대규모 파일 시스템을 위한 동적 해싱 디렉토리

김신우, 이용규
동국대학교 컴퓨터공학과

Dynamic Hashing Directories for Large File Systems

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

요 약

최근 대용량 데이터의 저장과 검색을 요구하는 파일시스템이 필요하게 됨에 따라, 별도의 서버를 두지 않고 분산된 클라이언트가 메타데이터를 직접 관리하면서 모든 저장 장치들에 접근할 수 있는 SAN 기반 리눅스 클러스터 파일시스템의 연구가 활발하게 진행 중에 있다. 이러한 대규모 파일 시스템을 위해서는 동적 해싱을 이용한 디렉토리 관리가 요구되므로, 본 논문에서는 그 중 확장 해싱 디렉토리 구조와 선형 해싱 디렉토리 구조를 설계 및 구현하고, 구현된 시스템을 이용하여 성능평가를 통해 두 디렉토리 구조의 성능을 분석한다. 비교 분석 결과, 파일의 삽입 성능에서는 선형 해싱 기반의 디렉토리가 우수하였으나, 공간 활용면에서는 확장 해싱 기반의 디렉토리가 우수한 성능을 보였다.

1. 서론

최근 기업들의 데이터가 매년 75~1150%의 비율로 증가함에 따라 기업이 요구하는 스토리지의 양은 기하급수적으로 늘어나게 되었다. 따라서, 네트워크 기반 공유 저장 장치를 이용하여 작은 규모의 컴퓨터들을 클러스터로 연결한 하나의 통합된 시스템을 구축하려는 연구가 활발히 진행 중에 있다.

이들 파일 시스템들이 사용하는 대용량의 저장장치로 별도의 고속 데이터 전용 네트워크인 화이버 채널을 통해 클라이언트와 저장 장치들을 연결하는 SAN (Storage Area Network)을 들 수 있으며, 최근에 이를 이용한 파일 시스템으로는 미네소타 대학에서 구현된 GFS(Global File System)[2]와 한국전자통신연구원에서 개발하고 있는 SANtopia[5]가 이에 해당된다. 이와 같은 SAN 기반 대용량 파일 시스템들은 별도의 서버를 두지 않고 분산된 클라이언트가 메타데이터를 직접 관리하면서 저장 장치들에 접근하여 모든 저장 장치들에 접근을 일정하도록 하여 하나의 서버에 업무가 집중되는 현상을 막을 수 있다.

한편, 대부분의 UNIX 시스템에서는 디렉토리 내의 파일 이름들을 파일의 생성 순서로 유지하므로 특정 파일의 이름을 디렉토리 내에서 탐색할 때 순차적으로 검색하여야만 한다. 따라서, 많은 파일 이름들을 포함한 SAN 기반 대용량 파일 시스템에서 UNIX 시스템과 같은 방법으로 디렉토리 구조를 갖게 되면, 특정 파일을 검색하는 데 많은 시간이 소요될 수 있다. 그러므로, SAN 기반 대용량 파일 시스템에서는 기존의 파일 시스템의 디렉토리에서의 비효율적인 순차적 검색을 극복하기 위해서 동적 해싱인 확장 해싱 (Extendible Hashing)[1]을 이용하거나, 선형 해싱 (Linear Hashing)[1]을 이용하여 디렉토리 구조를 관리한다. 확장 해싱은 디렉토리의 파일의 수가 많고 적음에 상관없이 모두 수용 가능하고, 비록 많은 수의 파일이 존재하더라도 해싱의 특성상 빠른 검색이 가능하며, 선형 해싱 또한 디렉토리의 파일의 수가 많고 적음에 관계없이 수용 가능하고 별도로 해시테이블이 존재하지 않으므로, 삽입시 오버플로우가 발생하여도 디렉토리 엔트리 블록의 수가 천천히 증가하여 확장 해싱의 해시 테이블이 2배 증가하는 데 소요되는 시간에 비해 데이터 삽입 시간이 적게 소요될 수 있다.

본 논문에서는 SAN 기반 대용량 파일 시스템을 위한 확장 해싱을 이용한 디렉토리 구조와 선형 해싱을 이용한 디렉토리 구조를 설계 및 구현하고, 성능 평가를 통해 두 디렉토리 구조를 비교 분석한다.

2. 확장 해싱을 이용한 디렉토리 구조

SAN 파일 시스템에서는 확장 해싱을 이용하여 디렉토리를 관리할 수 있다. 디렉토리에 삽입하고자 하는 디렉토리 엔트리 이름을 이용하여 해시 함수를 통해 해시 값을 구하고, 이를 활용하여 직접 저장할 데

이터 블록의 주소를 찾아 삽입, 삭제 및 검색을 할 수 있다. 확장 해싱은 데이터를 순차 접근이 아닌 직접 접근하므로 빠른 연산이 가능하다.

2.1 해시 함수

해시 함수는 데이터 통신에서 에러 검출 코드로 활용되는 CRC-32 코드(32-bit Cyclic Redundancy Check Code)[4]를 사용하며, 그림 1은 CRC 해시 함수를 이용하여 해시 값을 알아내는 과정이다.

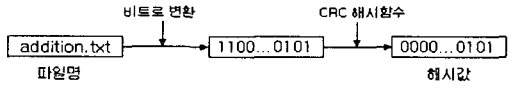

그림 1. 해시 함수를 적용하는 과정

처음에 파일 이름을 입력받아서 이를 비트로 변환한다. 이 때 변환된 비트는 최소 32비트 이상이 되며, CRC-32에서 주로 사용하는 키 값인 0x04c11db7로 나눈다. 이때, 32비트 미만의 나머지가 생기고 이를 해시 값으로 사용하기 위해서 상위 비트를 0으로 채우면 32비트의 해시 값을 구할 수 있다.

2.2 확장 해싱 디렉토리

SAN 파일 시스템의 디렉토리 구조는 파일의 수를 고려하여 결정한다. 파일의 수가 적을 때는 inode 블록에 디렉토리 엔트리를 함께 저장하고, 파일의 수가 많아져 inode 블록에서 오버플로우가 발생하면 확장 해싱을 이용하여 디렉토리 엔트리를 저장한다.

2.2.1 디렉토리 구조

Inode 블록에 디렉토리 엔트리를 직접 저장할 수 없게되면 확장 해싱을 이용하는 해시 테이블 구조로 전환되며 그림 2와 같다.

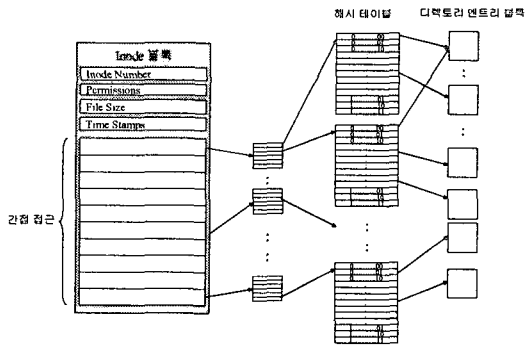

그림 2. 확장 해싱을 이용한 디렉토리 구조

Inode 블록을 접근하여 해당 해시 테이블 엔트리 블록을 검색하고 그 곳에서 원하는 디렉토리 엔트리 블록의 주소를 검색한다. 이와 같은 디렉토리 구조는 거대한 해시 테이블을 가질 수 있으므로 대용량의 디렉토리 엔트리들을 수용할 수 있고, 찾고자 하는 디렉토리 엔트리를 해시 값을 이용하여 빠르게 검색할 수 있다.

한편, 한 블록의 크기를 4KB, 보통 파일 이름의 길이를 8bytes라 생각할 때 166개의 디렉토리 엔트리들을 저장할 수 있다. 또, 해시 값이 32비트이므로 해시 테이블은 최대 2^{32}의 크기를 가질 수 있어, 이때 166개의 디렉토리 엔트리들을 저장할 수 있는 디렉토리 엔트리 블록이 50%만 채워졌다고 생각할 때, 최소 222억개 $((256\times1/2)^4 \times 166\times1/2 = 22{,}280{,}142{,}848)$ 정도의 디렉토리 엔트리를 저장할 수 있다.

2.2.2 디렉토리 연산

그림 3은 확장 해싱에서의 삽입 연산 함수이다.

```
알고리즘 exhash_insert
입력 : 디렉토리 엔트리 정보
출력 : 저장
{
 입력된 정보중에서 파일 이름을 이용하여 해시 값을 계산;
 해시 값을 이용하여 해시 테이블에 연결된 알맞은 디렉토리 엔트리 블록을 찾음;
  if(찾은 디렉토리 엔트리 블록에 오버플로우 발생){
    if(찾은 디렉토리 엔트리 블록의 링크수 == 1){
      /* hash_table glow */
      if(해시 테이블 블록에 오버플로우 발생){
       /* 레벨 증가 */
          레벨의 증가를 위해서 2개의 디렉토리 블록 할당;
          디렉토리 구조 2배로 확장;
      }
      else{  /* 같은 레벨에서의 디렉토리 구조 확장 */
          해시테이블 블록 할당, 디렉토리 구조 2배로 확장;
      }
      비교하는 비트수 증가;
      새로운 디렉토리 엔트리 블록 할당;
    }
    else {  /* hash_table split */
      새로운 디렉토리 엔트리 블록할당;
      링크 포인터 변경;
    }
  }else  찾은 공간에 데이터 저장;
}
```

그림 3. 확장 해싱 디렉토리의 엔트리 삽입

해시 값을 이용하여 저장할 디렉토리 엔트리 블록을 찾아간다. 그러나, 할당하고자 하는 디렉토리 엔트리 블록이 오버플로우가 발생하면 그 블록에 연결된 링크의 수를 본다. 하나의 링크 포인터로 연결된 경우에는 디렉토리 크기를 2배로 확장시키고, 그렇지 않을 경우에는 새로운 디렉토리 엔트리 블록을 추가하여 링크 포인터를 수정해 준다.

3. 선형 해싱을 이용한 디렉토리 구조

SAN 파일 시스템은 선형 해싱을 이용하여 디렉토리 구조를 관리할 수 있다. 해시 값을 이용하여 저장할 데이터 블록에 데이터를 저장하나, 별도의 해시 테이블이 존재하지 않는 선형 해싱 구조에서는 삽입시 오버플로우가 발생하여도 디렉토리 엔트리 블록만 증가하므로, 확장 해싱 구조에서의 해시 테이블이 2배 증가하는 데 소요되는

시간에 비해 데이터 삽입 시간이 적게 소요될 수 있다.

3.1 해시 함수

선형 해싱은 블록의 분할된 여부에 따라 각각 다른 해시 함수를 사용하여 해시 값을 구한다. 각 해시 함수의 치역은 전의 것보다 두 배씩 되는 해시 함수 계열(h_0, h_1, h_2, ...)을 사용한다.

h_i(해시 값) = h(해시 값) mod (2^iN)

d_0: N을 표현하는데 필요한 비트 수

$d_i = d_0+i$ 블록의 초기 개수

예를 들면, N = 32 → d_0 = 5 일 때, 다음과 같다.

h_0 = h mod 32, 치역: 0-31 →

h_1 = h mod 64, 치역: 0-63 → ...

라운드 번호 레벨(Level) 중에는 해시 함수 h_{Level}과 $h_{Level-1}$만을 사용하며, 각 라운드가 시작될 때 있던 블록들이 첫 블록부터 마지막 블록까지 차례로 분할되어 블록의 수가 두 배로 되면 다음 레벨로 증가한다. 본 논문에서 사용한 해시 값은 확장 해싱에서 사용한 CRC 해시 함수를 사용해서 나온 값을 사용한다.

3.2 선형 해싱 디렉토리

SAN 파일 시스템의 디렉토리 구조는 파일의 수를 고려하여 결정하며, 파일의 수가 적을 때는 inode 블록에 디렉토리 엔트리를 함께 저장하고, 파일의 수가 많아져 inode 블록에서 오버플로우가 발생하면 선형 해싱을 이용하는 구조로 변하여 디렉토리 엔트리를 저장한다.

3.2.1 디렉토리 구조

Inode 블록에 디렉토리 엔트리를 직접 저장할 수 없게되면 선형 해싱을 이용한 디렉토리 구조로 전환되는데 그림 4와 같다.

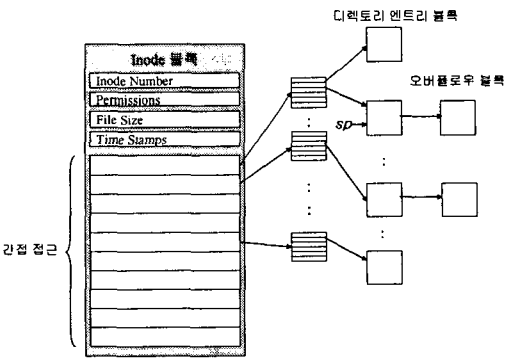

그림 4. 선형 해싱을 이용한 디렉토리 구조

Inode 블록에 디렉토리 엔트리 블록의 주소를 저장한다. 파일 이름의 해시 값을 이용해서 디렉토리 엔트리 블록을 찾는다. 이때, 디렉토리 엔트리 블록에서 오버플로우가 발생하면 스플릿 포인터 *sp*를 이용하여 확장한다.

그림 4에서 보는 바와 같이 해시 값이 32비트이므로 디렉토리 엔트리 블록을 최대 2^{32}까지 가질 수 있으므로, 최소 222억개 ($(256\times1/2)^4 \times 166\times1/2$ = 22,280,142,848) 정도의 디렉토리 엔트리를 저장할 수 있다.

3.2.2 디렉토리 연산

그림 5는 선형 해싱에서의 삽입 함수로, 새로운 데이터를 블록에 삽입하며 디렉토리를 확장할 수 있다.

```
알고리즘 linear_insert
입력 : 디렉토리 엔트리 정보
출력 : 삽입 성공 여부
{
  디렉토리 엔트리가 삽입될 블록을 찾음;
  if(오버플로우)
  {
    새로운 디렉토리 엔트리 블록을 할당받음;
    if(블록 할당 실패)
    { return false; }
    할당받은 블록에 입력으로 받은 엔트리 정보를 삽입;
    디렉토리 엔트리 블록의 헤더 정보를 수정;
    /* 오버플로우시 선형 해싱 구조를 확장하기 위해서
       구조의 변경이 필요 */
    스플릿 연산을 수행;
  }
  else
  {
    디렉토리 엔트리 블록에 입력으로 받은 엔트리 정보를
    블록의 끝에 삽입;
  }
  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 스플릿 포인터 *sp*에 위치한 블록을 분할하는 과정을 거친다.

4. 성능평가

본 절에서는 SAN 파일 시스템을 위해 구현한 확장 해싱 디렉토리 구조와 선형 해싱 디렉토리 구조의 성능을 평가한다. 성능은 펜티엄 III-450MHz, LINUX 6.2, 커널 2.2.18에서 실험하였다. 실험에 이용된 디스크는 Seagate사의 ST51080A 모델[6]이며 표 1과 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 디렉토리 엔트리 삽입

확장 해싱 디렉토리 구조에서와 선형 해싱 디렉토리 구조에서의 디렉토리 엔트리를 삽입하는 데 소요되는 평균 수행 시간의 성능 평가 결과는 그림 6과 같다.

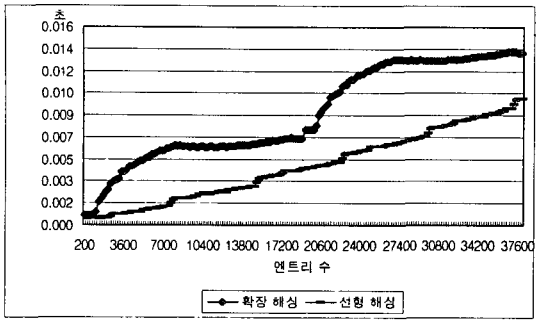

그림 6. 디렉토리 엔트리 평균 삽입 시간

확장 해싱 디렉토리 그래프에서 갑자기 증가하는 부분은 해시 테이블이 inode 블록에서 분리되는 지점으로 이는 다른 때보다 오버헤드가 있음을 보여주고, 선형 해싱은 전반적으로 천천히 평균 삽입 시간이 증가함을 보여준다. 오버플로우가 발생했을 경우, 해시 테이블의 증가하는 부분을 수행하는 확장 해싱 디렉토리가 선형 해싱 디렉토리보다 많은 수행 시간을 소요함을 볼 수 있다.

4.2 디렉토리 엔트리 검색

확장 해싱 디렉토리 구조에서와 선형 해싱 디렉토리 구조에서의 디렉토리 엔트리를 검색하는 데 소요되는 평균 수행 시간의 성능 평가 결과는 그림 7처럼 두 해싱이 비슷하거나 선형 해싱이 적게 소요됨을 보인다.

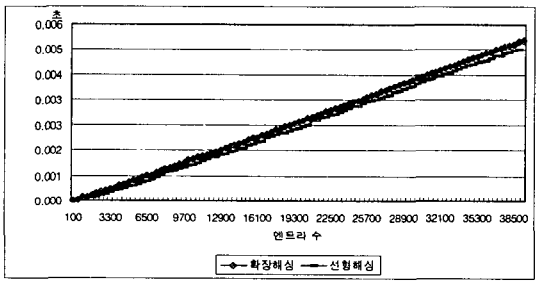

그림 7. 디렉토리 엔트리 평균 검색 시간

확장 해싱 디렉토리는 해시 테이블을 거쳐서 디렉토리 엔트리 블록에 접근하는데 반해, 선형 해싱 디렉토리에서는 직접 디렉토리 엔트리 블록에 접근할 수 있어 선형 해싱 디렉토리가 검색 시간이 적게 걸리나, 선형 해싱 디렉토리에서도 오버플로우가 많이 발생하게되면 여러 번의 블록 접근을 하게되므로 소요 시간이 비슷하게 될 수 있다.

4.3 디렉토리 엔트리 블록 사용 수

확장 해싱 디렉토리 구조에서와 선형 해싱 디렉토리 구조에서 사용하는 디렉토리 엔트리 블록 수는 그림 8과 같다. 확장 해싱 디렉토리가 해시 테이블 블록을 이용하기는 하나, 해시테이블의 깊이가 8일 때까지는 inode 블록에 해시테이블을 넣고 있어, 별도의 블록을 사용하지 않고 있다. 이에 반해, 선형 해싱은 별도의 오버플로우 블록을 사용하므로, 엔트리 수가 많아짐에 따라 사용하는 블록 수도 많아지게 된다.

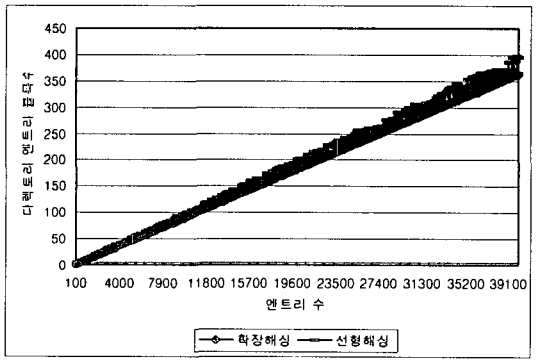

그림 8. 디렉토리 엔트리 블록 사용 수

5. 결론

본 논문에서는 대규모 SAN 기반 대용량 파일 시스템을 위한 확장 해싱 디렉토리 구조와 선형 해싱 디렉토리 구조를 설계 및 구현하였다. 이들 디렉토리 구조는 디렉토리 엔트리의 수를 고려하여 적은 양의 디렉토리 엔트리들은 inode 블록에 직접 저장함으로써 한 번의 접근으로 원하는 엔트리 정보를 검색할 수 있게 하였고, 많은 양의 디렉토리 엔트리들에 대해서는 확장 해싱을 이용하거나 선형 해싱을 이용하도록 하였다. 성능평가를 통하여 파일의 삽입 성능에서는 선형 해싱 기반의 디렉토리가 우수하였으나, 공간 활용면에서는 확장 해싱 기반의 디렉토리가 우수한 성능을 보였다.

향후에는 해시 함수와 오버플로우 발생율에 따라 두 해싱 디렉토리의 성능이 달라질 수 있으므로, 이에 따른 두 해싱 디렉토리의 성능 비교 연구가 요구된다.

[참고문헌]

[1] Panos E. Livadas, File Structures, Prentice-Hall, 1990.
[2] 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.
[3] Ashok Rathi, Huizhu Lu, G. E. Hedrick, " Performance comparison of extendible hashing and linear hashing techniques," Proceedings of the 1990 ACM SIGSMALL, pp. 178-185, Crystal City, Virginia, United States.
[4] Andrew S. Tanenbaum, Computer Networks, Prentice-Hall, 1996.
[5] 이용규, 김신우, 손덕주, "SAN 환경 공유 디스크 파일 시스템의 메타데이터 관리," 정보과학회지 19권 3호, pp. 33-42, 2001. 3.
[6] Seagate ST51080A Hard-Disk Spec., http://www.seagate.com/support/disc/ata/ st51080a.html, 2003.