalid) 상태로 기록한다. 덮어 쓰기 발생 시 블록의 여유 영역이 가득 찼을 경우는 합병 연산을 수행한다. 합병 연산은 빈 블록을 할당한 후 기존 블록의 여유 영역의 데이터를 검사하여 가장 최신의 데이터를 찾아 새로 할당받은 블록에 기록하고, 기존 블록의 데이터 영역의 유효한 데이터를 찾아 새로 할당받은 블록에 기록한다. 새로 할당받은 블록에 대한 정보를 매핑 테이블에 기록하고 기존 블록을 소거한다.

이 알고리즘은 여유 영역만큼 실제 플래시 메모리의 블록 수를 늘려야 하기 때문에 논리적인 플래시 메모리의 양과 물리적인 양이 다르다. 즉 사용자에게 보이는 메모리의 크기는 여유 영역을 제외한 크기로 보이므로 성능에 비해 플래시 메모리 사용량이 많다는 문제가 있다.

3.3.2 복사 블록 기법

M-system의 FMAX는 복사 블록(mirror block)[4]을 사용한다. 램 테이블에는 블록매핑을 실시한다. FTL의 쓰기 연산이 호출되면, FTL은 주어진 시작 논리적 섹터 번호를 이용하여 논리적 블록 번호를 계산하고 이 값을 이용하여 매핑 테이블로부터 물리적 블록 번호를 찾아낸다.

해당하는 물리적 블록 번호가 없는 경우, 빈 블록을 할당하여 매핑 테이블에 기록하고 섹터의 여유 영역에 논리 블록 번호를 기록한 후 고정 섹터 방식으로 데이터를 기록한다. 해당하는 블록이 존재하고 해당 섹터가 비어있으면, 해당 섹터에 고정섹터 방식으로 데이터를 쓴다. 해당 섹터에 할당된 블록이 존재하고 해당 섹터에 데이터가 쓰여 있으면 덮어쓰기 연산이 발생한다.

덮어쓰기가 발생한 경우 복사 블록을 할당하여 복사 블록에 고정 섹터 방식으로 데이터를 쓴다. 이렇게 복사블록이 고정 섹터 방식으로 데이터를 저장하고 있으면 단순 블록 상태(simple block state)라고 한다. 덮어쓰기 된 섹터에 대해 다시 쓰기 명령이 들어오면 복사 블록에 고정 섹터 방식으로는 덮어쓰기가 불가능하므로 비어있는 섹터를 찾아 변동 섹터 방식으로 기록하는데 이러한 상태의 복사 블록을 복합 블록 상태(compound block state)라고 한다. 만약 복사 블록에 빈 섹터가 없으면 합병 또는 교환 연산을 수행하는데 복합 블록 상태라면 합병 연산을, 단순 블록 상태이면 교환 연산을 수행한다. 단순 블록 상태라면 복사 블록의 모든 내용이 오프셋에 맞게 순차적으로 기록되어 있으므로 새로운 블록을 할당 받지 않고 교환 연산을 수행한다. 복합 블록 상태이면 합병 연산을 수행하는데 교환연산은 복사블록의 내용을 옮기지 않고 매핑 테이블만 갱신하므로 두 개의 블록을 소거해야하는

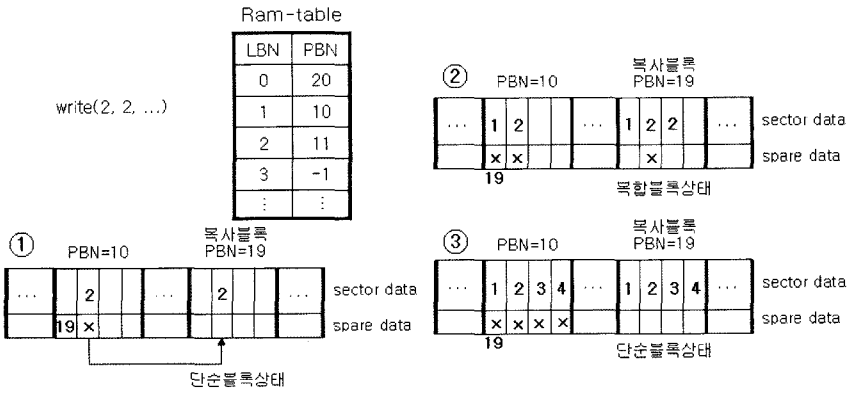

그림 7 M-system사의 알고리즘 구조도

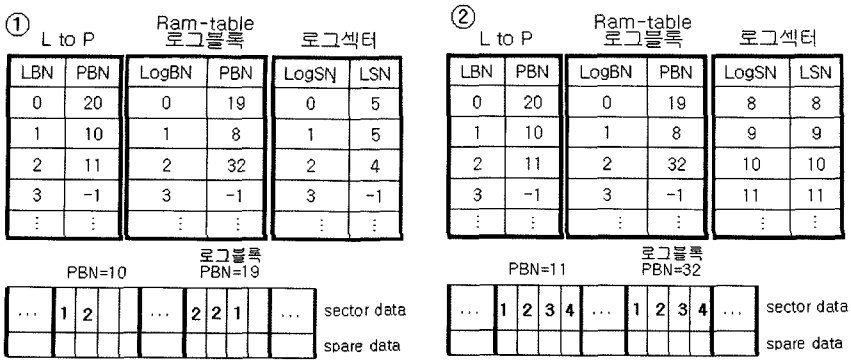

그림 8 로그 블록 기법을 사용하는 구조도

합병보다 소거 횟수를 줄인다. 또 합병은 소거하기 전 유용한 데이터를 새로운 블록에 기록해야 하므로 읽기, 쓰기 연산도 추가 수행된다. 이 알고리즘은 모든 블록에 대해 복사블록을 가지므로 모든 블록에 대해 덮어쓰기 명령이 수행되면 논리적인 크기의 두 배에 해당하는 물리적인 플래시 메모리의 크기가 필요하다.

3.3.3 로그 블록 기법

로그 블록 기법[5]은 로그 블록, 로그 블록 테이블을 사용하고 블록매핑을 한다. FTL의 쓰기 연산이 호출되면, FTL은 주어진 시작 논리적 섹터 번호를 이용하여 논리적 블록 번호를 계산하고 이 값을 이용하여 램 테이블로부터 물리적 블록 번호를 찾아낸다.

해당하는 물리적 블록 번호가 없는 경우, 빈 블록을 할당하여 섹터를 유효하다고 기록하고 데이터를 고정 섹터 방식으로 기록한 후 해당 블록의 첫 번째 섹터에 데이터 블록이라고 기록 한다. 해당하는 물리적 블록 번호가 매핑 테이블에 있는 경우, 해당 섹터가 비어 있으면 데이터를 기록하고 해당 섹터에 데이터가 있으면 덮어 쓰기 연산이 발생한다.

덮어 쓰기 연산이 발생할 경우 로그 블록 테이블에서 해당 블록에 해당하는 로그 블록이 있는 지 없는 지를 검사한다. 로그 블록이 없는 경우 로그 블록 테이블에 빈 로그 블록이 있는 지를 검사한다. 빈 로그 블록이 있는 경우 새로운 블록을 할당하여 로그 블록이라고 기록한 후 로그 블록에 변동 섹터 방식으로 데이터를 기록한다. 로그 블록이 가득 찼을 경우 로그 블록 중 하나를 희생 블록으로 선정하여 합병 연산을 수행한다. 합병 연산 수행 시, 데이터 블록을 새로 할당받아 해당 로그 블록의 섹터별 가장 최신 데이터를 찾아내어 기록하고 그 외에 기록이 되지 않은 기존 데이터 블록에 있는 데이터를 새로 할당 받은 데이터 블록에 기록한다. 덮어 쓰기 연산 발생 시 로그 블록 테이블에 해당 블록에 해당하는 로그 블록이 있는 경우, 빈 섹터가 발견되면 빈 섹터에 변동 섹터 방식으로 데이터를 기록한다. 빈 섹터가 없으면 교환 또는 합병 연산을 하는데 로그 블록에 순차적으로 데이터가 쓰인 경우 교환 연산을 수행하고, 로그 블록이 비순차적으로 되어 있으면 합병 연산을 수행한다. 이 방법은 여유 영역 기법이나 복사 블록 기법보다 더 나은 성능을 보이나 플래시 메모리의 사용량은 로그 개수 만큼 더 사용하므로 앞의 두 방법보다 더 적어 우수하다고 할 수 있다.

4. 플래시 메모리에서의 인덱스

데이터베이스의 인덱스의 경우 새로운 데이터의 삽입은 대부분 순차적인 쓰기 연산이 아닌 임의 쓰기 연산일 가능성이 높다. 순차쓰기는 기존 FTL에서도 빠르게 동작하는 연산이지만 임의 쓰기 연산은 플래시 메모리의 성능을 저하하는 연산으로써 이를 빠르게 처리할 필요가 있다. 플래시 메모리에서 인덱스를 빠르게 처리하기 위한 기법을 살펴본다.

4.1 BFTL

B+트리에 대한 효율적인 플래시 메모리 접근을 위한 연구로 BFTL[6]이 있다. BFTL은 B트리에 노드 삽입, 삭제, 갱신 등이 발생할 때 마다 색인 단위(index unit)를 생성한다. 색인 단위는 특정 노드에 속한 데이터(entry)와 색인 단위를 식별하기 위한 식별자 그리고 이것을 처리하기 위한 명령어 코드 등의 정보로 구성된다. 노드의 변경으로 인해 발생한 색인단위는 일시적으로 예약 버퍼(reservation buffer)라고 불리는 버퍼에 유지된다. 그리고 버퍼가 다 차면 버퍼에 유지되고 있던 색인 단위들을 플래시 메모리에 기록(commit) 한다. 색인단위는 플래시 메모리의 섹터보다 상대적으로 작다. 따라서 BFTL은 한 개의 섹터에 여러 개의 색인단위를 모아서 플래시 메모리에 기록한다.

BFTL은 노드에 변경을 반영하는 색인단위가 플래시 메모리상의 여러 섹터에 저장된다. 따라서 하나의 노

드를 구축하기 위해서는 여러 섹터에 저장되어있는 색인 단위를 모아야 한다. 이것은 특정 데이터를 검색하는데 비효율적으로 많은 읽기 연산을 발생시킨다. 또한 색인단위를 관리하기 위해 사용하는 노드 변환 테이블은 B+트리를 구축할 때 발생하는 많은 색인구조 때문에 빈번한 압축을 수행한다. 압축은 많은 읽기와 쓰기 연산을 발생하기 때문에 성능을 나쁘게 한다.

4.2 BOF(B-tree on flash memory)

BOF[7]는 BFTL의 문제점을 개선한 방법이다. BFTL은 하나의 노드에 관련된 색인단위를 여러 섹터에 저장하기 때문에 노드를 구축할 때 많은 읽기 연산이 발생한다. 또한 노드 변환 테이블을 관리하는데 압축 연산의 수행으로 인해 성능이 나빠지게 한다. BOF는 이 문제를 해결하기 위해 하나의 노드에 관련된 색인단위는 하나의 페이지에만 저장하도록 하였다.

BOF는 BFTL과 비교하여 읽기 성능을 증가 시켰고, 비효율적으로 운용해 오던 노드 변환 테이블을 제거하여 성능을 개선하였다. 그러나 BFTL이나 BOF는 버퍼에서 플래시 메모리로 데이터가 기록되는 순간에만 초점을 맞추었다. BFTL이나 BOF는 버퍼에 데이터가 입력되는 순간이나 버퍼에 데이터를 유지하는 동안 데이터를 효율적으로 관리 하지 못한다. BFTL이나 BOF에서 버퍼에 데이터를 유지하는 방법은 노드에 삽입, 삭제, 갱신 등의 연산에 의해 생성된 색인단위를 발생 순서대로 버퍼에 유지하는 것이다. 따라서 특정 노드에 반복적인 갱신이나 노드 분할과 같은 연산이 일어날 경우 노드의 특정 데이터에 관련된 색인단위가 중복되어서 버퍼에 유지될 수 있다. 이것은 버퍼의 공간 사용률을 떨어트리고, 색인단위의 데이터를 플래시 메모리로 기록하는 주기를 빠르게 만든다.

5. 플래시 메모리와 저장 관리자

앞에서 살펴본 FTL 들은 모두 순차적인 쓰기 연산이나 랜덤 쓰기 연산이더라도 동시에 몇 개의 페이지에 대한 쓰기 연산일 경우에는 좋은 성능을 보인다. 디지털 카메라 등의 이동형 기기는 이와 같은 엑세스 패턴을 가지므로 앞에서 설명한 FTL로 좋은 성능을 보일 수 있다. 하지만 데이터베이스는 거의 무작위적인 페이지 접근 경향을 보인다. 그러므로 기존의 FTL은 데이터베이스에서 아주 나쁜 성능을 보인다.

5.1 IPL

데이터베이스 한 레코드를 수정하게 되면 그 레코드가 속한 페이지를 변경하게 된다. 이런 변경은 결국 그 페이지를 디스크에 쓰게 된다. 플래시 메모리의 소거 단위인 블록은 데이터베이스 페이지의 크기보다 아주 크다. 블록의 크기는 대블록일 경우 128K 바이트이므로 데이터베이스의 페이지 크기가 8K 바이트일 경우 16개의 페이지가 한 블록에 적재된다. 이때 하나의 레코드 변경으로 인하여 페이지를 변경하게 되면 블록 내의 몇 개의 섹터가 변경하게 되고 이것은 많은 경우 합병(merge) 연산을 수반하게 된다. 이러한 합병 연산은 새로운 블록를 할당받고 기존 블록의 내용과 변경된 페이지의 내용을 새로운 블록으로 복사하고 기존 블록을 소거하는 연산을 말한다. 이것은 비용이 아주 비싼 연산으로써 이러한 비용을 줄이기 위하여 IPL[8]에서는 블록 내의 몇개의 섹터를 그 블록에서 변경된 사항을 기록하는 로그로 둠으로써 레코드의 변경이 합병 연산으로까지 확대되는 것을 막아 쓰기 연산의 성능을 향상하는 기법이다. 하지만 이 방법은 버퍼 관리자에서 페이지를 읽어들일때 블록 내의 페이지와 로그 데이터를 병합하여야 하는 오버헤드가 있다.

5.2 버퍼 교체 전략

버퍼에서 페이지를 교체할 때 기존의 데이터베이스는 LRU로 교체 페이지를 선택한다. 이렇게 선택된 페이지는 내용이 변경된(dirty) 페이지일 수도 있고 그렇지 않을 수도 있다. 만약 변경된 페이지일 경우는 버퍼에서 한 페이지를 교체할 때 두 번의 I/O가 발생한다. 플래시 메모리에서도 이는 마찬가지이나 변경된 페이지의 경우 한번은 읽기 연산이고 한번은 쓰기 연산이다. 쓰기 연산은 앞에서 살펴본 바와 같이 비용이 비싼 연산이기 때문에 수행 시간이 많이 걸린다. CFLRU[9]는 버퍼에서 페이지를 교체할 때 가능하면 변경되지 않은 페이지(clean page)를 먼저 선택하는 방법을 택하여 가능하면 쓰기 연산을 뒤로 미루는 방법을 취한다. 이렇게 되면 업데이트된 페이지가 버퍼에 오래 살아남을 수 있게 되어 그 페이지가 교체되기 전에 다시한번 업데이트가 되면 쓰기 연산 회수를 절약할 수 있는 장점이 있다. 이 방법은 스왑 메모리에서 적용한 방법이므로 임의 쓰기 연산이 많은 데이터베이스에서도 좋은 성능을 낸다는 것은 보장하기가 어렵다.

이와 유사한 연구로 FLRU[10]는 버퍼를 교체할 때 예상 비용을 계산하여 그 비용이 가장 작은 페이지를 교체 대상 페이지로 선택한다. 이때의 비용 계산은 dirty 페이지일 경우와 그렇지 않은 경우를 구분하고, 시간에 대한 가중치를 두어 비용을 계산하는 방법을 취하고 있다.

6. 플래시 메모리 기반 데이터베이스의 질의 최적화

기존의 데이터베이스는 데이터들이 디스크에 저장되어 있다고 가정하고 있다. 이러한 데이터베이스에서 질의 최적화를 할 때 가장 빠르게 수행될 수 있는 플랜으로 디스크를 참조하는 회수가 가장 작은 것을 선택한다. 이것은 CPU 연산에 비하여 디스크 연산 시간이 상대적으로 아주 크므로 I/O 회수를 줄이는 것을 가장 큰 목표로 한다. 디스크의 성능을 결정하는 것은 디스크의 특정 트랙으로 헤드를 옮기거나 디스크 원판이 회전하여 원하는 섹터를 헤드 아래에 위치시키는 기계적인 시간이 필요하기 때문이다.

플래시 메모리는 디스크에서와 같은 기계적인 시간이 필요하지 않다. 디스크에서는 원하는 데이터를 빠르게 읽기 위하여 데이터를 클러스트링하여 저장한다. 이것은 헤드의 움직임을 최소로 하여 기계적인 지연시간을 줄이기 위해서이다. 하지만 플래시 메모리는 데이터의 저장 위치와 관계없이 일정한 속도로 데이터를 읽어 들일 수 있다. 기계적인 지연시간이 없으므로 데이터를 물리적으로 클러스트링하는 단위가 디스크와 다를 것이다.

디스크에서는 섹터를 읽거나 쓰는데 걸리는 시간이 동일하다. 하지만 앞에서 살펴본 바와 같이 플래시 메모리는 읽기와 쓰기 연산의 수행시간이 서로 다르다. 그러므로 질의 최적화를 할 때 페이지를 읽어 들이는 회수와 페이지를 쓰는 회수를 구분하여 플랜을 작성하여야 한다. 플래시 메모리에서는 쓰기 연산이 쓰기 연산 외에 소거 연산을 유발할 수 있기 때문에 가능하면 소거 연산의 회수를 줄일 수 있도록 플랜을 생성하는 것이 질의를 빠르게 수행할 수 있는 방법이 될 것이다. 이처럼 플래시 메모리 기반의 데이터베이스에서 질의 최적화 방법은 디스크 기반의 데이터베이스와 다른 방법이 필요하며 이에 대한 연구가 진행되어야 할 것이다.

이와 다르게 PicoDBMS[11]에서는 플래시 메모리에서의 조인 연산을 빠르게 수행하기 위하여 조인될 수 있는 애트리뷰트에 대하여 연결 리스트로 미리 링크를 마련해두는 방법을 취했다. 하지만 이것은 스마트 카드에서 사용되는 NOR 플래시 메모리와 같은 시스템을 가정하고 있으며 일반적인 데이터베이스 서버에서 적용하기는 곤란한 방법이다.

4. 결 론

앞으로 이동형 기기나 많은 응용에서 저장 장치로 플래시 메모리가 널리 사용될 것으로 예상된다. 플래시 메모리의 용량이 매년 크게 증가하고 있으므로 조만간 임베디드 데이터베이스나 데이터베이스 서버에서 그 저장 장치로 플래시 메모리를 채택하는 경향이 증가할 것으로 예상된다. 하지만 플래시 메모리는 그 소자의 특성상 읽기, 쓰기 연산의 성능이 비대칭적이다. 또한 덮어 쓰기 연산을 수행하려면 소거 연산을 수행해야 하는데 이들의 수행 시간이 아주 다르다. 기존의 데이터베이스는 저장 장치로 디스크를 가정하고 있기 때문에 저장 시스템, 버퍼 관리자, 질의 처리기 등이 모두 디스크 기반의 모델로 구성되어 있다. 하지만 기존의 시스템을 플래시 메모리에서 동작하면 원하는 성능을 얻을 수 없다. 그러므로 저장 관리자, 버퍼 관리자, 인덱스, 질의 최적화기 등이 플래시 메모리에서 효율적으로 동작할 수 있도록 새로운 기법들이 제안되어야 할 것이다.

참고문헌

[1] Eran Gal and Sivan Toledo, "Algorithms and Data Structures for Flash Memories," ACM Computing Surveys, 37(2), Jun. 2005

[2] Tae-Sun Chung, Dong-Joo Park, Sangwon Park, Dong-Ho Lee, Sang-Won Lee and Ha-Joo Song, "System Software for Flash Memory: A Survey," EUC, Aug. 2006

[3] Takayuki Shinohara, "Flash Memory Card with Block Memory Address Arrangement," United States Patent 5,905,993, 1999

[4] Petro Estakhri and Berhanu Iman, "Moving Sequential Sectors within a Block of Information in a Flash Memory Mass Storage Architecture," United States Patent 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 Compact Flash Systems," IEEE Transactions on Comsumer Electronics, 48(2), 2002

[6] Chin-Hsien Wu, Li-Pin Chang and Tei-Wei Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems," Real-Time Computing Systems and Applications, 2003

[7] 남정현, 박동주, "플래시 메모리 상에서 B-트리 설계 및 구현," 정보과학회논문지:데이터베이스, 34(2), 2007년 4월

[8] Sang-Won Lee and Bongki Moon, "Design of Flash-Based DBMS: An In-Page Logging Approach," SIGMOD, 2007

[9] Seon-Yeong Park, Dawoon Jung, Jeong-uk Kang,

Jin-Soo Kim, Joonwon Lee, "CFLRU: A Replacement Algorithm for Flash Memory," International Conference on Compilers, Architecture and Synthesis for Embedded Systems, 2006

[10] 박종민, 박동주, "플래시 메모리상에서 시스템 소프트웨어의 효율적인 버퍼 페이지 교체 기법", 정보과학회논문지:데이터베이스, 34(2), 2007년 4월

[11] Christophe Bobineau, Luc Bouganim, Philippe Pucheral and Patrick Valduriez, "PicoDBMS: Scaling Down Database Techniques for the Smartcard," VLDB, 2000

박 상 원

1994 서울대학교 컴퓨터공학과 학사
1997 서울대학교 컴퓨터공학과 석사
2002 서울대학교 컴퓨터공학과 박사
2002~2003 세종사이버대학교 디지털콘텐츠학과 전임강사
2003~현재 한국외국어대학교 정보통신공학과 조교수

관심분야 : 플래시 메모리, 임베디드 데이터베이스, XML
E-mail : swpark@hufs.ac.kr