

대용량 파일시스템을 위한 선형 해싱 디렉토리 구현

이현석*, 김신우, 이용규
동국대학교 컴퓨터공학과

Implementation of a Linear Hashing Directory for Very Large File Systems

Hyun Suk Lee, Shin Woo Kim, Yong Kyu Lee
Dept. of Computer Engineering, Dongguk University

요 약

대규모의 대용량 파일 시스템에서 전통적인 파일 시스템인 UNIX와 같은 디렉토리 구조를 갖게 되면, 파일 탐색 시 순차 검색으로 인해 많은 시간이 걸리게 된다. 이러한 문제를 해결하고자, 본 논문에서는 파일의 수를 고려하여 파일의 수가 적을 때는 inode 블록에 디렉토리 엔트리를 함께 저장하고, 파일의 수가 많아져 inode 블록에서 오버플로우가 발생하면 선형 해싱(Linear Hashing)을 이용하여 디렉토리 엔트리를 저장하고자 한다. 선형 해싱 디렉토리 구조의 설계 및 구현에 대해서 설명하고, 일반적으로 대용량 파일 시스템에서 많이 사용되는 B+ 트리 디렉토리 구조와 성능을 비교 분석한다.

1. 서론

최근 정보의 양이 급속히 증가하고 이를 관리해야 할 파일의 수가 많아짐에 따라, 이들을 효과적으로 관리하고 검색하는 것이 실제 시스템의 성능을 높이는 중요한 관점이 되었다.

그러나, UNIX[5]와 같은 전통적인 파일 시스템에서는 파일을 탐색할 때 순차적으로 검색하므로 많은 시간이 걸리게 된다. 따라서, GFS(Global File System)[3]나 JFS(Journaling File System)[8]와 같이 많은 수의 파일을 포함한 대용량 파일 시스템에서는 전통적인 파일 시스템과 달리 직접 검색 방법을 이용한 디렉토리 구조를 이용함으로써 이런 문제를 해결하고자 하였다[6][7].

최근 각광을 받고 있는 대용량 저장 장치인 SAN(Storage Area Network)을 이용한 대표적인 파일 시스템인 GFS에서는 확장 해싱(Extendible Hashing)[1]을 이용하여 디렉토리를 관리한다. 확장 해싱 디렉토리는

엔트리의 빠른 탐색을 가능하게 하고, 해시 테이블의 크기를 증가시키면서 보다 많은 엔트리의 유지를 가능하게 한다. 이러한 확장 해싱의 사용은 파일의 양이 많고 적음과 관계없이 사용 가능하고 많은 수의 파일을 검색시 비교적 적은 디스크 블록 접근 횟수를 갖는 것이 특징이다. 그러나, 확장 해싱의 특성상 해시 테이블 내에 블록 주소를 중복으로 저장하는 경우가 발생한다.

리눅스에서 사용하는 대표적인 저널링 파일 시스템인 JFS에서는 B+ 트리를 이용하여 디렉토리를 관리한다. B+ 트리 구조를 사용하여 파일 이름으로 정렬된 형태의 목록을 쉽게 얻을 수 있으며, 블록 헤더에 자유 공간 리스트를 유지하여 저장 공간을 효율적으로 활용할 수 있다. 또한, 파일의 순차적인 탐색이 용이하고, 파일의 양이 많아도 비교적 적은 디스크 블록 접근으로 파일을 검색할 수 있다.

본 논문에서는 대용량 파일 시스템에 적합하도록 기존의 확장 해싱과 B+ 트리 이외에 다른 직접 검색 방법인 선형 해싱(Linear Hashing)[2][4]을 이용한 디

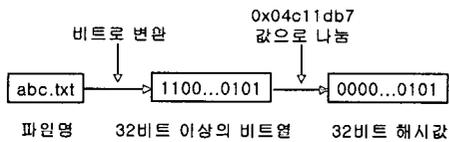
렉토리 구조의 설계 및 구현에 대해 설명한다. 또한 그 중에서도 일반적으로 많이 사용하고 있는 B+ 트리를 이용한 디렉토리 구조를 구현하여 선형 해싱과 성능을 비교 분석한다.

2. 선형 해싱 디렉토리

본 절에서는 대용량 파일 시스템에서의 빠른 검색을 위한 선형 해싱 디렉토리 구조에 대해 살펴본다.

2.1 해시 함수

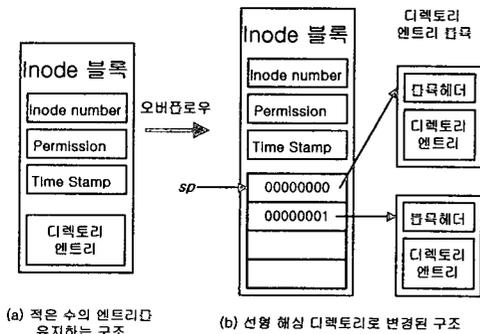
선형 해싱에서 오버플로우가 보다 적게 발생하기 위해서는 다양한 해시값을 생성하는 해시 함수가 요구된다. 본 논문의 선형 해싱 디렉토리 구조에서는 데이터 통신에서 활용되는 CRC-32 오류 검출 코드(32-bit Cyclic Redundancy Check Code)를 사용한다. CRC-32 오류 검출 코드는 하나의 비트 에러를 검출하기 위해 고안된 것으로, 이를 해시 함수에 이용하면 해시값을 고루 분포 시켜주는 특징을 가진다.



[그림 1] 해시 함수를 적용하는 과정

[그림 1]은 선형 해싱 디렉토리 구조에서 CRC 해시 함수를 적용하는 과정이다. 먼저 파일 이름을 32비트 이상의 비트열로 변환하고 이를 CRC-32에서 사용하는 키값으로 나누게 되면 32비트의 해시값을 얻을 수 있다.

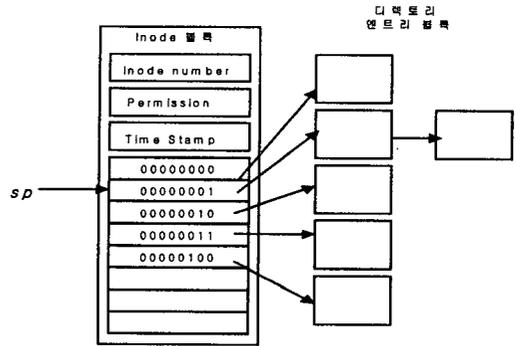
2.1 디렉토리 구조



[그림 2] 디렉토리 구조의 전환

선형 해싱 디렉토리는 적은 수의 엔트리를 빠르게 접근하기 위해서 앞의 [그림 2]의 (a)와 같이 처음 디렉토리 엔트리들을 Inode 블록에 저장한다. 이후 Inode 블록에 엔트리가 오버플로우되면 [그림 2]의 (b)와 같이 선형 해싱 구조로 변경한다.

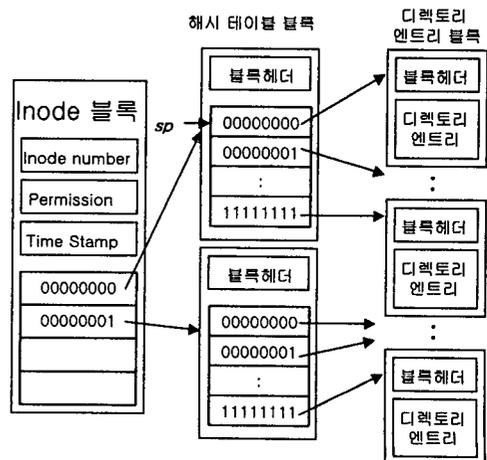
선형 해싱 디렉토리 구조의 일반적인 구조는 [그림 2]와 같으며 블록의 크기는 4KB이다.



[그림 3] 선형 해싱 디렉토리 구조

Inode 블록에 2^8 만큼의 엔트리를 가질 수 있는 해시 테이블을 유지하고, 해시 테이블은 디렉토리 엔트리 블록의 주소를 저장한다. 파일 이름의 해시 값을 이용해서 해시 테이블을 거쳐 디렉토리 엔트리 블록을 찾는다. 이때, 디렉토리 엔트리 블록에서 오버플로우가 발생하면 스플릿 포인터 *sp*를 이용하여 확장한다.

[그림 3]의 구조에서 다시 오버플로우가 발생하게 되면 [그림 4]와 같이 Inode 블록에서 해시 테이블을 분리하여 새로 할당된 해시 테이블 블록으로 테이블을 확장한다. 해시 테이블은 Inode 블록을 통해 간접 접근한다.



[그림 4] 해시 테이블의 확장

한편, [그림 4]의 해시 테이블은 최대 2^{16} 만큼의 엔트리를 가질 수 있는 해시 테이블이며, 만약 파일 이름의 길이가 모두 8바이트라고 가정하면, 하나의 디렉토리 엔트리 블록에 약 160개의 엔트리가 저장된다. 이 때, 디렉토리 엔트리 블록에 약 절반의 엔트리가 들어간다고 가정하면, 약 5백만개 ($256 \times 256 \times 160 \times 1/2$)의 디렉토리 엔트리를 두번의 디스크 블록 접근으로 탐색이 가능하게 된다. 또한, 해시 값은 32비트이므로 해시 테이블은 4 레벨까지 확장이 가능하다.

2.3 디렉토리 연산

선형 해싱 디렉토리에서 주로 사용하는 연산은 디렉토리 엔트리 블록을 찾기 위한 삽입 연산과 삭제 연산 그리고 탐색 연산이 있다.

[그림 5]와 같이 `dir_add` 함수를 이용하여 새로운 파일을 삽입하며 디렉토리 구조를 확장할 수 있다.

```

알고리즘 dir_add
입력 : 디렉토리 엔트리 정보
출력 : 삽입 성공 여부
(
    디렉토리 엔트리가 삽입될 블록을 찾는다;
    if ( 오버플로우 )
    (
        새로운 디렉토리 엔트리 블록을 할당받는다;
        if ( 블록 할당 실패 )
            return false;

        할당받은 블록에 입력으로 받은 엔트리 정보를
        삽입한다;
        디렉토리 엔트리 블록의 헤더 정보를 수정한다;
        /* 오버플로우시 선형 해싱 구조를 확장하기
        위해서 구조의 변경이 필요 */
        스펙트 연산을 수행한다;
    )
    else
    (
        디렉토리 엔트리 블록에 입력으로 받은 엔트리
        정보를 블록의 끝에 삽입한다;
    )
    return true;
)
    
```

[그림 5] 선형 해싱 디렉토리에서의 삽입

삽입은 디렉토리 엔트리 블록을 찾아서 삽입할 공간이 있으면 그 찾은 블록에 삽입한다. 그러나, 공간이 없으면 새로운 블록을 할당받아 삽입하고 스펙트 포인터 `sp`에 위치한 블록을 분할하는 과정을 거친다.

[그림 6]은 탐색 연산에 관한 알고리즘이다.

```

알고리즘 dir_search
입력 : 파일의 이름
출력 : 탐색 성공 여부
(
    파일의 이름을 사용하여 해시값을 구한다;
    /* 해시 테이블의 위치를 찾기 위해 index 값을
    구한다. */
    index=해시값 % 2^d;
    if ( index < sp )
        index=해시값 % (2^(d+1));

    해시 테이블의 index 번째의 디렉토리 엔트리
    블록 주소를 얻는다;
    디렉토리 엔트리 블록으로부터 디렉토리 엔트리를
    순차 탐색한다;

    if ( 탐색성공 )
        return true;
    else
        return false;
)
    
```

[그림 6] 선형 해싱 디렉토리에서의 탐색

탐색은 해시값과 스펙트 포인터 `sp` 값을 이용해서 해시 테이블의 위치를 찾고 해시 테이블에 연결된 디렉토리 엔트리 블록을 찾아내어 블록 내에서 엔트리를 찾는다.

4. 성능 분석

본 절에서는 본 논문에서 구현한 선형 해싱 디렉토리 구조와 일반적인 파일시스템의 디렉토리 구조인 B+트리 구조의 성능을 비교한다. 실험에 이용된 디스크는 Seagate사의 ST51080A 모델[9]이며 디스크의 성능은 [표 1]과 같다.

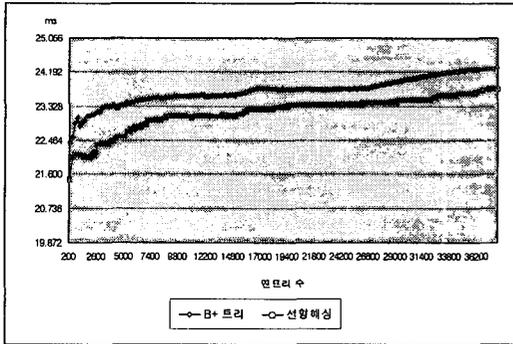
[표 1] Seagate사의 ST51080A 디스크의 성능

매개변수	값
평균 탐색 시간	10.5 ms
평균 회전 지연 시간	5.58 ms
데이터 전송률	67.7 Mbps

4.1 삽입 시간

[그림 7]은 선형 해싱 디렉토리 구조와 B+ 트리 디

렉토리 구조에서 각각의 엔트리를 순차적으로 삽입하면서 걸리는 시간의 평균을 나타낸다.

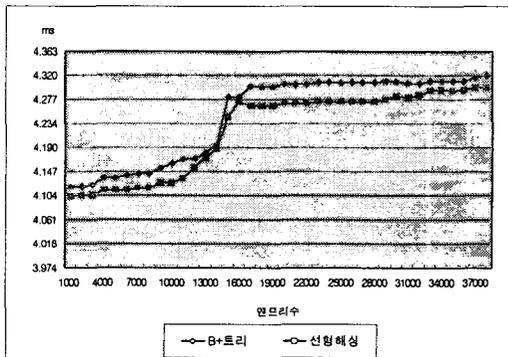


[그림 7] 평균 삽입 시간

[그림 7]에서 보는 바와 같이 선형 해싱 디렉토리 구조가 B+ 트리 디렉토리 구조에 비해서 디렉토리 엔트리를 삽입하는 데 소요 시간이 적음을 알 수 있다. 이것은 삽입 시간에 탐색 시간과 디렉토리 엔트리 블록 내에 엔트리들을 순차적으로 유지하는 시간이 포함되기 때문인데, 선형 해싱은 B+ 트리 구조보다 탐색에 시간이 덜 걸리고 B+ 트리 구조처럼 엔트리를 순차적으로 유지하지 않기 때문에 좀더 좋은 성능을 보여 주게 된다.

4.2 탐색 시간

[그림 8]은 선형 해싱 디렉토리 구조와 B+ 트리 디렉토리 구조에서 각각의 엔트리를 탐색하는데 평균 걸리는 시간을 보여준다.



[그림 8] 평균 탐색 시간

[그림 8]에서 보는 바와 같이 평균 탐색 시간도 B+ 트리 디렉토리 구조에 비해 선형 해싱 디렉토리 구조가 더 적은 시간이 걸리는 것을 알 수 있다. 이것은 B+ 트리 디렉토리에서 디렉토리 엔트리 블록을 찾을

때 파일 이름을 키값으로 트리를 탐색하는 데에 약간의 시간이 더 소모되는 것으로 보여진다. B+ 트리 디렉토리에서는 파일 이름으로 일일이 비교 검색하는데 반해, 선형 해싱 디렉토리에서는 해시 함수를 이용하여 해시값을 생성하여 그 해시값으로 직접 찾아가기 때문이다.

5. 결론

본 논문에서는 대용량 파일 시스템을 위한 디렉토리 구조를 선형 해싱을 사용하여 구현하였다. 선형 해싱을 사용함으로써 파일 탐색시에 일반적인 디렉토리 구조인 B+ 트리 구조보다 빠르게 디렉토리 엔트리 블록에 접근할 수 있었다. 또한 디렉토리 엔트리 수의 증가에 따라 디렉토리 레벨을 증가시켜 많은 엔트리를 저장할 수 있게 하였고, 많은 수의 엔트리 탐색에도 일정한 성능을 유지할 수 있게 하였다.

향후에는 서로 다른 환경에서의 파일시스템에 대한 연구와 다양한 성능 평가를 통해서 각각의 구조와 환경에 맞는 디렉토리 구조에 관한 연구가 필요하다.

[참고문헌]

- [1] Ronald Fagin, et. al., "Extendible Hashing - A Fast Access Method for Dynamic Files," ACM Transactions on Database Systems, vol. 4, no. 3, pp. 315-344, September 1979.
- [2] W. Litwin, "Linear hashing: a new tool for file and table addressing," Proceedings of the 6th Conference on Very Large Databases, pp. 212-223, Montreal, Canada, Oct 1980.
- [3] Kenneth W. Preslan, et. al., "A 64-bit, Shared Disk File System for Linux," Proceedings of the 16th IEEE Mass Storage Systems Symposium, pp. 22-41, San Diego, California, March 1999.
- [4] K. Ramamohanarao and R. Sacks-Davis, "Recursive linear hashing," ACM Transactions on Database Systems, vol. 8, no. 9, pp. 369-391, September 1984.
- [5] Uresh Vahalia, UNIX Internals: The New Frontiers, Prentice-Hall, 1996.
- [6] 김신우, 이용규, 김경배, 신범주. "SAN 기반 공유 파일 시스템을 위한 디렉토리 구조 설계" 한국멀티미디어 학회 '01 추계 학술발표논문집, pp. 503-507, 2001. 11.
- [7] 김신우, 이현석, 이용규. "대용량 파일 시스템을 위한 디렉토리 구조 비교," 한국정보처리학회 '03 춘계 학술발표논문집, vol. 10, no. 1, pp. 455-458, 단국대학교, 2003. 5.
- [8] JFS Overview, <http://www-106.ibm.com/developerworks/library/1-jfs.html>.
- [9] Seagate ST51080A Hard-Disk Spec., <http://www.seagate.com/support/disc/ata/st51080a.html>.