The Journal of GIS Association of Korea, Vol.10, No.1, pp.93-106, March 2002

GIS에서 대용량 파일을 위한 대용량 공유 디스크 파일시스템의 메타데이터 구조

김경배 · 이용주 · 박춘서 · 신범주*

Metadata Structrues of Huge Shared Disk File System for Large Files in GIS

Gyoung-Bae Kim · Young-Ju Lee · Choon-Seo Park · Bum-Ju Shin*

요 약

기존의 파일시스템은 소형의 파일을 효과적으로 저장하고 관리하기 위해서 설계되었다. 따라서 기존의 유닉스나 리눅스와 같은 파일 시스템은 지리정보시스템에서 발생되는 대용량의 지리정보 데이터를 효과적으로 처리하는 것은 어렵다.

본 논문에서는 지리정보시스템에서 발생되는 기가에서 테라바이트의 대용량 데이터 파일을 저장하기 위한 효과적인 메타데이터 구조 및 관리 기법을 제안한다. 제안된 기법에서는 대용량 파일을 저장하기 위해 동적 다단계 기법을 사용하고 있으며, 대용량의 파일 시스템을 제공하기 위하여 동적 비트맵 기법을 사용한다. 본 논문에서 제안된 기법은 SAN 환경에서의 대용량 공유 디스크 파일시스템인 SANtopia에서 구현되었다.

주요어 : 대용량 파일시스템, 메타데이터, SAN, 리눅스, 공유디스크 파일시스템

ABSTRACT : The traditional file systems are designed to store and manage for small size files. So, we cannot process the huge files related with geographic information data using the traditional file system such as unix file system or linux file system. In this paper, we propose new metadata structures and management mechanisms for the large file system in geographic information system. The proposed mechanisms use dynamic multi-level inode for large files

* 한국전자통신연구원 컴퓨터소프트웨어연구소 컴퓨터시스템연구부
Dept of Computer System Research, Computer & Software Research Labs, ETRI

and dynamic bitmap for huge file system. We implement the proposed mechanisms in the metadata structures of SANtopia is shared disk huge file system for storage area networks(SAN).

Keywords : huge file system, metada, SAN, linux, shared disk file system

1. 서 론

지리정보시스템에서는 영상 이미지, 비디오, 오디오 및 텍스트와 같은 방대한 양의 데이터를 저장하고, 이를 효과적으로 관리 할 수 있어야 한다[1,2]. 이러한 지리정보시스템에서 발생되는 대용량의 데이터를 저장하기 위해서는 테라 바이트 이상의 물리적인 저장장치의 제공과 함께 이들 저장장치를 위한 파일 시스템이 제공되어야 한다.

대용량 저장장치의 제공을 위해 최근에 주목을 받고 있는 시스템으로는 SAN (Storage Area Network)[3]를 들 수 있다. SAN은 별도로 고속의 데이터 전용 네트워크인 Fibre Channel[4]을 통해 클라이언트와 저장 장치들을 연결한다. SAN을 지리정보시스템에 이용하면 대량의 파일을 저장하기 위한 물리적인 공간을 얻을 수 있으나, 이 물리적인 대용량의 공간에 파일을 효과적으로 저장하고 관리하기 위한 대용량 파일 시스템이 필요하다.

컴퓨터가 처음으로 개발된 이래로 파일 시스템과 관련된 연구와 개발은 지속적으로 이루어져 왔고 많은 제품들이 출시되었다. 대표적으로 유닉스 파일시스템[5], 리눅스 파일 시스템(Ext2)[6], 윈도우즈 파일 시스템(NTFS)[7] 등을 들 수 있다. 이러한 파일 시스템에서는 디스크나 씨디롬 등과 같은 저장 장치에 파일을 저장하고 관리하기 위해서 파일에 대한 정보를 관리하기 위한 아이노드, 디렉토리 엔트리, 그리고 데이터 블록 등의 메타데이터를 효과적으로 관리해야 한다[8,9,10]. 이를 위해서 유닉스나 리눅스 파일 시스템에서는 파일 시스템을 생성할 때, 저장 장치 공간을 아이노드, 디렉토리 엔트리, 데이터 블록의 세 개의 영역으로 구분하고, 이를 관리하기 위해 각 객체에 해당하는 비트맵 영역을 정적으로 할당하여 관리하는 기법을 사용한다. 그러나 기존의 파일 시스템은 파일의 크기가 기가 바이트 미만은 파일들을 효과적으로 저장하고 이를 관리하기 위해 설계되었다. 따라서 지리정보시스템에서 다루는 영상 이미지, 비디오, 오디오 및 텍스트와 같은 방대한 양의 데이터를 기존의 파일 시스템 구조에서는 효과적으로 저장할 수 없다.

따라서 본 논문에서는 지리정보시스템에서 발생되는 멀티미디어 데이터와 같이 크기가 큰 다수의 파일들을 효과적으로 저장하기 위한 대용량 파일 시스템을 위한 새로운 메타데이터 구조 및 관리기법을 제안한다.

제안된 기법에서는 대용량 파일 시스템

을 제공하기 위하여 고정된 테이블을 이용하는 기존의 방법과는 달리 파일 시스템에서 발생되는 메타데이터에 따라 동적으로 비트맵을 할당하고 디스크 영역을 관리하는 동적 비트맵 기법을 사용한다. 또한 기존의 유닉스나 리눅스에서 파일의 최대 크기를 제한하는 정적 아이노드 대신에 파일의 크기에 따라 아이노드가 확장되는 동적 아이노드 기법을 사용하여 대용량의 파일을 효과적으로 저장한다. 제안된 기법은 한국전자통신연구원에서 개발중인 SAN 환경에서의 공유 디스크 파일 시스템인 SANtopia[11, 12]에서 구현되었다.

본 논문의 구성은 다음과 같다. 2장에서 관련연구로서 기존의 파일시스템에 대하여 설명한다. 3장에서는 대용량 공유 디스크 파일시스템인 SANtopia에 대하여 설명하고, 4장에서는 대용량 파일 시스템을 제공하기 위한 동적 비트맵 구조를 설명한다. 5장에서는 파일의 크기에 제한받지 않고 파일을 생성할 수 있는 아이노드 구조에 대하여 설명하고, 6장에서는 본 논문에서 제안한 기법에 대한 성능 평가를 수행하고 7장에서 본 논문에 대한 결론을 맺는다.

2. 관련연구

2.1 유닉스 파일 시스템

UNIX[5]는 주로 프로그램 개발과 문서 작성 환경을 위한 시분할 운영체제로 Bell 연구소에서 60년대 말 처음으로 발표되었고, 그 이후 계속적으로 기능을 추가하며 새로운 버전을 출시하였다. 그 중 본 보고서에서는 1983년 발표한 UNIX 시스템 V를 중심으로 살펴본다. UNIX는 파일을 모아 그룹으로 만들어 계층적으로 관리하며 새로운 내용을 추가하여 정해진 한도 내에서 동적으로 파일을 크게 할 수 있는 특징이 있다. 그리고 하나의 파일을 동시에 읽고 수정할 수 있으며, 파일의 접근을 조절하고 데이터의 무결성을 유지하는 특징도 있다.

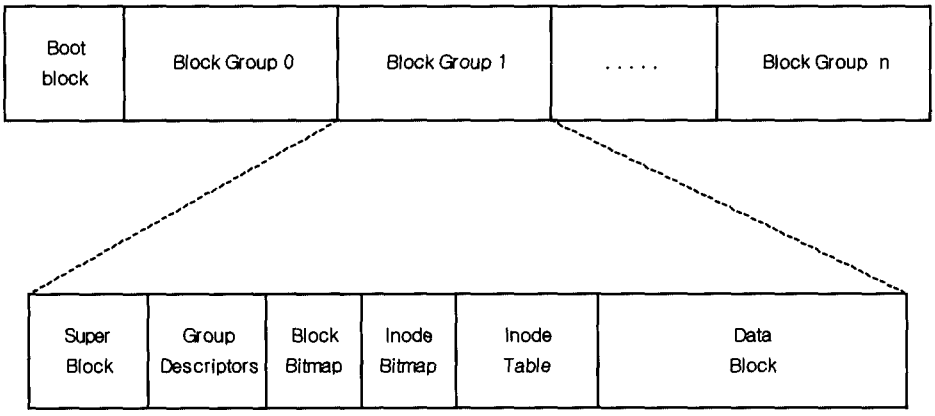
[그림 1] 유닉스 파일시스템(s5fs)의 구조

UNIX 파일 시스템의 배치는 다음 [그림 1]과 같이 부트 블록, 수퍼 블록, 아이노드 리스트 그리고 데이터 블록 순서로 되어 있다. 부트 블록은 파일 시스템의 시작부분에 위치하며, 전형적으로 첫 번째 섹터를 차지하고 있다. 이 부분에는 운영체제를 초기화하기 위한 부트 트랩 코드가 있다. 수퍼 블록은 파일 시스템의 상태 정보를 가지고 있다. 파일 시스템에 있는 총 블록의 수, 빈 공간 리스트 내

에 있는 아이노드의 수, 빈 공간 비트맵, 블록의 크기, 빈 공간의 수 그리고 사용중인 블록의 수 등의 정보가 있다. 아이노드 리스트는 파일의 정보를 가지고 있는 아이노드들이 모여 있다. 데이터 블록은 파일 데이터와 관리용 데이터들로 구성되어 있다.

2.2 리눅스 파일시스템

리눅스 시스템에서는 다양한 형태의 파일시스템이 지원되고 있으며 대표적인 파일 시스템은 Ext2 파일 시스템이다. Ext2 파일시스템은 BSD의 Fast File System(BSD FFS)[13] 의 영향을 많이 받았으며, [그림 2]와 같은 구조를 이루고 있다.



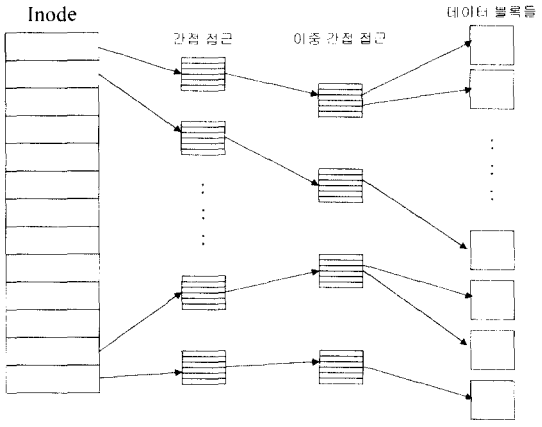
[그림 2] 리눅스 파일시스템(Ext2)의 구조

파티션은 FFS에서 실린더 그룹에 대응되는 블록그룹(block group)으로 나누어지며, 각 블록그룹은 파일시스템을 관리하기 위한 슈퍼블록, 그룹을 관리하기 위한 그룹기술자, 데이터 블록의 할당을 관리하기 위한 블록 비트맵, 아이노드의 할당을 관리하기 위한 아이노드 비트맵, 그리고 파일을 저장하기 위한 아이노드 테이블과 파일, 간접블록, 그리고 디렉토리를 저장하기 위한 데이터 블록으로 이루어져 있다.

Ext2 파일 시스템에서는 아이노드 테이블의 크기와 아이노드 비트맵의 크기를 결정하기 위해 데이터블록 4KB 당 1개의 아이노드를 할당하는 고정된 방법을 사용한다. 한 블록의 크기가 1024 바이트인 경우에 각 그룹 당 8192개의 블록을 관리할 수 있고, 총 블록 수는 실제 파티션 블록 수보다 작은 것은 8192 블록으로 그룹을 할당하고 마지막에 할당하기에 부족한 블록들은 사용하지 않는다. 블록 비트맵과 아이노드 비트맵 블록 내의 1 비트는 1 블록을 나타낸다. 따라서 1개의 비트맵 블록으로 8192개(1 그룹)의 블록 사용상태를 표현할 수 있다.

2.3 GFS(Global File System)

GFS[14,15]는 기존의 UNIX 파일 시스템을 개선한 독특한 플랫(Flat) 구조를 이루고 있다. GFS에서의 아이노드 구조는 [그림 3]과 같다. 모든 데이터 블록들은 트리의 높이와 같은 리프 레벨에만 위치한다. 이는 모든 데이터 블록의 임의 접근 시간이 같아지도록 하며, 매우 큰 파일의 경우는 트리의 높이를 증가시키면 되므로 파일의 크기에 제한이 없어지는 장점을 갖는다.



[그림 3] GFS inode의 플랫 구조

그러나, 이와 같은 장점들에도 불구하고, GFS에서는 항상 플랫 구조를 유지하여야 하므로 파일의 크기가 커질수록 데이터 블록의 평균 임의 접근 시간이 길어지는 결과를 초래한다. 예를 들어, 어떤 파일 A의 아이노드 트리가 정(full) k-ary일 경우, 즉, A의 크기가 이 트리의 높이 h로 수용할 수 있는 최대의 크기일 경우에, 이보다 조금이라도 큰 파일 B는 플랫 구조를 유지하기 위하여 트리의 높이가 h+1이 되어야 하므로 이 새로운 파일 B는 A보다 데이터 블록을 임의 접근하는 데에 한번씩의 간접 접근이 추가로 필요해진다.

3. SANtopia 시스템

3.1 시스템 환경

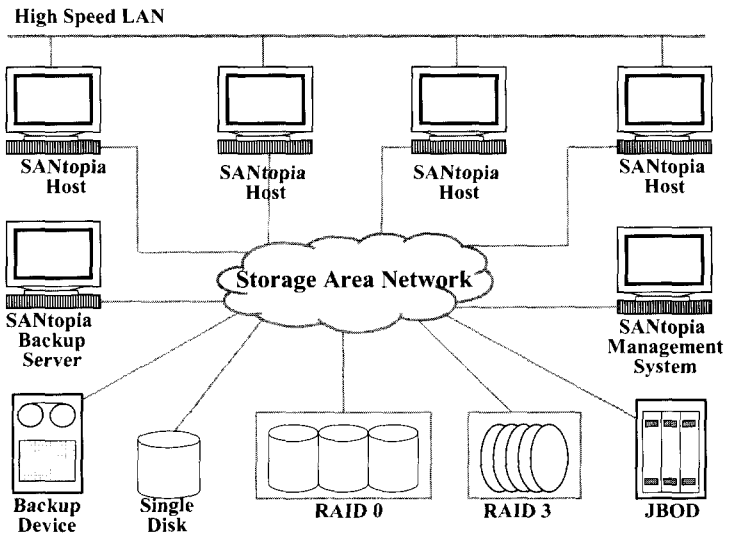
SANtopia가 동작하는 시스템 환경은 [그림 4]와 같다. [그림 4]에 나타낸 바와 같이 SANtopia는 디스크, JBOD, 그리고 RAID 및 백업 장비와 이를 이용하기 위한 호스트들이 화이버 채널로 연결된 SAN 환경에서 각 호스트에서 수행되는 Linux 운영체제에 포함된 시스템 소프트웨어이다. SANtopia는 이 같은 SAN 환경에서 대용량 데이타를 저장할 수 있도록 물리 디스크들을 모아서 저장 풀(storage pool)을 제공하는 논리 볼륨(logical volume)을 지원하고, SAN에 접속된 호스트들 사이에 공유 가능한 대용량 파일 시스템을 지원한다.

여러 호스트들 사이에 공유 가능한 파일 시스템을 제공하기 위해선 호스트들 사이에 통신이 필수적이다. 호스트들 사이에 통신하기 위하여 사용될 수 있는 경로로 두 가지가 가능하다. 첫째는 통신을 위한 경로로 SAN을 이용하는 것이며, 둘째는 서버들을 연결하는 LAN을 이용하는 것이다. 초기 설계 단계에서 전자의 방법을 사용하는 것을 고려하였으나, SAN이 블록 데이타의 전송에 적합하게 만들어져 있을 뿐 아니라 추가적인 프로토콜을 개

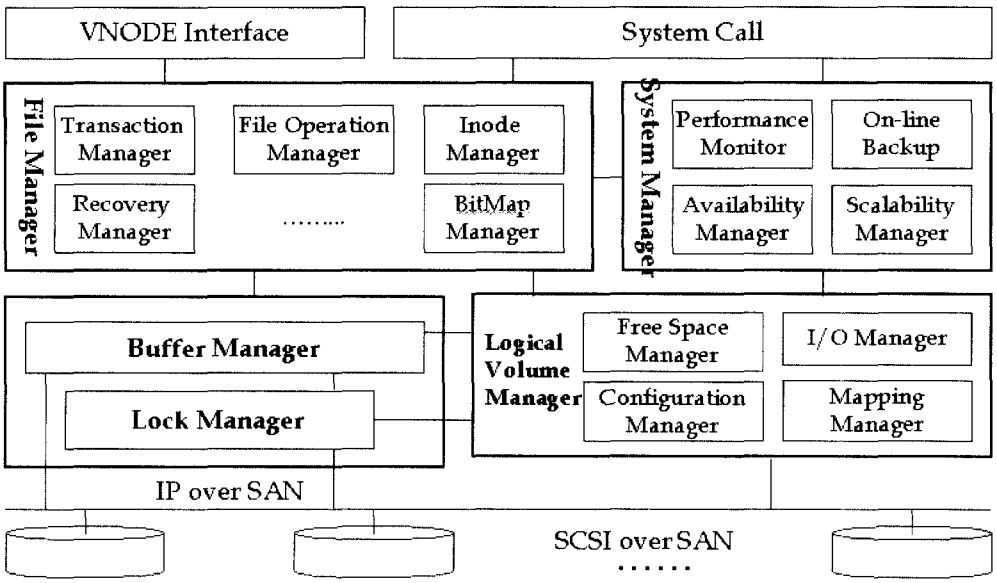
발하여야 하며, 블록 데이타 전송을 방해할 수 있기 때문에 장점보다는 단점이 많은 것으로 판단되어 제외하였다. 따라서 후자의 방법을 사용한다. 후자의 방법은 TCP/IP를 이용할 수 있다는 장점이 있는 반면, LAN의 속도가 성능 저하의 원인이 될 수 있기 때문에 SANtopia는 Gigabit Ethernet 스위치로 연결된 환경에서 운영되는 것을 가정한다.

3.2 SANtopia 시스템 구조

SANtopia는 [그림 5]와 같이 각 기능에 따라 볼륨 관리자, 버퍼 관리자, 잠금 관리자, 파일 관리자로 구성된다. 볼륨 관리자는 SAN에 연결된 모든 디스크를 하나의 관리 대상으로 볼 수 있도록 모은 다음 사용자가 정한 방식에 따라 이것을 분할하여 볼륨을 구성하고 상위레벨의 모듈에서 이 볼륨 위에 I/O를 실행할 수 있도록 해주는 모듈이다. 볼륨 관리자는 SAN에 연결되어 있는 독립적인 디스크 장치들을 모아 하나의 커다란 저장공간 풀을 형성하고 사용자의 요구에 따라 적절히 공간을 분할하여 볼륨을 형성하거나 공간을 회수 추가하여 볼륨을 변경 또는 삭제할 수 있도록 해주는 구성관리(configuration manager) 모듈, 구성 관리자에 의해 생성된 볼륨에 대해 상위 모듈에서 요구하는 입출력을 실제로 물리적 저장 공간인 디스크 장치 위에 수행하는 I/O 관리자, 물리적 디스크 장치들의 모임으로 구성된 저장 공간을 논리적인 저장 공간으로 가상화하여 사용자에게 동적인 구성환경을 제공하는 매핑 관리자(mapping manager), 그리고 볼륨 관리자가 관리하는 저장 공간의 사용 유무에 대한 정보를 유지하고 상위 모듈에서 저장 공간을 요구/철회시 이를 반영하도록 해주는 자유공간 관리자



[그림 4] SANtopia 시스템 환경



[그림 5] SANtopia 시스템 구조

(free space manager)로 구성된다.

기존 리눅스 시스템들은 각 호스트들이 독립적으로 운용되는데 비해서 SANtopia의 호스트는 클러스터 형태로 운용된다. 클러스터 환경에서 시스템의 성능향상을 위해서는 버퍼의 통합관리가 필수적이다. 즉, 개별 호스트에 있는 버퍼들을 통합해서 전역 버퍼(Global Buffer)로 관리함으로써 버퍼의 자료 적중률(hit ratio)을 높여야 한다. 각 호스트에 산재해 있는 버퍼에 어떤 내용이 배치되어 있는지 관리하고 이를 바탕으로 호스트 간에 버퍼의 내용을 주고 받음으로써 전체 클러스터의 디스크 접근 횟수를 줄이고 성능을 향상시키는 역할을 담당한다.

잠금 관리자는 다수의 호스트가 동시에 자료를 사용하기 때문에 자료의 일관성(consistency)을 유지하기 위한 잠금 기능을 지원한다. 잠금 관리자는 호스트 간의 자료에 대한 경쟁이 발생할 때 접근 순서를 결정해주는 역할을 담당함으로써 자료의 일관성을 보장함은 물론 데드락 및 기아(starvation) 상황이 발생하지 않도록 하는 기능을 담당한다.

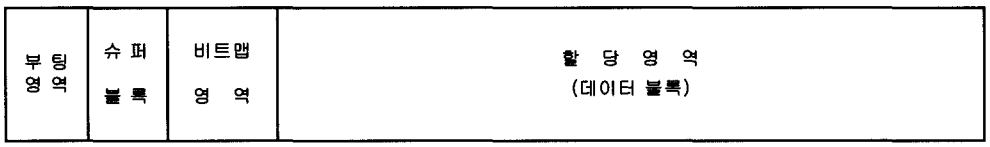
파일 관리자는 대용량 파일을 위한 메타데이타 기능을 지원하고 있으며, SANtopia용 파일 시스템을 위한 슈퍼블록과 파일 및 디렉토리를 위한 inode 등의 메타데이타를 관리하고 사용자가 요구하는 파일 및 디렉토리 등에 대한 각종 연산기능을 제공한다. 또한 저널링(journaling) 기법[14]을 이용하여 파일 시스템의 고장으로부터 메타데이터를 안전하게 회복할 수 있는 기능을 지원한다.

4. 대용량 파일 시스템을 위한 동적 비트맵

4.1 동적 비트맵 파일 시스템의 구조

제안된 파일시스템의 구조는 [그림 6]과 같이 4가지 영역으로 구분된다. 파일 시스템을 부팅을 위한 부팅영역, 파일시스템을 관리하기 위한 정보를 관리하기 위한 슈퍼블록, 객체를 저장하기 위한 할당영역, 그리고 할당영역을 관리하기 위한 비트맵 영역으로 구분된다.

기존의 파일 시스템과의 차이점은 아이노드나 데이터 엔트리 등을 저장하기 위한 별도의 영역을 두지 않고 파일시스템에서 발생되는 모든 객체를 할당영역에 저장하는 것이다. 이를 위해서 제안된 파일 시스템에서는 객체에 따라서 할당영역의 에 요청된 객체를 위한 공간을 할당하고 이를 비트맵을 이용하여 관리하는 기법을 사용한다.



[그림 6] 동적 비트맵을 이용한 파일시스템 구조

4.2 동적 비트맵과 할당영역의 매핑

1) 할당 영역과 비트맵의 구성

제안된 기법의 파일시스템에서는 파일을 관리하기 위하여 디스크를 할당 영역과 이를 관리하기 위한 비트맵 영역으로 구분한다. 각 비트맵은 할당영역을 비트맵 할당의 단위인 할당그룹으로 구분하여 관리하고, 동일한 할당 그룹 내에는 동일한 유형의 파일 시스템 객체를 저장한다. 즉, 비트맵 블록의 크기를 기준으로 한 비트 블록 내에 관리할 수 있는 할당 영역의 크기가 결정되고, 이를 할당그룹으로 사용한다.

예를 들어, 블록의 크기가 1024 바이트이고 할당의 단위가 블록인 경우에 한 비트맵에서 1000개의 할당 영역을 관리하기 위한 비트가 존재하면, 할당 그룹의 크기는 1000이 되고, 동일한 할당 그룹에는 동일한 객체가 저장된다.

비트맵의 크기는 다음의 수식에 의해서 구하고, 파일 시스템의 생성(mkfs) 시에 크기를 결정하여 초기화 작업을 수행한다.

$$BTS = (DS - (SBT + BTS) / (1 - \lceil (BLS - BHS) \rceil)$$

BS : 비트맵의 크기
DS : 전체 디스크 공간의 크기
BTS : 부팅영역의 크기
SBS : 슈퍼블록의 크기
BLS : 블록의 크기
BHS : 비트맵 헤더 정보의 크기

2) 동적 비트맵의 구조

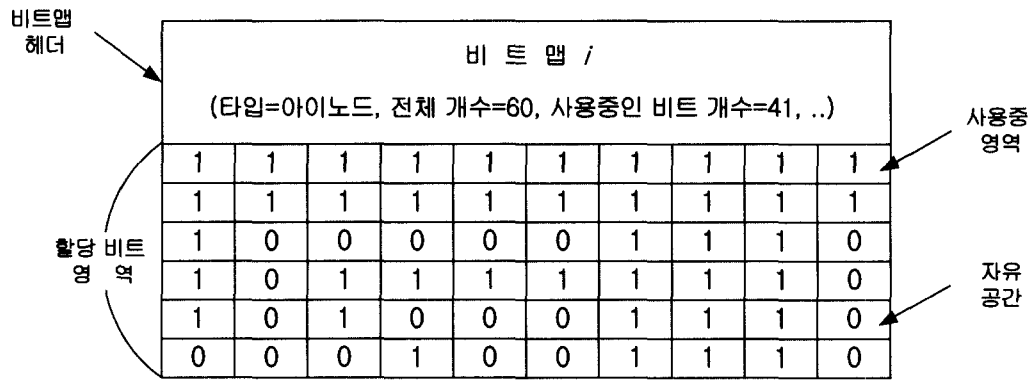
디스크 공간인 할당영역의 할당 여부의 관리를 위한 동적 비트맵의 구조는 [그림 7]과 같다. 각 비트맵의 비트맵 헤더와 할당영역의 사용 유무를 나타내는 할당 비트 영역으로 구성된다. 비트맵을 관리하기 위해서 비트맵 헤더에는 비트맵의 종류, 비트맵이 유지하고 있는 할당 영역의 크기, 그리고 현재 사용중인 할당영역의 개수 등의 정보를 저장한다. 각 할당영역의 사용여부는 할당 영역 비트가 1이면 해당 할당 영역이 사용 중이고, 0이면 사용중이 아님을 나타낸다. [그림 7]에서는 비트맵이 아이노드를 관리하기 위해서 사용중인 i 번째 비트맵이고, 총 60개의 할당 영역을 관리하고 있으며, 현재 41개의 할당 영역이 사용중임을 나타내고 있다.

5. 대용량 파일을 위한 동적 다단계 아이노드 구조

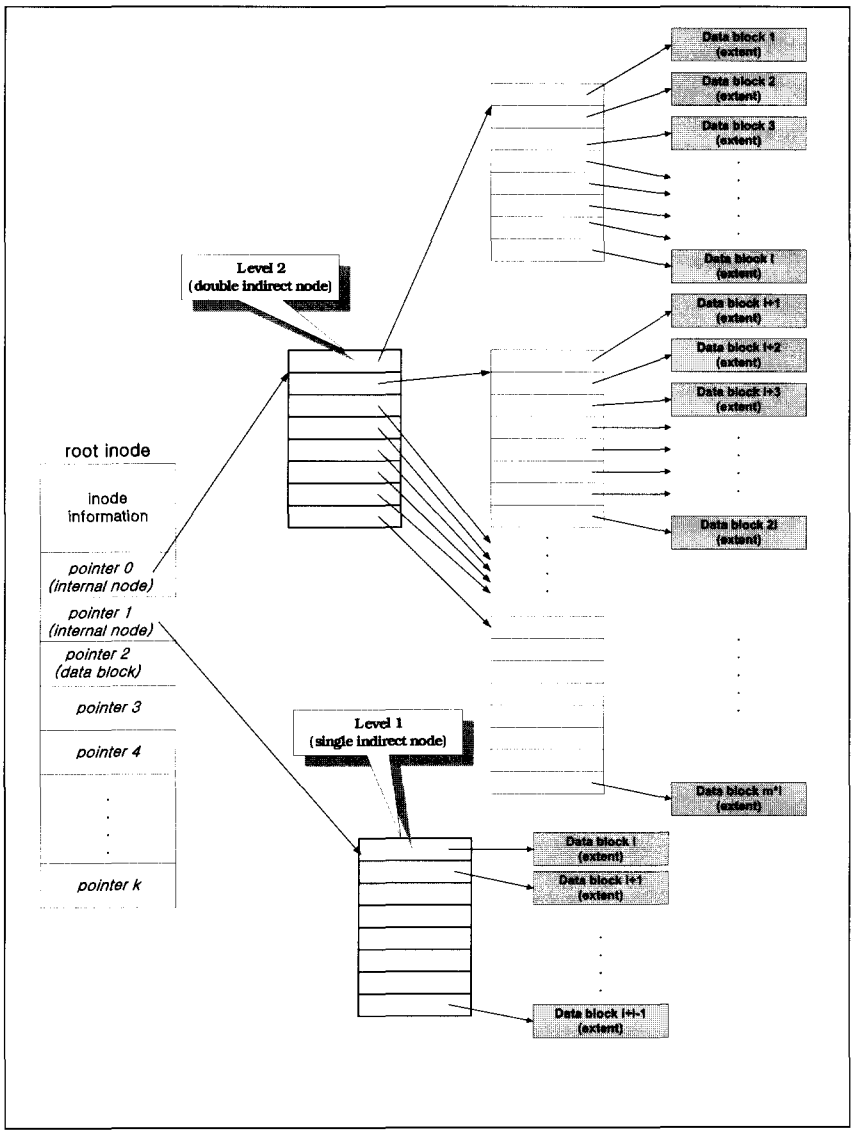
기존의 리눅스의 ext2 파일 시스템에서는 아이노드에 데이터 블록을 위한 포인터를 직접, 간접, 이중, 삼중을 사용하는 고정된 방식의 포인터를 사용하므로 인해 데이터 블록이 1KB인 경우 생성할 수 있는 최대의 파일 크기는 약 16GB정도로 제한된다.

본 논문에서는 동적 다단계 아이노드 구조를 제안한다. 제안된 기법에서는 익스텐트의 사용으로 인한 메모리의 낭비를 줄이기 위한 방법으로 아이노드의 크기를 초과하기 전까지 아이노드 내에 데이터를 저장하는 stuffed inode기법을 사용한다. 또한, 대용량 파일을 저장하기 위한 구조로 [그림 8]과 같이 동적 다단계 아이노드를 사용한다.

제안된 기법에서는 아이노드가 서로 다른 레벨을 가질 수 있다. 파일의 크기가 점점 증가하여 각 레벨이 가득 차게 되면 한 레벨을 증가시켜 맨 위쪽에 이전의 레벨의 포인터를 복사한 후 데이터 블록의 크기가 증가하는 경우에 하나씩 데이터 블록을 위한 포인터의 레벨을 증가시키는 기법을 사용한다. 그림 6의 예에서는 아이노드가 2 레벨의 포인터를 가지고 있으



[그림 7] 동적 비트맵의 구조 예



[그림 8] 동적 다단계 아이노드 구조

며, n 개의 데이터 블록을 위한 포인터를 사용하고 있다.

본 기법을 이용하여 SANtopia에서 동적 다단계 아이노드 구조를 이용하여 대용량 파일을 효과적으로 관리하면서 레벨의 분기 시에 노드의 생성으로 인해 발생되는 성능의 감소와 빈 노드로 인한 메모리의 낭비를 줄이도록 하였다.

6. 성능 평가

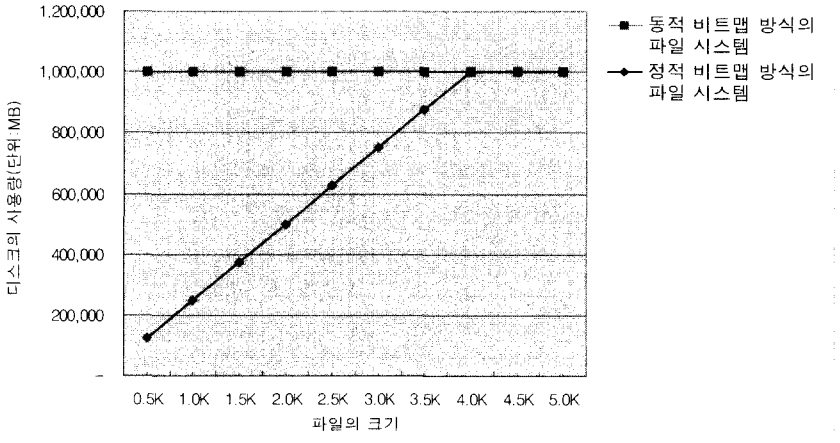
성능의 평가는 기존 정적할당방식을 사용하는 리눅스의 파일시스템(Ext2)과 논문에서 제안하는 동적 비트맵 방식의 파일 시스템을 비교 평가하였다.

<표 1> 성능평가 사용 인자

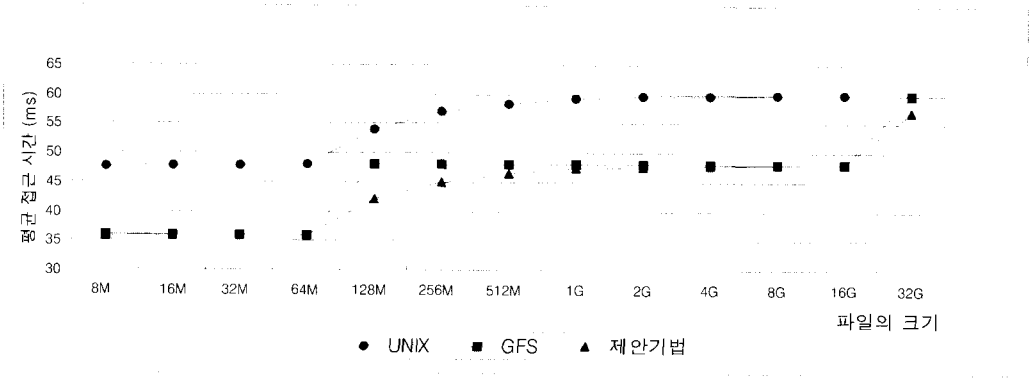
항 목	설 정 값
총 할당 디스크의 크기	1 테라바이트
할당 블록의 크기	4,096 바이트
비트맵 블록의 크기	1,024 바이트
비트맵 헤더의 크기	24 바이트
아이노드 할당 크기	4,096 바이트
비트맵 당 아이노드의 개수 (할당 그룹의 크기)	1,000개
단위 아이노드의 크기 (아이노드 크기)	128 바이트

성능 평가에 사용된 값은 <표 1>과 같다. 파일시스템을 위한 기본영역을 제외한 총 할당 디스크의 크기는 1테라바이트로 설정하였고, 디스크에서 사용되는 비트맵의 크기는 1,024바이트로 관리되며, 각 비트 당 할당되는 할당 영역의 크기는 4,096바이트이다. 비트맵을 관리하기 위한 비트맵 헤더의 크기는 24바이트를 사용하고, 단위 아이노드의 크기는 기존의 파일 시스템에서 사용하는 128 바이트를 사용하였다. 비트맵 블록의 1,024바이트 중에서 24바이트가 헤더로 사용됨으로 한 비트맵 블록은 8,000개의 비트를 이용하여 할당 블록을 관리한다. 따라서 할당 블록의 크기는 32MB이다. 기존의 시스템에 대한 아이노드의 할당은 리눅스 시스템에서 사용하는 4,096바이트 당 1개의 아이노드를 할당하는 것을 적용하였다.

[그림 9]는 파일의 크기에 따라 파일 시스템에서 사용하는 디스크의 사용량을 보여주고 있다. 파일의 크기가 4KB이상인



[그림 9] 파일의 크기에 따른 디스크 사용량



[그림 10] 임의의 블록 접근 시간

경우는 기존의 기법이나 제안된 기법의 차이가 없이 할당 디스크 공간의 모두 사용하지만, 파일의 크기가 적은 경우에는 기존의 기법이 낭비가 심하다. [그림 9]에서 파일의 크기가 0.5KB인 경우에는 전체 1TB중에서 125GB의 공간만을 사용하므로 디스크 공간의 12.5% 만을 사용하지만, 제안된 기법은 파일의 크기에 제안을 받지 않고, 전체 디스크 공간을 이용할 수 있다.

기존의 기법에서는 정적 비트맵으로 인해 생성 가능한 파일의 개수가 제한되므로 디스크 공간의 남아 있음에도 불구하고, 아이노드가 없어서 더 이상의 파일을 생성할 수 없기 때문에 파일의 크기가 적은 경우에는 매우 낮은 효율을 보인다. 특히, 대용량 파일 시스템의 경우에는 낭비되는 디스크의 용량이 전체 용량에서 차지하는 비중이 매우 크게 됨으로 제안된 기법을 적용하는 경우 매우 효율적으로 디스크를 사용할 수 있다.

[그림 10]에서는 파일의 저장하기 위한 아이노드를 비교 평가하였다. 기존의 UNIX, GFS, 그리고 이 논문에서 새로이 제안된 기법의 아이노드 구조에서 파일의 크기에 따른 임의의 데이터 블록에 대한 평균 접근 시간을 비교하였다.

파일 크기가 증가함에 따라, 제안된 기법이 GFS보다 천천히 완만하게 증가하고 있음을 그래프를 통해서 확인할 수 있다. 이것은 임의의 데이터 블록 접근 시간 면에서 플랫 구조보다 우수하다는 것을 보여준다.

7. 결 론

영상 이미지, 미디오, 오디오 및 텍스트와 같은 방대한 양의 데이터를 저장하고, 이를 효과적으로 관리하기 위한 지리정보 시스템에서는 대용량 저장장치 뿐만 아니라 대용량 파일들을 효과적으로 저장하기 위한 대용량 파일시스템이 필수적이다.

본 논문에서는 파일 시스템에서 파일을 유지하고 관리하기 위해 슈퍼블록, 아이노드, 그리고 디렉토리엔트리 등의 메타

데이터를 고정된 영역에 할당하고 이를 비트맵을 이용하여 관리하는 정적 비트맵 방식에서 발생되는 디스크 공간의 낭비를 막기 위한 새로운 동적 비트맵 할당 기법을 제안하였다. 제안된 기법은 메타데이터를 위한 비트맵을 동적으로 할당하여 저장공간의 효율적으로 사용할 수 있으므로 파일의 크기에 따라 디스크의 이용률이 좌우되는 기존 문제점을 해결하였다. 또한, 대용량 파일을 저장하기 위하여 다단계 아이노드 구조를 이용함으로써 GFS의 플랫 구조에서 파일의 크기가 커짐에 따라 급격히 증가하던 데이터 블록의 임의 접근시간을 단축하도록 하였다.

성능평가를 통해서 기존의 기법이 파일의 크기가 0.5KB인 경우에는 전체 1TB중에서 125GB의 공간만을 사용하므로 디스크 공간의 12.5% 만을 사용하지만, 제안된 기법은 파일의 크기에 제안을 받지 않고 전체 디스크 공간을 이용할 수 있으므로 지리정보 시스템을 위한 대용량 파일 시스템에 적합함을 보였다.

참고문헌

[1] S. Aronoff, "Geographical Information Systems: A Management perspective," WDL Publication, pp.151-188, 1989.

[2] R.G. Healey, "Geographical Information Systems - Principales and Applications," Longman Scientific and Technical, pp.251-267, 1991.

[3] "Stroage Area Network(SAN) Solutions," White paper, DELL, 1999, http://www.dell.com.

[4] Clit Jurgens, "Fibre Channel: A Connection to the Future," IEEE Computer, vol. 28, no. 8, pp. 82-90, August 1995.

[5] Maurice J. Bach, The Design of the UNIX Operating System, Prentice-Hall, 1986.

[6] Moshe Bar, Linux File Systems, McGraw-Hill, 2001.

[7] Werner Vogels, "File system usage in Windows NT 4.0," Proceedings of the 17th ACM symposium on Operating systems principles, Charleston, South Carolina, United States, pp.93-109, 1999.

[8] Gregory R. Ganger et al., "Soft updates: a solution to the metadata update problem in file systems," ACM TOCS Vol.18, Issue 2, pp.127-153, May 2000.

[9] Gregory R. Ganger and Yale N. Patt, "Metadata Update Performance in File Systems," USENIX 1994 Operating Systems Design and Implementation Proceedings, Monterey, California, pp.49-60, Nov. 1994.

[10] Y.K. Lee, S.W. Kim, G.B. Kim, and B.J. Shin, "Metadata Management of the SANtopia File System," ICPADS2001, Korea, pp.492~499, 2001.06.

[11] G.B.Kim, C.S.Kim, and B.J.Shin, "Global File Sharing System for SAN," ICACT 2001, Korea, pp.870-874, 2001.

[12] 신범주, 김경배, 김창수, 김명준, "네트워크 저장 장치를 위한 클러스터 파일 시스템 개발," 정보처리학회지, 8권, 4호, 2001. 7.

[13] Keith A. Smith and Margo Seltzer, "A

Comparison of FFS Disk Allocation Policies," USENIX 1996 Annual Technical Conference, San Diego, CA, pp.15-25, Jan. 1996.

[14] Kenneth W. Preslan, et. al., "Implementing Journaling in a Linux Shared Disk File System," Proceedings of the 17th IEEE Mass Storage Systems Symposium, pp. 351-378, College Park, Maryland, March 2000.

[15] Steven R. Soltis, et. al., "The Global File System," Proceedings of the 5th NASA Goddard Conference on Mass Storage Systems and Technologies, College Park, Maryland, September 1996.