# 대용량 파일시스템을 위한 동적 비트맵

김 경 배[†] · 이 용 주[††] · 박 춘 서[††] · 신 범 주[†††]


## 요 약

본 논문에서는 동적 비트맵 할당 기법을 사용하는 새로운 파일 시스템 기법에 대하여 제안한다. 기존의 파일 시스템들이 슈퍼블록, 아이노드, 디렉토리 엔트리와 같은 메타데이터를 위해 고정된 비트맵 구조의 의존하는 반면, 제안된 기법은 파일 시스템의 특성에 따라 비트맵과 할당 영역을 할당한다. 제안된 기법은 기존의 파일 시스템에서 디스크의 이용율이 파일의 크기에 의존하는 문제를 해결하였다. 성능평가를 통해서 동적 비트맵을 이용한 제안된 파일 시스템이 기존의 기법에 비해 효율적으로 디스크를 사용함을 보였다.


# Dynamic Bitmap for Huge File System

Gyoung-Bae Kim[†] · Yong-Ju Lee[††] · Choon-Seo Park[††] · Bum-Joo Shin[†††]


## ABSTRACT

In this paper we propose a new mechanism for file system using a dynamic bitmap assignment. While traditional file systems rely on a fixed bitmap structures for metadata such as super block, inode, and directory entries, the proposed file system allocates bitmap and allocation area depends on file system features. Our approach gives a solution of the problem that the utilization of the file system depends on the file size in the traditional file systems. We show that the proposed mechanism is superior in the efficiency of disk usage compared to the traditional mechanisms.


**키워드 : 대용량 파일시스템(Huge file system), 동적 비트맵(Dynamic bitmap), 메타데이터(Metadata), 동적 할당(Dynamic allocation)**

## 1. 서 론

파일이란 컴퓨터의 디스크나 테이프와 같은 대용량 기억매체에 어떤 정보들의 모임을 저장하는 장소를 말한다. 기존의 유닉스나 리눅스와 같은 파일 시스템에서는 다중사용자를 위해 디스크에 수천 개의 파일을 저장하고 있으며, 이러한 파일의 효과적인 관리가 매우 중요하다.

컴퓨터가 처음으로 개발된 이래로 파일 시스템과 관련된 연구와 개발은 지속적으로 이루어져 왔고 많은 제품들이 출시되었다. 대표적으로 유닉스 파일시스템(s5fs)[1], 리눅스 파일 시스템(Ext2)[2], 윈도우즈 파일 시스템(NTFS)[3] 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[4-7]. 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. 각 객체에 대하여 비트맵의 1비트를 할당하여 사용하면 1로 설정하고, 사용하지 않으면 0으로 설정한다.

객체를 위한 영역과 비트맵의 크기를 결정하기 위해서는 파일 시스템에 저장되는 파일의 크기가 중요하다. 만약 데이터를 저장할 수 있는 공간이 100GB이고 파일의 크기가 1MB라고 가정하면, 파일 시스템에는 100,000개의 파일을 저장할 수 있으므로 100,000개의 아이노드를 저장할 수 있는 영역을 할당하고, 100,000개의 비트를 저장할 수 있는 아이노드 비트맵 영역을 설정하면 된다. 그러나 파일의 크기가 매우 다양함으로 정확한 크기의 비트맵 영역을 설정하는 것이 어려우므로 유닉스나 리눅스 파일 시스템에서는 4KB 당 1개의 아이노드를 할당하는 방법을 사용한다.

그러나, 이러한 정적인 비트맵 방법은 다음과 같은 두 가지 문제가 발생한다.

첫째, 파일 시스템에 저장되는 파일의 크기가 적은 경우

† 종신회원 : 한국전자통신연구원 컴퓨터시스템연구부 선임연구원
†† 정 회 원 : 한국전자통신연구원 컴퓨터시스템연구부 연구원
††† 정 회 원 : 밀양대학교 컴퓨터·정보통신공학부 교수
논문접수 : 2002년 1월 25일, 심사완료 : 2002년 7월 21일

아이노드의 개수가 모자라게 되어 저장 공간의 낭비가 발생한다. 즉, 파일 시스템에 4KB 미만의 파일이 다량으로 발생하는 경우 저장장치에는 파일을 저장할 수 있는 공간이 존재하지만 파일의 정보를 관리하기 위한 아이노드가 없어서 더 이상의 파일을 저장할 수 없다.

둘째, 파일 시스템에 저장되는 파일의 크기가 큰 경우 아이노드의 낭비가 발생한다. 즉, 파일 시스템에 4KB이상의 파일 다량으로 저장되게 되면 아이노드는 남아 있지만 저장장치에 파일을 저장할 공간이 없어서 더 이상 파일을 저장할 수 없다.

따라서 파일 시스템에 저장되는 파일의 크기를 예측할 수 없으므로 파일의 크기에 기반 한 정적 비트맵 방식은 저장 공간의 낭비나 비트맵의 낭비를 초래한다. 특히, 인터넷의 발전으로 전자메일과 같은 소량의 데이터를 주로 저장해야하는 메일 서버 시스템의 경우는 저장 공간의 낭비를 초래하고, 음성이나 영상과 같은 멀티미디어 데이터를 저장하고 서비스하는 멀티미디어 시스템에서는 비트맵이 낭비된다.

이러한 문제를 해결하기 위한 방법으로 Frangipani[8], XFS[9], GFS[10], VxFS[11], SANtopia[12]와 같은 대용량 파일 시스템 등이 연구되었다. 그러나 이들 대용량 파일 시스템의 경우에는 기존의 정적 비트맵 기법을 사용하므로 이러한 문제를 해결할 수 없다. [13]에서는 동적인 기법에 대하여 연구하였으나 가상 메모리 상에서의 vnode의 효과적인 관리를 다루고 있으며, [14]에서는 디스크 상에서 클러스터 형태를 유지하도록 파일을 할당하는 기법에 대하여 다루고 있다.

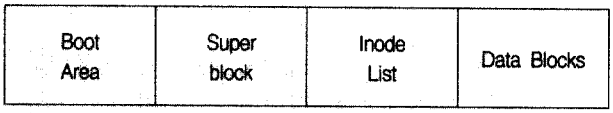
본 논문에서는 메타데이터를 위한 비트맵을 동적으로 할당하여 저장공간의 효율적으로 사용할 수 있는 동적 비트맵 기법을 제안한다. 제안된 기법은 고정된 테이블을 이용하는 기존의 방법과는 달리 파일 시스템에서 발생되는 메타데이터에 따라 동적으로 비트맵을 할당하고 디스크 영역을 관리함으로써 파일의 크기에 따라 디스크의 이용률이 의존되는 기존 문제점을 해결하였다.

본 논문의 구성은 다음과 같다. 제 2장에서 관련 연구로 기존의 파일 시스템에 대하여 설명하고, 제 3장에서는 본 논문에서 제안하는 동적 비트맵을 위한 디스크 할당 구조 및 관리 기법에 대하여 설명하고, 제 4장에서는 성능 평가를 수행하며, 마지막으로 제 5장에서 결론을 맺는다

## 2. 관련 연구

### 2.1 유닉스 파일 시스템

UNIX[2]는 주로 프로그램 개발과 문서 작성 환경을 위한 시분할 운영체제로 Bell 연구소에서 60년대 말 처음으로 발표되었고, 그 이후 계속적으로 기능을 추가하며 새로운 버전을 출시하였다. 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.

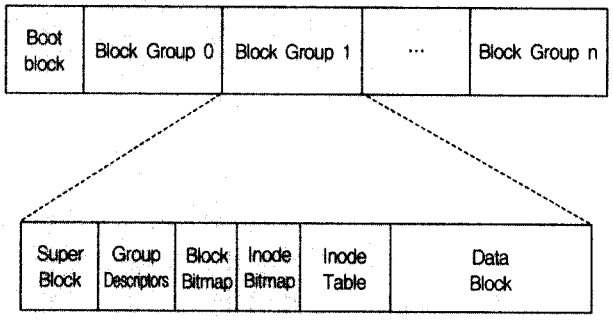



(그림 1) 유닉스 파일시스템(s5fs)의 구조

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 내에 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)[15]의 영향을 많이 받았으며, 그림과 같은 구조를 이루고 있다.




(그림 2) 리눅스 파일시스템(Ext2)의 구조

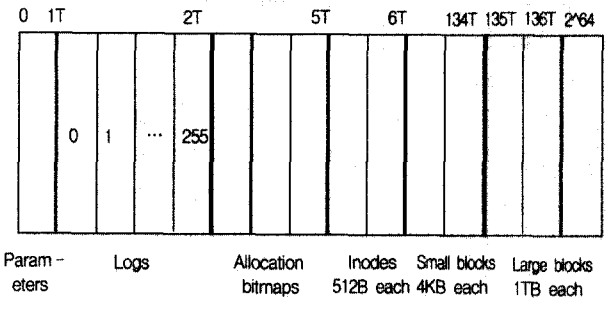
파티션은 FFS에서 실린더 그룹에 대응되는 블록그룹(block group)으로 나누어지며, 각 블록그룹은 파일시스템을 관리하기 위한 슈퍼블록, 그룹을 관리하기 위한 그룹기술자, 데이터 블록의 할당을 관리하기 위한 블록 비트맵, 아이노드의 할당을 관리하기 위한 아이노드 비트맵, 그리고

파일을 저장하기 위한 아이노드 테이블과 파일, 간접블록, 그리고 디렉토리를 저장하기 위한 데이터 블록으로 이루어져있다.

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

### 2.3 Frangipani

Frangipani는 하나의 공유된 저장 공간으로써 다중의 저장장치에 있는 디스크들을 관리하는 새로운 확장 가능한 파일 시스템이다. 이 시스템은 두 레벨 구조로 되어 있다. 아래 레벨에서는 분산 저장 서비스는 확정성과 높은 유용성을 제공하고, 자동적으로 디스크들을 관리한다. 위 레벨에서는 분산 락 서비스를 사용하여 많은 머신들이 운영되고 있다. 이 파일 시스템의 특징은 아주 간단한 내부 구조를 가지고 있다는 것이다.




(그림 3) Frangipani의 파일 시스템 구조

이 시스템의 레이아웃은 (그림 3)과 같다. 맨 처음 1TB 공간에는 공유하는데 필요한 파라메터들이 있고, 다음 1TB 공간에는 256개의 log들을 가지고 있다. 그 다음의 3TB 공간에는 블록 공간을 효율적으로 관리, 할당하기 위한 비트맵이 있다. 그 다음 1TB 공간은 아이노드들을 가지고 있고, 한 디스크 블록 크기를 512B로 하고 있다. 그 다음 6TB에서 134TB까지 공간에서는 4KB 단위의 작은 블록들을 가지고 있고, $2^{64}$까지 1TB 단위의 큰 블록들을 가지고 있다. Frangipani 시스템은 작은 블록과 큰 블록들을 아이노드의 수를 고려하여 적절히 잘 사용함으로써 데이터 블록을 할당하고, 다른 시스템의 레이아웃과 비교해 볼 때, 백업과 파일의 재 저장하는데 드는 비용 면에서 충분히 유용하다.

## 3. 동적 비트맵을 이용한 메타데이터 관리 기법

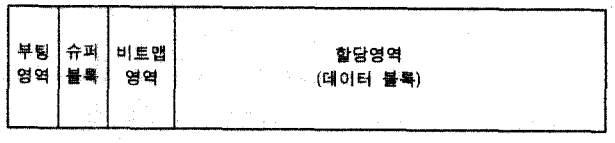
기존의 파일 시스템이 파일시스템 생성 시에 디스크의 일정영역을 아이노드나 데이터 블록을 위한 영역으로 구분하고, 비트맵을 이용하여 관리하는 정적인 비트맵 방식은 파일의 크기에 따라 저장공간의 효율성이 크게 좌우된다.

그러나, 파일 시스템에 저장되는 파일의 크기를 예측할 수 없으므로 파일의 크기에 기반 한 정적 비트맵 방식은 저장 공간의 낭비나 비트맵의 낭비를 초래한다. 특히, 인터넷의 발전으로 전자메일과 같은 소량의 데이터를 주로 저장해야하는 메일 서버 시스템의 경우는 저장 공간의 낭비를 초래하고, 음성이나 영상과 같은 멀티미디어 데이터를 저장하고 서비스하는 멀티미디어 시스템에서는 관리 영역의 낭비된다.

따라서 본 논문에서 제안하는 새로운 비트맵 관리 기법은 파일시스템에 발생되는 메타데이터의 종류에 따라서 디스크 공간을 할당하는 정적인 기법을 사용하므로 인해 대용량 시스템에서 디스크를 효율적으로 사용할 수 있다.

### 3.1 동적 비트맵 파일 시스템의 구조

제안된 파일시스템의 구조는 (그림 4)와 같이 4가지 영역으로 구분된다. 파일시스템을 부팅을 위한 부팅영역, 파일시스템을 관리하기 위한 정보를 관리하기 위한 슈퍼블록, 객체를 저장하기 위한 할당영역, 그리고 할당영역을 관리하기 위한 비트맵 영역으로 구분된다.




(그림 4) 동적 비트맵을 이용한 파일시스템 구조

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 할당영역에 요청된 객체를 위한 공간을 할당하고 이를 비트맵을 이용하여 관리하는 기법을 사용한다.

### 3.2 동적 비트맵과 할당영역의 매핑

#### 3.2.1 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 1024바이트이고 할당의 단위가 블록인 경우에 한 비트맵에서 1000개의 할당 영역을 관리하기 위한 비트가 존재하면, 할당 그룹의 크기는 1000이 되고, 동일한 할당 그룹에는 동일한 객체가 저장된다.

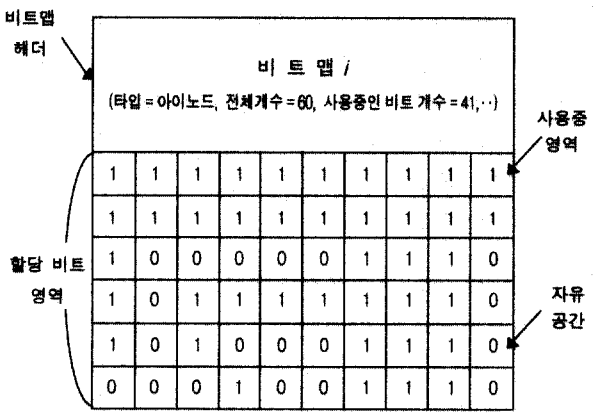
비트맵의 크기는 다음의 수식에 의해서 구하고, 파일 시스템의 생성(mkfs) 시에 크기를 결정하여 초기화 작업을 수행한다.

$$BTS = (DS - (SBT + BTS) / (1 - [ (BLS - BHS)] )$$

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

3.2.2 동적 비트맵의 구조

디스크 공간인 할당영역의 할당 여부의 관리를 위한 동적 비트맵의 구조는 (그림 5)와 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 5)에서는 비트맵이 아이노드를 관리하기 위해서 사용중인 i 번째 비트맵이고, 총 60개의 할당 영역을 관리하고 있으며, 현재 41개의 할당 영역이 사용중임을 나타내고 있다.

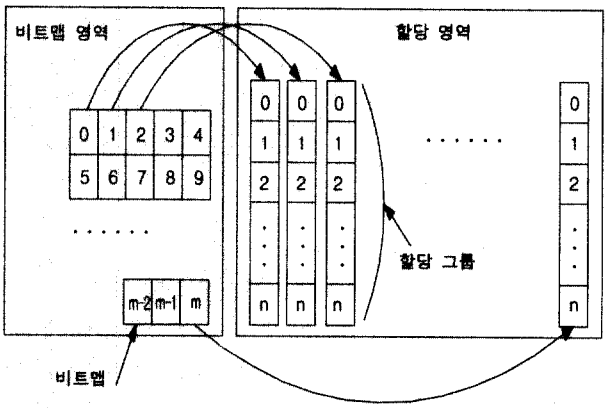



(그림 5) 동적 비트맵의 구조 예

3.2.3 동적 비트맵과 할당 영역의 매핑

동적 비트맵을 이용한 할당 영역의 매핑을 (그림 6)과 같다. 각 할당 영역은 비트맵에 의해서 관리될 수 있는 크기에 따라서 할당그룹으로 분할되어 관리되고, 분할된 할당그룹은 비트맵영역의 각 비트맵에 있는 할당 비트 영역에 의해서 관리된다.

(그림 6)에서 하나의 할당 비트맵에서 관리하는 할당영역의 크기는 n + 1 개이고 총 비트 맵의 개수는 m + 1 개 이므로 파일시스템의 크기는 (n + 1)*(m + 1)이 된다.




(그림 6) 비트맵과 할당영역의 맵핑

각 비트맵의 영역은 (그림 5)의 구조로 되어 있으며, 이를 이용하여 할당영역의 비트맵을 관리한다.

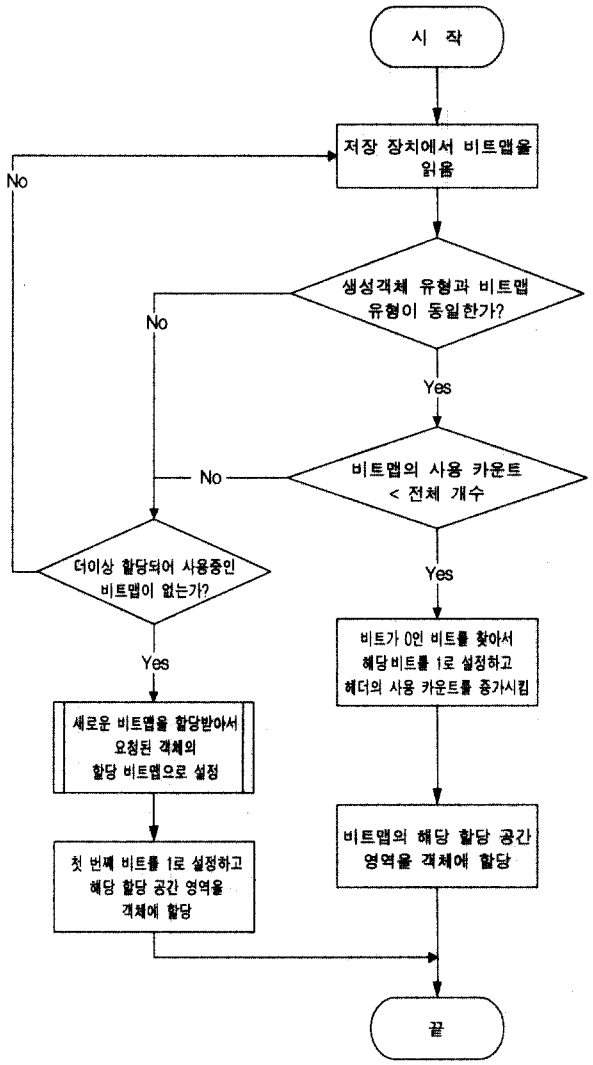
3.3 동적 비트맵을 이용한 메타데이터의 할당 및 삭제 알고리즘

3.3.1 메타데이터의 할당

(그림 7)은 할당 연산 과정을 수행하는 흐름도로서, 저장 장치에서 비트맵을 읽어, 해당 비트맵의 헤더 정보에서 유형 값을 읽어서 파일 시스템에서 요청된 객체 유형과 동일한가를 판단한다. 만약 유형이 동일하면 해당 비트맵에서 사용하지 않는 비트맵이 존재하는 가를 검사하기 위해 사용 카운트와 전체 개수를 비교한다. 즉, 사용자 카운트와 전체 개수가 동일한 경우는 모든 비트가 사용하고 있음을 의미한다. 사용하지 않는 비트가 있으면 해당 비트를 검색하여 해당 비트를 1로 설정하여 해당 할당 영역이 사용중임을 표시하고, 헤더의 사용자 카운트를 증가시킨 후, 비트맵에 해당하는 할당 공간의 영역을 해당 객체에게 할당한다.

만약, 유형을 검사하는 과정과 비트맵의 사용 카운터를 검사하는 과정을 만족하지 못하면 새로운 비트맵을 저장 장치에서 읽어들여야 한다. 이를 위해서 현재 비트맵 영역에 할당되어 사용중인 비트맵이 없는 가를 검사한다. 사용중인 비트맵이 존재하면 저장 장치에서 새로운 비트맵을 읽어 들이지만, 존재하지 않는 경우에는 현재 할당되어 있는 비트맵에 생성하고자 하는 객체를 위한 비트맵이 존재하지 않으므로 새로운 비트맵을 할당받아야 한다.

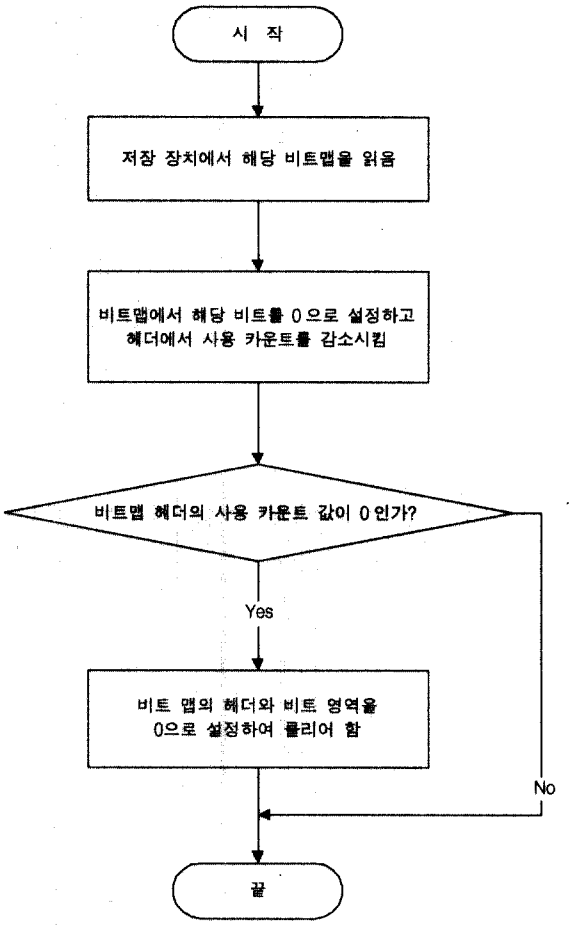
비트맵 영역을 검색하여 사용하지 않는 비트맵을 선택하여 요청된 비트맵으로 유형의 설정하고, 새롭게 할당된 비트맵의 첫 번째 비트를 할당하여 첫 번째 비트맵에 해당하는 할당 공간 영역을 요청된 객체에게 할당한다.




(그림 7) 동적 비트맵을 이용한 메타데이터 할당 알고리즘

3.3.2 메타데이터의 삭제

(그림 8)은 비트맵에서 객체를 삭제하는 과정의 흐름도이다. 파일 시스템에서 객체에 대한 삭제 요청을 받게 되면 요청된 객체가 존재하는 비트맵을 저장장치에서 읽어들인다. 읽어들인 비트맵에서 해당 비트를 찾아 비트 값을 0으로 설정하여 다음에 사용 가능하도록 설정하고 헤더정보에서 사용카운터를 감소시킨 후, 비트맵의 헤더 사용카운터의 값이 0인가를 검사한다. 만약, 헤더 사용카운터가 0이면 현재 비트맵이 관리하고 있는 저장 장치의 할당 영역을 사용하지 않는 것이므로 해당 비트맵을 0으로 설정하여 다른 객체를 위한 영역으로 할당 할 수 있도록 한다.




(그림 8) 동적 비트맵을 이용한 메타데이터의 삭제 알고리즘

## 4. 성능평가

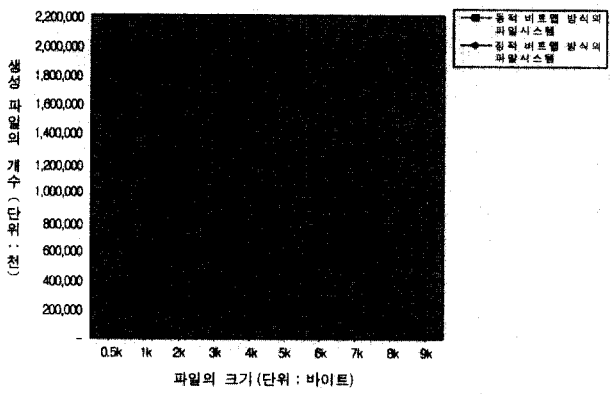
성능의 평가는 기존 정적할당방식을 사용하는 리눅스의 파일시스템(Ext 2)과 논문에서 제안하는 동적 비트맵 방식의 파일시스템을 비교 평가하였다.

〈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 1테라바

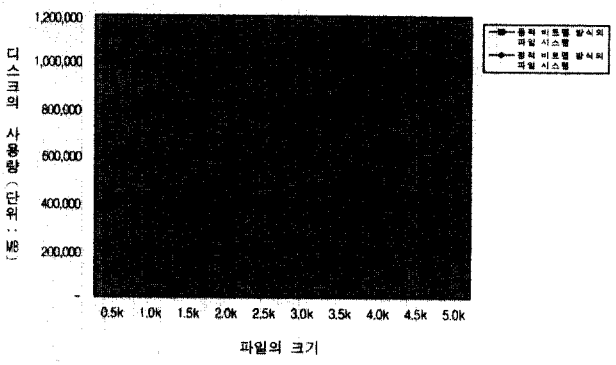
이트로 설정하였고, 디스크에서 사용되는 비트맵의 크기는 1,024바이트로 관리되며, 각 비트 당 할당되는 할당 영역의 크기는 4,096바이트이다. 비트맵을 관리하기 위한 비트맵 헤더의 크기는 24바이트를 사용하고, 단위 아이노드의 크기는 기존의 파일 시스템에서 사용하는 128바이트를 사용하였다. 비트맵 블록의 1,024바이트 중에서 24바이트가 헤더로 사용됨으로 한 비트맵 블록은 8,000개의 비트를 이용하여 할당 블록을 관리한다. 따라서 할당 블록의 크기는 32MB이다. 기존의 시스템에 대한 아이노드의 할당은 리눅스 시스템에서 사용하는 4,096바이트 당 1개의 아이노드를 할당하는 것을 적용하였다.




(그림 9) 파일 크기에 따른 생성 가능 파일의 개수

(그림 9)는 파일의 크기에 따른 생성 가능한 파일의 개수를 보여주고 있다. 기존의 정적 기법을 사용하게 되면 4,096바이트 당 1개의 파일을 생성할 수 있으므로 파일의 최대 생성 가능한 파일의 개수는 250,000,000이다. 따라서 (그림 9)와 같이 4KB 파일의 크기가 이상인 경우는 기존의 정적 비트맵 방식과 제안된 동적 비트맵 방식과 동일하지만, 파일의 크기가 작은 경우에는 본 논문에서 제안된 동적 기법이 월등히 많은 파일을 생성할 수 있음을 보이고 있다. 즉, 파일의 크기가 1KB인 경우에 기존의 기법에서는 250,000,000개의 파일만을 생성할 수 있지만, 제안된 기법에서는 8배 많은 1,000,000,000개의 파일을 생성할 수 있다. 따라서 제안된 기법을 사용하면 대용량 파일시스템의 경우 기존의 기법에 비해 대량의 파일을 생성할 수 있다.

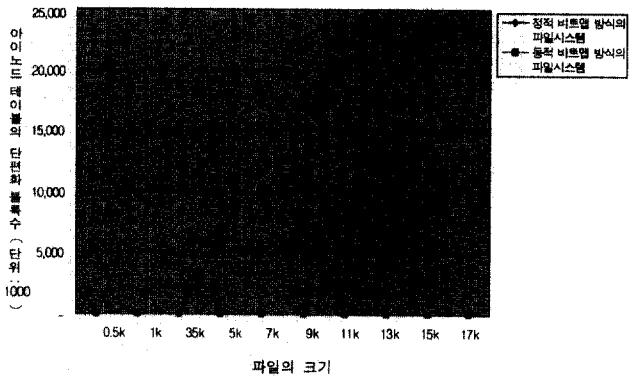
(그림 10)은 파일의 크기에 따라 파일 시스템에서 사용하는 디스크의 사용량을 보여주고 있다. 파일의 크기가 4KB 이상인 경우는 기존의 기법이나 제안된 기법의 차이가 없이 할당 디스크 공간의 모두 사용하지만, 파일의 크기가 적은 경우에는 기존의 기법이 낭비가 심하다. (그림 10)에서 파일의 크기가 0.5KB인 경우에는 전체 1TB중에서 125GB의 공간만을 사용하므로 디스크 공간의 12.5% 만을 사용하지만, 제안된 기법은 파일의 크기에 제안을 받지 않고, 전체 디스크 공간을 이용할 수 있다.




(그림 10) 파일의 크기에 따른 디스크 사용량

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 11) 파일 크기에 따른 단편화된 아이노드 블록

(그림 11)은 파일 크기에 따른 아이노드의 단편화 현상을 보여준다. 기존의 기법에서는 파일 시스템을 생성할 때, 아이노드를 위한 공간인 아이노드 테이블 영역을 미리 할당하기 때문에 파일의 크기가 커지는 경우 이미 할당된 아이노드의 공간이 낭비된다. (그림 11)에서와 같이 파일의 크기가 11KB 인 경우에 약 90,000,000개 정도의 아이노드를 사용하지 않으므로 약 20GB의 아이노드 테이블이 낭비된다. 반면, 제안된 기법에는 1개의 비트맵이 관리하는 할당 그룹 만이 낭비될 수 있으므로 최대 단편화 블록 수는 할당그룹의 블록 수에 의해서 결정된다. 성능평가에서 한 개의 비트맵의 관리하는 할당 블록의 수가 1000개이고, 할당의 단

위가 4,096이므로 최대 단편화 블록 수는 4,000블록으로 약 32,000개의 아이노드가 낭비된다.

따라서, 기존의 기법은 정적인 비트맵 방식은 파일시스템의 생성 시에 디스크 공간을 정적으로 할당함으로 인해 파일의 크기가 작은 경우에는 디스크 공간이 남아 있음에도 불구하고, 이미 모든 아이노드가 사용되었으므로 아이노드 부족의 문제가 발생하여 디스크 공간의 낭비가 발생한다. 반대의 경우에도 이미 아이노드, 디렉토리 엔트리와 같은 메타데이터를 위해 할당된 공간을 사용하지 않으므로 디스크 공간의 낭비가 발생한다.

## 5. 결 론

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

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

## 참 고 문 헌


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

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

[3] 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.

[4] Gregory R. Ganger et al., "Soft updates : a solution to the metadata update problem in file systems," ACM TOCS Vol.18, Issue2, pp.127-153, May, 2000.

[5] 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.

[6] Y. K. Lee, S. W. Kim, G. B. Kim, and B. J. Shin, "Metadata Management of the SANtopia File System," ICPADS 2001, Korea, pp.492-499, Jun., 2001.

[7] 김신우, 이용규, 김경배, 신범주, "대용량 파일 시스템을 위한 메타데이터 구조 설계", 한국정보과학회 추계 학술발표논문집, Vol.27, No.2, pp.59-61, 숙명여자대학교, 제3호, pp.33-42, Mar., 2001.

[8] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee, "Frangipani : a scalable distributed file system," Proceedings of the sixteenth ACM symposium on Operating systems principles, Saint Malo, France, pp.224-237, 1997.

[9] http://oss.sgi.com/projects/xfs/.

[10] 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.

[11] http://www.veritas.com/.

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

[13] Ronald E. Barkley, T. Paul Lee, "A Dynamic File System Inode Allocation and Reclaim Policy," Proceedings of the Winter 1990 USENIX Conference, Washington, D. C., USA, pp.1-10, January, 1990.

[14] 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.

[15] Marshall K. McKusick, William N. Joy, Samuel J. Leffler, Robert S. Fabry, "A fast file system for UNIX," ACM TOCS, Vol.2, No.3, pp.181-197, Aug., 1984.



**김 경 배**

e-mail : gbkim@etri.re.kr

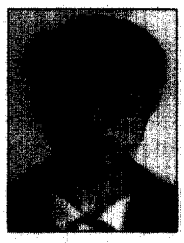
1992년 인하대학교 전자계산학과(학사)

1994년 인하대학교 전자계산학과(석사)

2000년 인하대학교 전자계산학과(박사)

2000년~현재 한국전자통신연구원(컴퓨터시스템연구부 선임연구원)

관심분야 : 자료저장시스템, SAN, NAS, 이동컴퓨팅, 실시간데이터베이스시스템 등


**이 용 주**

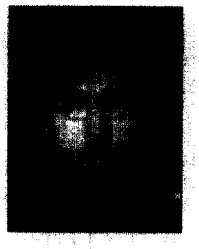
e-mail : yongju@etri.re.kr

1999년 전북대학교 컴퓨터공학과(학사)

2001년 전북대학교 컴퓨터공학과(석사)

2001년~현재 한국전자통신연구원(컴퓨터시스템연구부 연구원)

관심분야 : 하부저장 시스템, 네트워크 스토리지, 분산파일시스템, 멀티미디어 데이터베이스 등



**박 춘 서**

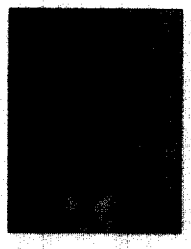
e-mail : parkcs@etri.re.kr

1999년 충북대학교 전기전자공학부(학사)

2001년 충북대학교 정보통신공학과(석사)

2001년~현재 한국전자통신연구원(컴퓨터 시스템연구부 연구원)

관심분야 : 자료 저장 시스템, SAN, 병렬 처리 시스템, 고차원 색인구조 등



**신 범 주**

e-mail : bjshin@mnu.ac.kr

1983년 경북대학교 전자공학과(학사)

1991년 경북대학교 컴퓨터공학과(석사)

1998년 경북대학교 컴퓨터공학과(박사)

1987년~2002년 한국전자통신연구원 (책임연구원, 시스템S/W연구팀장)

2002년~현재 밀양대학교 컴퓨터.정보통신공학부교수

관심분야 : 분산시스템, 고장감내 미들웨어, 이동컴퓨팅, 스토리지 클러스터 S/W 등