대용량 파일 시스템을 위한 메타데이터 구조 설계

김신우 ° *, 이용규*, 김경배**, 신범주**
*동국대학교 컴퓨터공학과, **한국전자통신연구원 인터넷서비스연구부

Metadata Structure Design for Very Large File Systems

Shin Woo Kim ° *, Yong Kyu Lee*, Gyoung Bae Kim**, Bum Joo Shin**
*Dept. of Computer Engineering, Dongguk University
**Internet Service Department, ETRI

요 약

멀티미디어 데이터의 크기가 커짐에 따라, 파일 시스템에 대량의 데이터를 저장하는 것이 필요하다. 기존의 파일 시스템이 대용량의 데이터를 저장하는 면에서 우수한 성과를 얻기 어려움에 따라, SAN(Storage Area Network)을 이용한 새로운 파일 시스템이 최근에 연구되고 있다. SAN을 이용한 파일 시스템인 GFS는 기존 파일시스템들에 비해 대량의 데이터를 저장할 수 있도록 설계되었지만, inode와 빈 공간의 관리가 효율적이지 못하였다. 본 논문에서는 이러한 문제점을 해결하기 위해서 GFS의 inode 구조, 데이터 블록 할당 기법, 그리고 빈 공간 관리 기법에 중점을 두어 메타데이터 구조를 개선한다. 그 결과 데이터 블록 접근 시간을 줄이는 새로운 inode 구조를 설계하고, 큰 파일에는 연속된 블록으로 된 익스텐트로 할당하고 작은 파일에는 블록들로 할당하는 새로운 데이터 할당 기법을 제시한다. 또한, 빈 공간을 신속히 할당 회수할 수 있도록 빈 익스텐트들과 블록들의 주소를 저장하여 두는 독창적인 빈 공간 지갑을 사용한다. 성능 분석 결과 이러한 개선 방안들이 기존의 다른 시스템들보다 효율적임을 알 수 있다.

1. 서론

정보화가 진행됨에 따라 정보의 양도 급속히 증가하고 있으며, 이를 저장할 대용량의 저장 장치들이 필요하다. 따라서, 최근에는 네트워크 기반 공유 저장 장치를 가진 분산 파일 시스템에 대한 연구가 활발히 진행중이다[1][4][5].

공유 저장 장치를 가진 분산 파일 시스템은 별도의 서버를 두지 않고 각각의 분산된 클라이언트가 메타데이터 관리를 하고 직접 저장 장치들에 접근하는 것을 말한다. 이 시스템은 모든 저장 장치들에 접근을 일정하도록 하여 부하의 균형을 이룰 수 있다. 그리고 각각의 클라이언트는 독립적으로 저장 장치들에게 데이터를 요구할 수 있으며, 하나의 장치에 이상이 생겨도 나머지 장치들에 거의 영향을 주지 않는다는 장점이 있다. 여기에 SAN(Storage Area Network)이 해당되고, 이를 이용한 파일 시스템으로 GFS(Global File System) [4][5][6]를 들 수 있으며, GFS는 대용량을 저장하기 위해서 UNIX의 단점을 보완한 시스템이다.

먼저, UNIX 시스템 V[2][3]는 inode를 이용하여 데이터 블록에 접근하는데 하나의 inode는 13개의 엔트리로 구성되어 있다. inode의 처음 10개의 엔트리는 inode에서 직접 접근하기 위한 데이터 블록들의 주소를 저장하고 있고, 11번째 엔트리부터는 데이터 블록들에 간접 접근, 이중 간접 접근, 삼중 간접 접근을 한다. 이러한 방식은 작은 파일인 경우에는 직접 데이터 블록에 접근할 수 있어 접근 시간이 적게 걸린다는 장점이 있는 반면, 큰 파일인 경우에는 여러 레벨들을 거쳐 데이터 블록에 간접 접근을 하기 때문에 접근 시간이 많이 걸리는 단점이 있다.

이러한 단점을 극복하기 위해서 GFS는 리프의 데이터 블록들이 모두 트리의 동일한 레벨에 존재하는 평면 파일 구조(Flat File Structure)를 이용한다. 먼저 dinode에서 파일의 위치를 읽은 후, 링크를 따라 트리의 높이(height)와 같은 레벨에 있는 데이터 블록에 접근한다. 이 구조는 파일에 데이터 블록을 할당할 때, 현재 트리로 모든 데이터 블록들을 저장할 수 없다면, 트리의 높이를 하나 증가시켜 모든 데이터 블록들을 저장한다. 그러므로 큰 파일인 경우, 유닉스에 비하여 모든 데이터 블록들에 대한 임의의 데이터 블록 접근 시간이 일정하고 접근 시간도 짧아지는 장점이 있다. 그러나, 데이터 블록 저장 공간이 부족하여 트리의 레벨을 증가하여야 할 경우 그에 따른 모든 데이터 블록의 접근 시간이 늘어나는 단점이 있다.

분산 환경에서 대용량의 데이터를 지원하기 위해서는 데이터 블록의 빠른 접근이 필요하고 저장공간의 효율적 사용이 요구된다. 우리는 이를 위해 GFS의 inode 구조의 개선을 통해 디스크 접근 시간을 줄이고자 하며, 아울러 GFS와 차별된 블록 할당 방법과 새로운 빈 공간 관리 방법을 제시하고자 한다. 앞으로 본 논문에서 제안할 파일 시스템을 MGFS(Modefied Global File System)라고 하기로 한다. 이는 GFS를 기본으로 하여 개선된 파일 시스템을 의미한다.

2. 대용량 파일 시스템을 위한 효율적인 메타데이터 구조 설계

GFS의 플랫 파일 구조를 두 레벨까지 데이터 블록을 할당할 수 있는 새로운 세미 플랫 파일 구조(semi_flat file structure)를 설계하고, 기존의 파일 시스템들과 달리 파일의 크기에 따라 큰 파일들은 일정한 연속적인 블록인 익스텐트 단위로 할당하며, 블록을 회수할 때 생기는 빈 공간(free space)을 빈 공간 지갑(purse)을 이용하여 관리하는 방법을 모색한다.

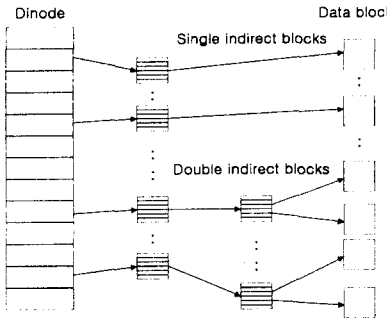
2.1 Inode의 구조

새로 제안한 inode 구조는 현재의 트리 높이에 데이터 블록들을 할당하고 더 필요한 부분들을 위해 트리의 높이를 하나 증가시켜 새로운 레벨에 남은 데이터 블록들을 둔다.

[그림 1]은 MGFS의 세미 플랫 파일 구조(semi_flat file structure)이다. 그림을 보면 모든 데이터 블록들이 동일한 레벨에 있지 않고 트리의 높이와 (높이-1)에 걸쳐 있음을 볼 수

본 연구는 한국전자통신연구원의 2000년도 위탁과제 "SAN 환경에서의 Global File Metadata 관리 기법 연구"의 일부로 수행됨.

있다. MGFS에서는 모든 데이터 블록들이 파일의 크기에 따라 GFS에서처럼 플랫 구조를 가질 수도 있고 [그림 1]처럼 두 레벨에 걸쳐 있을 수도 있다. 이것이 GFS와의 차이점이다. 이 방법은 GFS의 임의의 데이터 블록 접근 시간을 단축시킨다.



[그림 1] MGFS의 inode 구조

MGFS에서 inode 구조를 생성하는 알고리즘은 다음의 [그림 2]와 같다.

```
알고리즘 iscreate
입력 : 파일의 크기(n)
출력 : 파일의 inode 구조
{
  while(완료되지 않았음)
{
 파일의 크기를 이용하여 포인터 블록들이 위치할 레벨 k와 k-1을 구한다;
  /* 파일의 크기가 하나의 레벨로 적당할 때는 레벨 k만 구한다. */
 k+1 레벨에 있을 데이터 블록 수를 구한다;
 k+1 레벨에 있는 데이터 블록에 대한 포인터 블록 수를 구한다;
  /* p는 한 블록이 가지는 포인터의 개수 */
  /* k 레벨에는 p^(k-1) - ⌈ (n - p^(k-1)) / p^(k-1) ⌉ ×p개의 익스텐트와 블록이 있다. */
  /* k+1 레벨에는 n에서 k레벨에 있는 데이터 블록 수를 뺀 나머지가 있다. */
 k 레벨에서 k+1 레벨의 두 레벨에 데이터 블록들이 걸쳐 있게 inode 구조를  생성한다;
 inode 리스트에서 inode 번호를 제거하고 파일에 할당한다;
 }
}
```

[그림 2] 파일의 inode 구조 생성

2.2 데이터 블록 할당

파일의 크기가 64K 보다 작을 파일인 경우에는 일반적인 1K 블록 단위로 할당하고, 파일의 크기가 64K 이상인 경우에는 연속된 블록들인 64K의 익스텐트(extent) 단위로 할당하고 마지막에 남은 블록들은 내부 단편화를 막기 위해 1K 블록 단위로 할당한다.

이러한 할당 방법은 디스크의 탐색 시간을 줄이므로 읽는 속도가 향상되고, 또 데이터 블록들을 가리키는 포인터의 수가 줄어듦으로써 디스크 공간을 효율적으로 사용할 수 있다.

2.3 빈 공간 관리

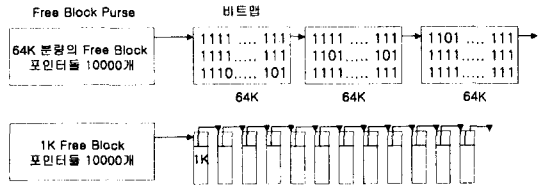
파일 크기에 따른 블록 할당방법을 효과적으로 지원하기 위해서는 빈 공간도 익스텐트 단위와 블록 단위의 두 개의 리스트로 관리해야 한다. 이때 이 논문에서 독창적으로 빈 공간의 주소들의 모임인 빈 공간 지갑을 이용한다.

가. 지갑을 이용한 빈 공간 관리

지갑(purse)은 현재 할당 가능한 익스텐트나 블록들의 디스크 시작 주소와 연속된 개수의 쌍으로 구성되며 대부분의 할당과 회수가 가능하도록 충분한 크기를 갖는다.

(1) 지갑과 비트맵을 이용한 빈 공간 관리

이 방법은 빈 공간을 익스텐트 단위와 블록 단위의 두 개의 단위로 각각 관리하며 익스텐트를 관리할 때에는 비트맵을 이용하고 블록을 관리할 때는 링크드 리스트를 이용한다. 이때, 비트맵에서 빈 공간은 이미 사용한 것으로 표시하여 빈 공간을 할당할 때 일일이 빈 공간을 찾는 시간과 비트맵 표시하는 시간을 줄이게 된다.



[그림 3] 빈 공간 캐쉬를 이용한 빈 공간 관리(bitmp)

[그림 3]처럼 지갑을 두어 할당하며 요구하는 양이 그것을 초과할 때에는 링크를 따라서 저장 공간을 할당한다. 빈 공간 지갑을 이용하여 빈 공간을 할당하다가 일정 비율 이상의 공간이 할당됐다면 연결된 공간의 주소를 가져와서 지갑에 있는 주소를 적정 수준으로 만든다.

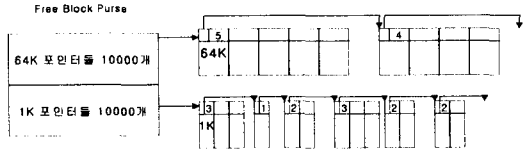
(2) 지갑과 링크드 리스트를 이용한 빈 공간 관리

이 방법은 앞의 방법에서 비트맵을 사용하는 대신 링크드 리스트를 이용한다. 익스텐트 단위와 블록 단위의 두 개의 단위로 각각 관리하며 각각의 공간들은 링크드 리스트를 이용하여 연결되어 있다.

나. 연속적인 공간 할당을 위한 빈 공간 관리

이 방법은 앞에서 설명한 링크드 리스트를 이용한 빈 공간 관리 방법과 거의 같다. 차이점이라면 블록들이 몇 개의 연속된 블록들로 되어 있다는 것이다. 이 방법은 연속적으로 되어 있지 않은 것보다 지갑에서 같은 포인터로 더 많은 블록을 관리, 할당할 수 있으므로 공간 절약과 접근 시간을 줄일 수 있는 장점이 있다.

[그림 4]와 같이 블록들을 링크 시킬 때 가능하면 연속적인 공간이 묶여지도록 관리한다. 맨 앞에 있는 익스텐트나 블록에 연속되어 있는 익스텐트나 블록들의 개수를 숫자로 표시한다.



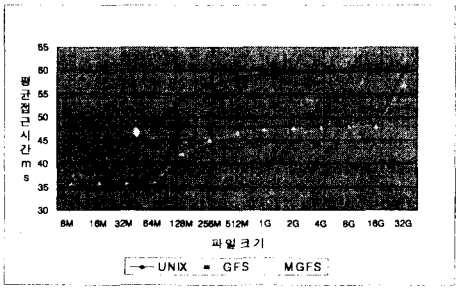
[그림 4] 빈 공간 지갑을 이용한 연속적인 빈 공간 관리

3. 성능 평가

본 절에서는 앞에서 제안한 방법들이 기존의 방법들에 비해 성능을 얼마나 향상시킬 수 있는가를 분석하고자 한다.

3.1 Inode 구조의 성능 평가

기존의 UNIX, GFS, 그리고 MGFS의 inode 구조에 대한 임의의 블록에 대한 평균 접근 시간을 비교한다.

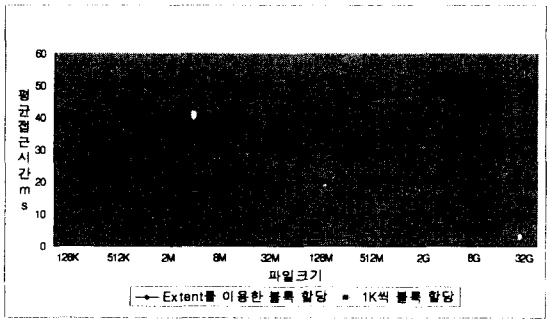


[그림 5] 임의의 블록 평균 접근 시간 비교

[그림 5]에서 MGFS는 GFS에서 급격하게 증가하는 부분이 완만하게 증가하고 있음을 그래프를 통해서 확인할 수 있다. 이것은 MGFS의 세미 플랫 구조가 GFS의 플랫 구조보다 성능이 우수하다는 것을 보여준다.

3.2 블록 할당 기법의 성능 평가

파일의 크기와 상관없이 일정하게 데이터 블록을 할당하는 방법과 파일의 크기에 따라 데이터 블록을 할당하는 방법의 성능을 비교한다.



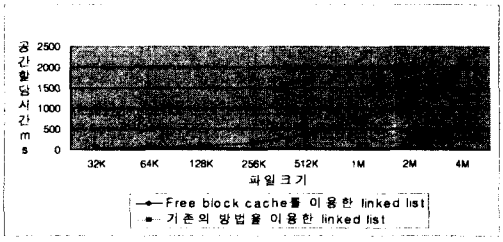
[그림 6] 블록 할당 기법 성능 비교

[그림 6]은 앞의 블록 할당 방법을 MGFS의 inode 구조와 연결지어 분석한 것이다. 익스텐트 블록으로 할당하면 그만큼 트리의 레벨이 서서히 증가하게 되므로 데이터 블록의 평균 접근 시간이 줄어들게 된다. 챠트에서 급격하게 증가하는 부분은 트리의 레벨이 하나 증가함을 나타낸 것으로 1K씩 블록 할당하는 방법이 더 빠른 레벨의 증가를 보여준다.

3.3 빈 공간 관리 기법의 성능 평가

2절에서 소개한 빈 공간 관리 기법 중에 빈 공간 지갑을 이용한 링크드 리스트와 기존의 방법을 이용한 링크드 리스트를 비교하고자 한다.

[그림 7]은 빈 공간 지갑을 이용한 데이터 접근은 항상 일정한 시간이 걸리는데 반해, 일반적인 데이터 접근은 데이터 블록을 많이 요구하면 할수록 빈 공간들을 하나 하나 확인하면서 찾아야 하기 때문에 할당 시간이 많이 걸리는 것을 보여준다.



[그림 7] 빈 공간 관리 기법 성능 비교

4. 결론 및 향후 연구

본 논문에서는 공유 저장 장치를 가진 분산 파일 시스템의 일종인 GFS (Global File System)의 구조를 개선하여 데이터 접근 시간을 줄일 수 있도록 GFS의 세미 플랫 파일 구조를 제안하였으며, 파일의 크기에 따라 블록의 크기를 다르게 하여 큰 파일의 경우에는 연속된 블록들인 익스텐트 단위로 할당하도록 하였다. 그리고, 블록을 회수할 때 생기는 빈 공간을 관리하는데 빈 공간 지갑을 두어 빈 공간의 주소를 관리함으로써 빠르게 빈 공간을 할당하거나 회수할 수 있도록 하였다. 결론적으로 데이터 블록에 대한 빠른 접근, 빈 공간의 빠른 할당과 회수, 그리고 저장공간의 효율적 활용이 가능하게 되었다.

SAN을 이용한 대용량 파일 시스템의 성능을 보다 개선하기 위해, 향후 데이터 블록의 크기에 따른 효율적인 버퍼 관리 방법이 요구된다. 또한 예측하지 못한 클라이언트의 문제 발생으로부터 메타데이터를 보존하기 위해서 트랜잭션 개념을 이용하는 저널링 메커니즘을 설계하는 것이 필요하다.

5. 참고 문헌

[1] Tomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, "Serverless Network File Systems," the 15th ACM Symposium on Operating Systems Principles, December 1995.

[2] Maurice J. Bach, "The Design of the UNIX Operating System," 1986.

[3] Milind M. Buddhikot, Xin Jane Chen, Dakang Wu, Guru M. Parulkar, "Enhancements to 4.4 BSD UNIX for Efficient Networked Multimedia in Project MARS," Proc. IEEE Int. Conf. on Multimedia Computing and Systems, Jun 1999.

[4] Steven R. Soltis, Thomas M. Ruwart, Matthew T.O Keefe, "The Global File System," the Fifth NASA Goddard Space Flight Center Conf. On Mass Storage Systems and Technologies, Sept 17-19, 1996, College Park, MD.

[5] Steven R. Soltis, Thomas M. Ruwart, Matthew T.O Keefe, "A 64-bit, Shared Disk File System for Linux," the Sixteenth IEEE Mass Storage Systems Symposium, March 15-18, 1999, San Diego, California.

[6] GlobalFileSystem June 30, 2000, http://www.globalfilesystem.org/.