실시간 재생 서비스를 위한 비트맵 방식의 연속 블록 할당 기법

박 기 현†

요 약

본 논문에서는 파일 데이터의 실시간 재생 서비스를 제공하기 위한 UNIX 파일 시스템의 연속 블록 할당에 대하여 다루고 있다. 설계된 블록 할당 방식은 사용자가 저장 데이터와 함께 지정한 소비 전송율을 지원할 수 있도록 적절한 디스크 위치에 데이터를 배치하여 실시간 재생 서비스를 지원할 수 있도록 하고 있다. 이를 위하여 데이터 전송율에 영향을 미치는 요소 중에서 가변적인 특성인 연속 블록 수와 저장 데이터들의 실린더 간격과의 관계를 분석하여 특정 실린더 간격 별로 연속 블록을 저장하는 방식을 채택하였다.

UNIX 시스템의 블록 할당 방식은 임의의 실린더 위치에서 연속 블록을 찾는 것이 비효율적이기 때문에 새로운 형태의 비트맵 방식의 유휴 블록 기법을 사용하였다. 비트맵 방식의 블록 관리 기법은 파일 시스템 블록들을 비트 단위로 사용/비사용 여부를 표시하여 임의의 실린더 위치에 존재하는 블록이 사용 가능한지를 효율적으로 검색할 수 있도록 하였다.

A Bitmap-based Continuous Block Allocation Scheme for Realtime Retrieval Service

Kihyun Park†

ABSTRACT

In this paper we consider continuous block allocation scheme of UNIX file system to support real time retrieval service. The proposed block allocation scheme is designed to place real time data at appropriate disk block location in considering the consume-rate that is given with real time data. To effectively determine the disk block location we analyze the relationship between consume-rate and the two variable factors that are the number of continuous blocks and the cylinder distance of logically consecutive data.

In traditional UNIX block allocation scheme it is in fact impossible to find continuous free disk blocks in a specific cylinder location. Thus we propose new bitmap-based free block allocation scheme that enables to determine whether a block in specific cylinder location is free state, or not.

Key words: Real Time Data, Free Block Management, Continuous Disk Block Allocation, File System

1. 서 론

멀티미디어 데이터의 특징은 여러 가지가 있지만 그 중에서 가장 기본이 되는 것은 실시간 데이터의 지원 여부이다. 인터넷 환경에서 클라이언트와 서버 사이의 실시간 데이터 전송을 지원하기 위해서는 기능적으로 세가지 부분으로 나누어 고려할 수 있다. 즉, 서버에서의 데이터 저장 장치의 실시간 재생, 클라이언트로 데이터 전달 과정에서의 실시간 전송, 클라이언트/서버 운영 체제의 실시간 처리 지원 등이 필요하다.

일반적으로 하드 디스크에 저장된 데이터를 실시간으로 처리하기 위해서 디스크 스케줄링 알고리즘을 개선하는 방식으로 많은 연구가 진행되어 왔다. 이러한 예로는 SSTF(Shortest Seek Time First)[5],

† 위덕대학교 정보통신공학과

SCAN[5], Preseeking Swap[3] 등의 알고리즘이 있으며, 주로 하드 디스크의 블록 데이터를 읽어 들일 때 블록의 처리 순서를 조정하여 탐색 시간을 최소화하는 방식을 사용하고 있다. 또 다른 접근 방식으로는 임의의 정해진 시간 내에 데이터가 읽혀 들일 수 있도록 한계 시간[5]을 정해 놓고 알고리즘을 작성하는 방법들이 있다. 이러한 방식들은 모두 처리할 데이터가 하드 디스크에 저장된 상태에서 원하는 데이터를 실시간으로 처리하기 위한 디스크 스케줄링 알고리즘에 관한 내용들이다.

데이터 저장 관점에서 탐색 시간을 줄이기 위한 연구 중에는 미러(mirrored) 디스크를 이용해서 이중으로 복사(replication) 하는 방안이 있다[11]. 이 논문에서는 두 개의 디스크에 어떠한 방법으로 데이터를 배치하는 것이 효율적인지를 분석적인 방법과 추정(heuristic)적인 방법으로 구분하여 다루고 있다. [12]에서는 실시간 요구 조건을 완화시키는 형태의 실시간 지원 방안을 연구하고 있다. 시스템 버퍼의 가변적인 특성이 고려되었으며, 소리와 영상 데이터가 인간의 귀와 눈으로 전달되는 과정에서 어느 정도의 데이터 손실이 발생해도 문제가 없다는 점이 고려되고 있다.

본 논문에서는 멀티미디어 데이터의 저장 장치인 하드 디스크에서의 실시간 재생에 대한 내용을 다루고 있지만 실시간의 재생의 요구 조건을 만족시키는 방안을 재생 관점의 접근 방법보다는 저장 관점에서 다루고 있다. 즉, 데이터를 저장하는 과정에서 재생 시의 실시간 요구 조건을 만족시킬 수 있도록 디스크 블록의 위치를 사전에 조정하는 방식을 사용하고 있다. 특히 데이터의 배치 과정에서 연속적인 블록 할당 원리를 적용하여 대용량의 실시간 파일 지원과 비실시간 파일을 동시에 지원할 수 있는 환경을 가정하였다.

파일 데이터 처리를 위한 연속 블록의 할당은 하드 디스크의 scatter-gather[6] 기능을 이용하여 구현되는데, 시스템 성능 측정 도구를 이용한 파일 시스템 성능 측정 시에 유리한 결과를 보인다. 그러나 일반적인 사용자 환경에서는 버퍼 관리의 문제와 또 다른 소프트웨어 오버헤드로 인하여 적용되지 않고 있다.

하드 디스크의 내용이 모두 실시간 데이터이고 이들이 항구적으로 디스크에 존재하는 경우에는 지극히 단순한 형태로 데이터 블록을 배치시킬 수 있다. 그러나 파일들이 계속적으로 생성되고 삭제되면서 비실시간 파일과 실시간 파일들의 서비스가 함께 이루어지는 일반적인 환경을 지원하는 것은 손쉬운 문제가 아닐 것이다. 본 논문에서는 이와 같은 다양한 환경에서 실시간 파일 서비스를 제공하기 위하여 어떻게 데이터들이 배치되어야 하는지에 대한 분석을 하고 이를 토대로 UNIX[6] 시스템에서의 디스크 유휴 블록을 효과적으로 관리하기 위한 방법론과 알고리즘을 제시하고 있다.

하드 디스크의 성능을 관리하기 위해서는 앞에서 언급된 것처럼 디스크 탐색 시간을 제어해야 한다. 재생 과정에서 탐색 시간을 제어하지 않고 저장 과정에서 이를 미리 조정하기 위해서는 논리적으로 이웃하는 파일 데이터들의 실린더 간격을 원하는 위치에 배치시키는 것이 중요하다. 또한 디스크의 회전 시간도 가변적인 성능 요소의 하나인데, 회전 속도를 예측 가능한 요소로 변경하여 고정 상수로 관리하기 위해서는 동일한 실린더 위치에서 이웃하게 위치시켜야 한다. 즉, 데이터 저장 과정에서 고려되어야 하는 두 개의 변수에는 파일 데이터 블록들이 어느 정도의 실린더 간격을 유지하느냐에 대한 내용과 동일한 실린더 내에서 연속적으로 몇 개의 블록이 이웃한지를 제어해야 한다.

논문에서는 디스크 재생율과 이 두 변수의 관계를 다른 디스크 특성 파라미터와 함께 분석하고, 고정 재생율을 갖는 실시간 재생 서비스를 지원하기 위해서 실린더 간격과 연속 블록의 수를 어떻게 제어하면서 디스크 블록을 배치시켜야 하는지를 분석하고 있다.

실린더 간격을 결정하는 것은 단순한 산술 계산에 의하여 얻을 수 있지만, 특정한 실린더 위치에 필요한 연속 블록이 있는지는 유휴 블록들의 상태를 확인해야 알 수 있다. 기존 UNIX 시스템에서의 블록 할당 방식은 이러한 확인 과정이 사실상 불가능하다. 따라서 새로운 형태의 블록 관리 기법이 필요하며 본 논문에서는 개별적인 블록들을 비트 단위의 비트맵 형식으로 관리하는 방식으로 변경하였다. 이 방식을 사용하면 특정한 위치의 특정 블록이 사용 중인지 아니면 유휴 상태인지를 손쉽게 확인할 수 있다.

재생율을 높이기 위해서는 실린더 간격이 가능하면 좁은 것이 유리하며 동일 실린더에서는 이웃하는

블록들의 수가 큰 것이 유리하다. 그러나 동적으로 생성/삭제되는 비 실시간 파일들이 공존하는 환경에서는 재생율을 지원하는 범위 내에서 융통성있게 관리될 필요가 있다. 제안된 블록 할당 알고리즘에서는 비트맵 방식으로 관리되는 유휴 블록의 할당 과정에서 실린더 간격을 적절히 결정하여 비실시간 서비스를 동시에 지원할 수 있도록 고려하고 있다. 즉, 사용자가 데이터를 저장하는 과정에서 입력했던 재생율과 동일한 값으로 재생을 하는 경우에는 비실시간 서비스를 제공할 수 없다. 그러나 작은 값으로 재생하는 경우에는 여유 시간이 발생할 수 있으므로 이 시간 동안에 효과적인 비실시간 서비스가 이루어질 수 있도록 블록들의 위치를 결정하고 있다.

2. 저장 메카니즘

2.1 데이터 전송율

탐색 시간에 영향을 주는 변수는 예측 가능한 요소와 예측 불가능한 요소로 구분될 수 있다. 먼저 예측이 불가능한 요소에는 하드 디스크 자체에 내장된 버퍼와 시스템 메모리에서 관리되는 파일 시스템용 버퍼가 존재한다. 이들 버퍼의 역할은 디스크 캐쉬 기능을 수행하며 히트율 (Hit Ratio)이 증가할수록 디스크 접근 속도가 빨라진다. 논문에서는 이와 같은 예측 불가능 요소에 대해서는 다루지 않는다.

예측 가능한 요소에는 다음과 같은 디스크 파라미터들을 다루는 것이다.

- T_b: 한 개의 디스크 블럭을 읽는 시간
- T_{track}: 트렉간 이동 시간
- T_{max}: 가장 안쪽과 바깥쪽 실린더간 이동 시간
- S: 디스크 트렉 수

현재의 디스크 헤드 위치에서 D 만큼 떨어진 실린더 위치에 존재하는 연속 블록을 읽을 때의 전송율은 다음과 같다. N은 연속 블록에 포함된 디스크 블록의 수이다.

전송율(블록/초) =

$$\frac{\text{블럭수}}{\text{읽는시간}} = \frac{N}{NT_b + T_{track} + (D-1)\frac{T_{max} - T_{track}}{S-1}} \quad \text{(식 1)[3]}$$

위의 식에서 T_{max}, T_{track}, S, T_b 등은 디스크별로 변하지 않는 상수이므로 가변적인 변수는 D와 N이 된다. 결과적으로 디스크에서 데이터를 읽는데 걸리는 시간은 하나의 연속 블록 내에 존재하는 블록의 수와 연속 블록들의 실린더 간격에 의하여 영향을 받게 된다. 참고로 위의 요소 이외에 디스크 회전 시간도 전송율에 영향을 주는데, 일반적으로 이 값에 대한 제어는 관련 소프트웨어의 부하가 오히려 성능에 나쁜 영향을 미치기 때문에 제어가 이루어지지 않는다. 논문에서 다루는 연속 블록내의 디스크 블록들은 회전 시간이 자동적으로 0이 된다.

그림 1은 시계 방향으로 세 개의 연속 블록을 읽는 과정을 보이고 있다. 각각의 연속 블록에는 N(i-1)=3, N(i)=2, N(i+1)=4 개의 디스크 블록이 존재하며 디스크 헤드의 현재 위치에서 각각의 연속 블록을 읽을 때의 실린더 간격은 D(i-1)=2, D(i)=3, D(i+1)=4가 된다. 식 1을 이용하여 이들의 전송율을 계산하면 각각 다음과 같이 된다.

$$\text{(i-1)의 전송율} = \frac{3}{3T_b + T_{track} + \frac{T_{max} - T_{track}}{S-1}}$$

$$\text{(i)의 전송율} = \frac{2}{2T_b + T_{track} + 2\frac{T_{max} - T_{track}}{S-1}}$$

$$\text{(i+1)의 전송율} = \frac{4}{4T_b + T_{track} + 3\frac{T_{max} - T_{track}}{S-1}}$$

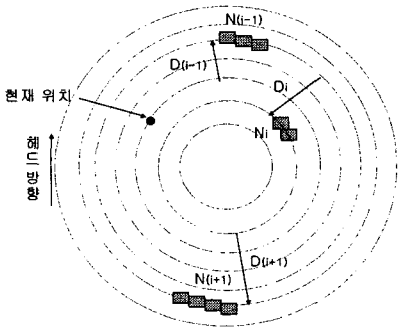

그림 1. 새 개의 연속 블록

식 1에서 디스크 파라미터 상수들이 고정된 경우에 전송율을 증가시키기 위해서는 N을 증가시키거나 D를 감소시켜야 한다. 그림 2는 이를 보이고 있는데, 두개의 고정된 N값인 N_x, N_y 값에 대하여 전송율과 D의 관계를 나타내고 있다. 먼저 N_x와 N_y의 그래프에서 알 수 있는 것처럼 N 값이 큰 경우가 작은

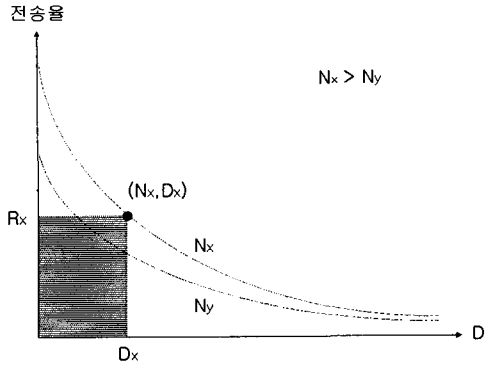

그림 2. 전송율과 D, N의 관계

경우에 비하여 전송율이 높은 것을 알 수 있으며, 동일한 N 값에 대해서는 D 가 작을수록 전송율이 증가된다.

2.2 버퍼의 영향

그림 1에서 세 개의 연속 블록의 전송율이 그림 3과 같다고 가정하자. 예를 들어서 (i-1)번째 블록의 경우는 해당 데이터들을 응용 프로그램에서 사용하기 위해서는 3개의 블록이 모두 읽혀지는 시간인 T(i-1) 시간이 경과된 이후이다. 그러나 일반적인 멀티미디어 응용 프로그램에서는 그림에서 R로 표기된 것과 같은 고정율의 데이터 전송율을 요구하기 때문에 미리 데이터를 버퍼에 읽어 들여야 한다.

일반적으로 각각의 연속 블록들을 읽기 위한 시간과 전송율은 다를 수 있다. 응용 서비스에게 고정 전송율 R로 순차적인 데이터 서비스를 하기 위해서는 다음과 같은 크기의 데이터를 미리 버퍼에 읽어 들어야 한다.

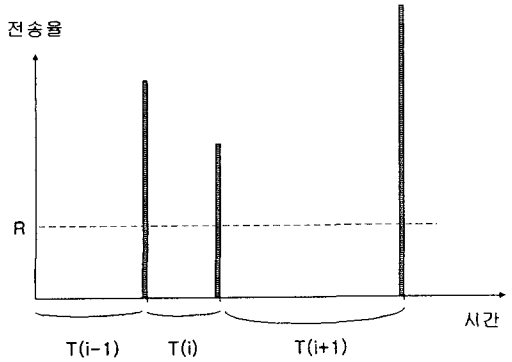

그림 3. 버퍼의 필요성

- $T(i-1) >= T(i)$: $(T(i-1) + T(i)) * R$
- $T(i-1) < T(i)$: $(T(i-1) + T(i)) * R + (T(i) - (T(i-1))) * R$

3. 운영 체제에 대한 고려

3.1 시스템 콜 인터페이스

UNIX 운영 체제에서 지원되는 파일 시스템 관련 시스템 콜에는 open(), read(), write() 등이 있다. 전송율의 표현은 파일 전체에 대한 내용이므로 다음과 같이 open() 시스템 콜에서 전송율을 표시할 수 있어야 한다. oflag의 값이 O_CREAT 인 경우는 파일 생성을 의미하므로 세번째와 네번째 파라미터가 사용된다. mode는 파일의 생성 모드이며, rate (바이트/초) 변수를 이용하여 실시간 파일의 전송율을 지정할 수 있다. 이후 write() 시스템 콜을 이용하여 저장되는 데이터는 적절한 디스크 블록에 위치시킴으로써 rate로 표시된 전송율을 만족시킬 수 있어야 한다.

```
int open(char path, int oflag, [mode_t mode, int
        rate])
```

oflag의 값을 O_RDONLY나 O_RDWR로 지정하여 실시간 파일의 데이터를 읽을 때는 mode는 사용되지 않고 rate를 이용하여 재생율을 지정할 수 있다. 이때 사용되는 재생율의 값은 파일이 생성될 때 지정된 전송율 값보다 작거나 같아야 한다.

3.2 Inode

파일이 생성될 때 지정된 전송율 값은 해당 파일이 존재하는 동안에는 파일과 함께 관리되어야 한다. 따라서 이 정보는 inode[6]에 보관되어야 한다. Inode는 디스크 상에 존재하는 on-disk inode와 메모리 상에 존재하는 in-core inode로 구분되므로 이들 두개의 구조체인 struct dinode와 struct inode에 다음과 같은 필드를 추가해주어야 한다.

```
int rate;     /* 바이트/초 단위의 전송율 */
```

3.3 블록 할당

기존 UNIX 시스템의 블록 할당 방식은 다음과 같이 유휴 디스크 블록들의 번호를 다른 디스크 블록에 보관하는 리스트 구조[6]를 갖고 있으며, 블록 할당

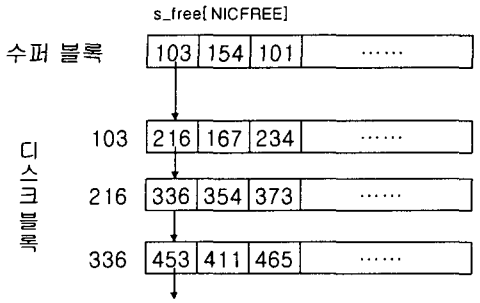

그림 4. UNIX 시스템에서의 유휴 블록 관리

의 속도를 높이기 위하여 일부의 블록 번호들은 파일 시스템의 수퍼 블록에 보관하여 캐쉬 기능이 가능하도록 하고 있다. 수퍼 블록에 보관된 유휴 블록 번호를 모두 사용한 경우에는 블록 번호 103이 데이터 블록으로 사용되고 블록 103에 보관된 번호들은 수퍼 블록으로 복사된다.

참고로 수퍼 블록의 내용들은 파일시스템이 마운트되면 자동으로 메모리로 올라가며 블록의 할당과 회수는 alloc() 함수와 free() 함수에 의하여 이루어진다.

기존의 방식을 이용하는 경우에는 원하는 트렉 간격을 만족하는 연속 블록들을 찾는 것이 용이하지 않으므로 다음과 같이 비트맵 방식의 유휴 블록 관리 형태로 변경하는 것이 필요하다.

먼저 구체적인 예로써 40G 바이트 디스크(헤드=16, 섹터=63, 실린더=79,406) 디스크를 가정하면 이 디스크의 전체 섹터들을 비트맵 형식으로 표시하기 위해서는 16×63×79406 비트가 필요하므로 약 9.5 M바이트의 영역이 필요하다. 그러나 일반적으로 섹터의 크기가 512 바이트인데 비하여 파일 시스템의 블록 크기는 4K 혹은 8K를 지원하므로 블록을 표시하기 위한 비트맵 공간은 이론적으로 1M 혹은 0.5M의 용량으로 가능하다.

8K 시스템을 가정하는 경우에 하나의 실린더에 포함된 블록들의 비트맵 표시를 위해서는(16×63) / (8K / 512) = 63 비트의 공간이 필요하다. 따라서 하나의 블록에는(8K×8) / 63개의 실린더 유휴 블록 정보를 보관할 수 있다. 관리의 용이함을 위하여 이들을 정수화하면 하나의 실린더에 포함된 블록들의 비트맵 정보는 다음과 같이 64비트의 구조체에 의하여 관리되고(마지막 1 비트는 사용되지 않음) 이들 정보들은 79,406개가 존재한다. 따라서 하나의 8K 블록에 (8K×8) / 64 = 1024개씩 총 79406 / 1024 = 78 블록이 사용된다. 따라서 약 624K 의 공간이 필요하게 된다.

double bitmap; /* 하나의 실린더에 포함된 9K 블록용 비트맵 */

최종적으로 유휴 블록을 관리하기 위한 공간의 배치는 다음과 같이 되고 이 정보들은 슈퍼 블록의 바로 다음에 위치하는 것으로 가정한다. 그림에 표시된 것처럼 회색 표시의 64 비트 정보는 하나의 실린더에 포함된 63개의 8K 블록들의 사용/비사용 상태를 나타낸다. 이들이 순차적으로 실린더에 포함된 블록들의 상태 정보를 보관하기 때문에 원하는 실린더 내에 연속 블록이 있는지를 손쉽게 찾을 수 있다.

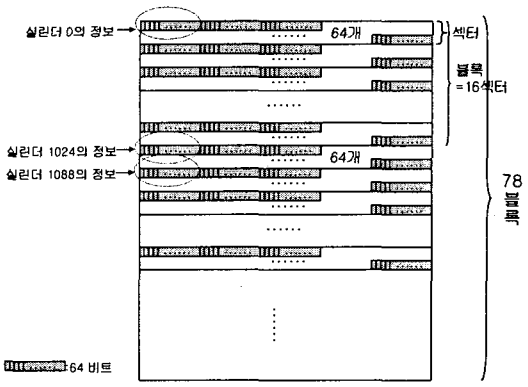

그림 5. 비트맵 방식의 유휴 블록 관리

유휴 블록을 할당하여 데이터를 저장하는 알고리즘은 다음과 같다. 먼저 입력 파라미터에는 저장할 데이터와 함께 전송율이 필요하다. 구성 파라미터인 JUMP는 연속 블록들간의 실린더 간격을 지정하기 위한 것으로 의도적으로 어느 정도의 간격을 갖도록 하였다. 이는 해당 파일을 재생할 때에 저장시에 지정했던 전송율보다 느린 재생을 하는 경우에 여유 시간이 발생하는데, 이 시간동안 중간에 위치한 비실시간 파일의 데이터 블록 서비스를 지원하기 위한 것이다.

실시간 파일을 저장하는 동안에 연속 블록들은 디스크의 한쪽 실린더 방향으로만 이동하며 안쪽이나 바깥쪽의 끝에 도달하면 반대 방향으로 다시 순환하게 되며, 식 1과 유휴 블록의 비트맵을 이용하여 원하는 연속 블록을 찾게 된다. 만일 JUMP 간격의 실린더 위치에서 원하는 블록을 찾을 수 없는 경우에는

```
입력 파라미터: 전송율 R, 저장 데이터
구성 파라미터: 연속 블록 간의 간격 JUMP
  (JUMP>0: 바깥방향 순환, JUMP<0: 안쪽방향 순환)

DataSize = <파일의 크기>;
LastCyl = <초기 실린더 위치>;
while (DataSize > 0) {
   CandidateCyl = CandidateCyl + JUMP;
   if (CandidateCyl > S) {
      CandidateCyl = S(CandidateCyl%S);
      JUMP =-JUMP;
   }
   else if (CandidateCyl < 0) {
      CandidateCyl =- CandidateCyl;
      JUMP =- JUMP;
   }
b:
   NumberOfBlocks = 식1과 |CandidateCyl-LastCyl| 을 이용, R을 만족하는 가장 작은 N을 계산;
   비트맵을 이용하여 CandidateCyl 실린더에 NumberOfBlocks 크기의 연속 블록이 있는지 확인;
   if (블록이 있는 경우) {
      찾은 연속 블록에 데이터를 저장;
      DataSize = DataSize-<연속블럭의 크기>;
      LastCyl = CandidateCyl;
   }
   else {
      CandidateCyl = CandidateCyl+JUMP/| JUMP|;
      Goto b;
   }
}
```

알고리즘 1. 실시간 파일의 저장 알고리즘

이 간격을 줄이면서 알고리즘을 반복하게 된다.

알고리즘의 검증을 위한 시뮬레이션은 UNIX 파일을 가상 디스크로 변환하여 이루어졌으며, UNIX 파일 내용의 4 바이트를 가상 디스크의 한 블록으로 가정하여 이루어졌다. 구성 파라미터인 JUMP 값을 변화시키면서 얻어진 결과는 이 값이 어느 정도 증가하는 과정에서는 디스크 이용 효율이 좋아지지만 일정 크기 이상에서는 효율의 변화가 없음을 알게 되었다. 이는 JUMP 값이 작으면 유휴 블록을 찾기가 어렵기 때문에 효율이 나쁜 것이며, 너무 크면 상대적으로 탐색 시간이 증가되어 효율 증가가 상쇄되는 것이다.

JUMP 값의 음/양으로 표현되는 순환 방향은 임의의 한 방향으로 고정하는 방법과 랜덤하게 방향을 선택하는 방안이 가능한데, 실험 결과는 동일한 것으로 나타났다. 유휴 블록의 정도에 따른 성능 측정에서는 유휴 블록의 개수가 적을수록 당연히 시스템 성능이 나쁜 것으로 나타났다.

4. 결 론

본 논문에서는 실시간 파일을 지원하기 위하여 데이터의 재생율과 파일 시스템 블록의 관계를 고찰하고 이를 UNIX파일 시스템에 적용하기 위한 내용을 다루고 있다. 연속 블록의 할당을 용이하게 하기 위하여 기존의 UNIX 블록 할당 알고리즘을 변형하여 비트맵 형식을 사용함으로써 원하는 실린더 위치에서 임의의 크기의 연속 블록을 쉽게 찾을 수 있도록 하였다.

제안된 알고리즘은 실시간 서비스 과정에서 비실시간 파일에 대한 서비스도 함께 고려되었으며 비트맵 방식의 처리는 다수의 디스크 시스템에서도 확장

이 용이하도록 설계되었다. 비트맵 정보를 보관하기 위한 디스크 공간은 전체 공간에서 1/(파일 시스템 블록 크기) 정도만 사용되므로 그다지 큰 비율은 아니다.

디스크에 저장되는 파일들 중에서 비실시간 파일과 실시간 파일의 비율이 어느 정도 되는지의 여부는 실제로 알고리즘에 많은 영향을 미칠 수 있다. 또한 비트맵을 사용하여 유휴 블록을 관리하게 되는 경우에 비실시간/실시간 파일의 저장 과정에서의 성능 변화여부도 추후에 다루어져야 할 것으로 생각된다.

참 고 문 헌

[1] P. Venkat Rangan, Harrick M. Vin, Efficient Storage Techniques for Digital Continuous Multimedia, IEEE Trans on Knowledge and Data Engineering. Vol. 5, No. 4, Aug, 1993, pp1041-4347.

[2] Nang J, Heo S, An Efficient Buffer Management Scheme for Multimedia File System, IEICE on Information Systems, 2000, pp.1225-1236.

[3] D. James Gemmel, Jiawei Han Delay-Sensitive Multimedia on Disks, IEEE multimedia, 1994, pp.56-67.

[4] Haskin RL, Tiger Shark A Scalable File System for Multimedia, IBM Journal V.42 N.2, 1998, pp.185-197.

[5] Ralf Steinmetz, Multimedia file systems survey: approaches for continuous media disk scheduling, Computer communications Vol. 18, No. 3, Mar. 1995, pp. 133-144.

[6] Uresh Vahalia, UNIX Internals The new Frontiers, Prentice-Hall, 1996.

[7] Jim Gemmel, Stavros Christodoulakis, Principles of Delay-Sensitive Multimedia Data Storage and Retrieval, ACM Transactions on Information Systems, Vol. 10, No.1, Jan. 1992, pp51-90.

[8] Cuneyt Akinlar, Sarit Mukherjee, A Scalable Distributed Multimedia File System Using Network Attached Autonomous Disks, IEEE Proc. on Computer and Communication System, 2000, pp.180-187.

[9] Jesus Carretero et al, Design and Evaluation of a Multimedia Integrated Parallel File System, IEEE Multimedia Systems, 1999, pp.323-327.

[10] Peter Bosch, Sape J. Mullender, Real-Time Disk Scheduling in a Mixed Media File System, IEEE Proc. Of Real-Time Tech. And Appl., 2000, pp.23-32.

[11] Athena Vakali, Yannis Manolopoulos, Data placement schemes in replicated mirrored disk systems, The Journal of Systems and Software 55, pp.115-128, 2000.

[12] James C Yee, Pravin Varaiya, Models and performance of real-time disk access policies, computer communication vol.18, num.10, 1995.

박 기 현

1985년 고려대학교 전자공학과 졸업

1987년 고려대학교 대학원 전자공학과(공학석사)

1998년 고려대학교 대학원 전자공학과(공학박사)

1987년~1994년 삼성전자 시스템 개발실 선임 연구원

1999년~현재 위덕대학교 조교수

관심분야 : 분산 시스템, 이동 컴퓨팅, 운영 체제