가변 크기 버퍼를 이용한 멀티미디어 데이타 버퍼 관리 기법

조 영 섭[†] · 김 재 홍[††] · 배 해 영[†††]

요 약

멀티미디어 데이타에 대한 처리 요구가 급증함에 따라 멀티미디어 데이타를 처리하는 저장 관리자의 성능은 시스템의 성능에 매우 큰 영향을 미친다. 일반적으로 멀티미디어 데이타의 크기는 매우 크기 때문에 멀티미디어 데이타의 디스크 입출력은 많은 시간을 소비하여 시스템 성능 저하를 가져온다. 따라서 멀티미디어 데이타의 버퍼 관리를 통해 디스크 입출력을 줄이는 것은 시스템 성능에 매우 중요하다.

본 논문은 멀티미디어 데이타 저장 관리자가 디스크 영역 관리에 일반적으로 사용하는 세그먼트가 물리적으로 연속적인 페이지들로 구성되어 디스크 입출력 단위로 사용된다는 특성을 고려하여 버퍼의 크기가 세그먼트의 크기에 따라 가변적인 버퍼 관리 기법을 제안한다.

또한 버퍼의 크기가 가변적이 됨에 따라 버퍼의 참조 행동뿐만 아니라 버퍼의 크기를 고려한 버퍼를 교체하는 기법을 제안한다. 제안된 멀티미디어 데이타 버퍼 관리 기법은 멀티미디어 데이타 저장 관리자인 KORED/STORM에서 구현한다.

A Multimedia Data Buffer Management Technique Using Variable Size Buffer

Yeong-Sub Cho[†] · Jae-Hong Kim[††] · Hae-Young Bae[†††]

ABSTRACT

As there has been much demands for processing multimedia data, a storage manager for multimedia data makes much effects on system performance. Because the size of multimedia data is usually very large, disk I/O for the data consumes much time and causes the system performance to be decreased. Therefore, it makes a better effect on system performance that a multimedia data storage manager decreases its disk I/O by the buffer management of multimedia data.

This paper proposes a buffer management technique which allocates the buffer to be equal to its corresponding segment which consists of physically continuous disk page set and is disk management unit for multimedia data in many multimedia data storage manager. As the size of buffer varies, it also proposes a buffer replacement policy which consider not only reference behavior of buffer but also buffer size The proposed multimedia data buffer management technique is implemented in KORED/STORM which is a storage manager for multimedia data.

† 정 회 원:인하대학교 대학원 전자계산공학과 박사과정
†† 정 회 원:영동공과대학 컴퓨터공학과 전임강사
††† 종신회원:인하대학교 전자계산공학과 교수

논문접수:1995년 8월 7일, 심사완료:1996년 8월 12일

1. 서 론

최근 기존의 문자, 숫자, 텍스트와 같은 데이타 뿐만 아니라 멀티미디어 데이타의 처리 요구가 일반화되면서 멀티미디어 데이타를 디스크에 저장 관리하는 멀티미디어 데이타 저장 관리자에 대한 많은 연구가 진행되어 왔다[1,2,4,7,10,12,13,15,17,18]. 멀티미디어 데이타의 디스크 입출력은 많은 시간을 요구하기 때문에 데이타 입출력 요구 때마다 디스크를 접근하는 것은 시스템 성능을 크게 저하시킬 수 있다. 따라서 멀티미디어 데이타를 버퍼로 관리하여 디스크 접근을 줄이는 것은 시스템 성능을 크게 향상시킬 수 있다[1,14].

그러나 버퍼 관리에 대한 연구는 데이타의 크기가 디스크 한 페이지 크기보다 작은 스몰 데이타(small data) 버퍼 관리 기법에 대하여 주로 진행되었으며, 멀티미디어 데이타에 대한 버퍼 관리 연구는 상대적으로 부족했다[2,3,4,5,6,8,11,16]. 이는 멀티미디어 데이타를 관리할 만큼의 메모리 영역의 할당에 큰 어려움이 있었기 때문이었다. 그러나 최근 주기억 장치 데이타 베이스 시스템이 등장할 정도로 메모리의 크기가 커졌기 때문에 멀티미디어 데이타를 위하여 메모리 버퍼 공간을 할당하는 것이 가능해 졌다[14]. 또한 멀티미디어 데이타의 참조가 빈번해지면서 참조된 멀티미디어 데이타의 재 참조 확률은 높아졌고 따라서 멀티미디어 데이타에 대한 버퍼 관리는 디스크 입출력 시간을 크게 줄일 수 있다.

멀티미디어 데이타를 위한 버퍼 관리자는 다음과 같은 조건을 충족시켜야 한다. 첫번째, 대부분의 멀티미디어 데이타 저장 관리자는 멀티미디어 데이타를 페이지단위로 관리하지 않고 물리적으로 연속적인 페이지들로 구성되는 세그먼트(segment)로 관리하고 멀티미디어 데이타에 대한 입출력을 세그먼트 단위로 수행한다. 세그먼트는 한번에 디스크에서 입출력이 가능하므로 멀티미디어 데이타에 대한 빠른 접근을 수행할 수 있게 한다. 멀티미디어 데이타를 위한 버퍼 관리자는 이러한 세그먼트 단위 디스크 입출력을 효과적으로 지원해야 한다. 두번째, 멀티미디어 데이타에 대한 부분 입력 요구를 효과적으로 지원해야 한다. 멀티미디어 데이타를 처리하는 응용 분야에서는 멀티미디어 데이타 전체를 검색하는 연산뿐만 아니라 멀티미디어 데이타의 일부만을 검색하는 연산을 많이 요구한다.

따라서, 본 논문은 세그먼트 단위의 디스크 입출력과 멀티미디어 데이타의 부분 입력 연산을 효과적으로 지원하는 버퍼 관리 기법을 제안한다. 제안된 버퍼 관리 기법은 멀티미디어 데이타의 세그먼트 단위 디스크 입출력을 효과적으로 지원하기 위해 버퍼 영역을 이진 버디 시스템(binary buddy system)[9,18]으로 관리하여 세그먼트 크기에 따라 버퍼를 할당한다. 세그먼트에 대응되는 버퍼들로 멀티미디어 데이타를 버퍼 관리하여 멀티미디어 데이타에 대한 부분 입력시 버퍼 영역의 낭비를 줄인다. 또한 버퍼의 평균 참조 간격을 이용하여 버퍼들을 교체함으로써 참조 지역성(refereence locality)을 반영하는데 성능이 뛰어난 LRU-K[3]를 이용하여 버퍼 교체를 수행하고 버퍼의 빠른 탐색을 위해 버퍼 크기를 고려하여 버퍼들을 관리한다. 즉 버퍼의 평균 참조시간과 버퍼의 크기를 복합적으로 고려한 버퍼 관리 기법을 제안하며 제안된 버퍼 관리 기법을 KORED/STORM(KOrean RElational Database management system/STORage Manager)에서 구현한다[17,18,20].

본 논문의 구성은 다음과 같다. 2장에서는 세그먼트를 기반으로 멀티미디어 데이타를 처리하는 저장 관리자의 버퍼 관리 기법에 대하여 고찰한다. 3장에서는 본 논문에서 제안하는 버퍼 관리 기법에 대하여 논한다. 4장에서는 제안된 버퍼 관리 기법의 설계와 구현에 대하여 논하고, 5장에서 결론과 향후 연구 과제를 서술한다.

2. 관련 연구

2.1 ESM(EXODUS Storage Manager)의 버퍼 관리자

ESM의 버퍼 관리자는 멀티미디어 데이타에서 일부만을 접근하는 부분 입력이 요구되면, 입력의 대상이 되는 부분만을 저장할 수 있도록 버퍼를 할당한다[4]. 예를 들면, 100K Bytes 크기의 멀티미디어 데이타 중에서 10K Bytes 크기의 부분 입력이 발생하면 10K Bytes 크기의 버퍼를 할당하며, 20K Bytes 크기의 부분 입력이 발생하면 20K Bytes의 버퍼를 할당한다. ESM의 버퍼 할당 방식은 멀티미디어 데이타에 대한 부분 입력이 요구될 때 멀티미디어 데이타 전체를 저장할

수 있는 크기의 버퍼 영역이 할당되는 방법에 비해 버퍼 영역의 낭비를 줄인다.

그러나 멀티미디어 데이타 중에서 입력의 대상이 되는 부분만을 저장할 수 있는 크기로 버퍼가 할당됨으로, 부분 입력이 이전의 부분 입력들의 내용과 동일하지 않으면 새로운 버퍼를 할당한다. 이와 같은 방식은 멀티미디어 데이타에 대한 부분 입력 요구가 증가하면 버퍼 영역의 낭비를 가져온다.

또한 ESM의 버퍼 관리자는 LRU와 MRU(Most Recently Used)를 이용하여 버퍼를 교체하는데 일반적으로 LRU와 MRU는 데이타의 참조 지역성을 잘 반영하지 못 한다는 단점을 가지고 있다[3].

2.2 Starburst의 버퍼 관리자

Starburst의 버퍼 관리자는 스몰 데이타에 대해서는 버퍼 관리를 지원하지만 멀티미디어 데이타에 대해서는 버퍼 관리를 지원하지 않는다. 따라서 멀티미디어 데이타에 대한 검색이 여러 번 요구되면 그 때마다 디스크 입출력이 발생하여 시스템의 응답시간이 증가되는 문제가 발생한다[1, 15].

2.3 EOS(Experimental Object Store)의 버퍼 관리자

EOS의 버퍼 관리자는 4K Bytes 크기까지의 멀티미디어 데이타에 대해서만 버퍼 관리를 지원하며 4K Bytes를 초과하는 멀티미디어 데이타는 버퍼로 관리하지 않는다[2]. 그러나 대부분의 멀티미디어 데이타의 크기가 4K Bytes보다 크기 때문에, 멀티미디어 데이타의 버퍼 관리를 통한 디스크 입출력의 감소를 기대할 수 없다.

따라서 제안하는 멀티미디어 데이타 버퍼 관리자는 세그먼트 단위의 디스크 입출력을 효율적으로 관리할 수 있어야 한다. 멀티미디어 데이타에 대한 부분 입력이 요구될 때 버퍼의 중복으로 발생하는 버퍼 영역의 낭비를 방지하고, 많은 멀티미디어 데이타를 버퍼로 관리될 수 있도록 버퍼로 관리되는 멀티미디어 데이타의 한계 크기가 정해져야 한다. 또한 버퍼를 교체할 때, LRU의 성능을 개선한 버퍼 관리 기법을 사용해야 한다.

3. 멀티미디어 데이타 버퍼 관리 기법

3.1 버퍼 할당 기법

디스크 상에서 세그먼트를 기반으로 관리되는 멀티미디어 데이타를 위한 버퍼 할당 방식으로 스몰 데이타 버퍼 관리자에서처럼 디스크 페이지 크기로 고정된 버퍼를 사용하여 관리할 수 있다. 그러나 디스크 페이지 크기로 고정된 버퍼를 이용한 버퍼 관리 기법에서는 멀티미디어 데이타를 디스크에서 읽는데 많은 시간이 소비되는 문제가 발생한다. 예를 들어, 16 페이지로 구성된 세그먼트를 디스크에서 읽어올 경우, 우선 16 개의 버퍼들을 할당하여야 한다. 이것은 디스크에서 한번에 전송될 수 있는 세그먼트를 16 개의 페이지별로 각각 전송해야 하므로 세그먼트를 읽어오는데 많은 시간을 소비하게 된다. 또한 하나의 세그먼트를 16 개의 페이지 크기 버퍼들로 관리해야 함으로 버퍼 관리가 복잡해진다. 따라서 세그먼트를 기반으로 관리되는 멀티미디어 데이타를 위한 버퍼는 세그먼트를 포함할 수 있는 크기로 버퍼 영역 내에서 물리적으로 연속적인 공간에 할당되어야 한다.

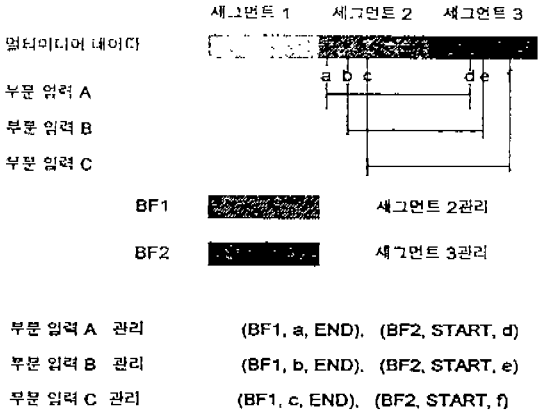
따라서 멀티미디어 데이타를 입력할 때, 멀티미디어 데이타 전체를 포함하도록 물리적으로 연속적인 공간을 버퍼로 할당한다면 세그먼트 단위의 디스크 입출력 효율은 저하되지 않는다. 그러나 이러한 버퍼 할당 방법에서는 멀티미디어 데이타에 대한 부분 입력이 요구되면 불필요한 디스크 입출력이 발생하며 버퍼 영역을 낭비하는 문제가 발생한다. 예를 들어, 100K 바이트의 멀티미디어 데이타 중에서 중간 부분의 10K 바이트의 입력을 요구할 경우, 90K 바이트의 불필요한 디스크 입출력과 버퍼 영역의 낭비가 발생한다.

이와 같은 문제를 해결하기 위해 멀티미디어 데이타의 입력이 요구되면 멀티미디어 데이타 중에서 입력 대상이 되는 부분만을 저장할 수 있도록 물리적으로 연속적인 공간을 버퍼로 할당할 수 있다. 이러한 방식은 불필요한 디스크 입출력을 발생시키지 않고 디스크 영역이 낭비되는 것을 방지하지만 하나의 데이타에 대한 부분 입력 요구가 빈번해지면 버퍼들 사이에 중복되는 부분이 발생하여 버퍼 영역을 낭비하는 문제가 발생한다.

본 절에서는 세그먼트 단위 디스크 입출력과 멀티미디어 데이타의 부분 입력을 효과적으로 지원하는 버퍼 할당 기법을 제안한다. 제안하는 버퍼 관리 기법은 멀티미디어 데이타를 버퍼로 입력할 때, 멀티미

디어 데이타를 구성하는 세그먼트의 크기와 동일하게 버퍼를 할당한다. 그리고 세그먼트에 대해 할당된 버퍼들을 결합하여 멀티미디어 데이타에 대한 입력을 관리한다. 세그먼트의 크기와 동일하게 버퍼를 할당하는 것은 세그먼트의 디스크 입출력 시간이 증가하는 것을 방지한다. 또한 하나의 세그먼트에 대하여 하나의 버퍼를 할당하므로 멀티미디어 데이타의 빈번한 부분 입력 요구시 버퍼 영역의 낭비를 감소시킬 수 있다.

(그림 1)은 제안된 버퍼 관리 기법에 따라 멀티미디어 데이타의 입력을 위해 버퍼가 할당하고 관리되는 것을 보여준다.



(그림 1) 멀티미디어 데이타의 입력을 위한 버퍼 할당
(Fig. 1) Buffer allocation for multimedia data input

(그림 1)에서 멀티미디어 데이타에 대한 입력 요구 A, B, C를 위해 입력되는 데이타를 포함하는 세그먼트들에 대해 각각 버퍼를 할당하고 이것들을 결합하여 멀티미디어 데이타에 대한 입력 요구를 관리한다. (그림 1)은 멀티미디어 데이타에 대한 부분 입력 요구가 빈번해져도 세그먼트에 대해 버퍼가 하나만 할당되어 버퍼 영역의 낭비를 방지할 수 있음을 보여준다.

또한 이진 버디 시스템을 이용하여 버퍼 영역을 관리하고 세그먼트 크기와 동일한 크기의 버퍼를 할당한다. 이진 버디 시스템은 메모리의 빠른 할당과 메모리가 반환될 때 이웃의 자유 메모리 영역과 자동적으로 병합하여 보다 큰 데이타를 빠르게 할당할 수 있는 장점을 지니고 있다. 따라서 이진 버디 시스템을 이용한 버퍼의 할당은 버퍼를 빠르게 할당할 수 있으므로 시스템 응답시간을 줄인다.

〈알고리즘 1〉 버퍼 할당 알고리즘

```
AllocateBuffer(Size, Address)
입력 : 입력되는 세그먼트의 크기
출력 : 할당된 버퍼의 주소
{
  /* 버퍼 영역을 관리하는 이진 버디 시스템을 이
     용하여 탐색 */
  입력되는 세그먼트를 저장할 수 있는 메모리 세
  그먼트를 탐색
  if (적당한 메모리 세그먼트가 존재) {
    if (메모리 세그먼트 크기 == 입력되는 세그먼
       트 크기) {
       메모리 세그먼트를 버퍼로 할당
       할당된 버퍼의 주소를 반환
    }
    if (메모리 세그먼트 크기 > 입력되는 세그먼트
       크기) {
       메모리 세그먼트 중에서 세그먼트를 저장
       할 수 있는 부분만큼을 버퍼로 할당
       메모리 세그먼트의 나머지 부분은 버퍼 영
       역으로 반환
       할당된 메모리 세그먼트의 주소를 반환
    }
  }
  할당할 수 있는 메모리가 없다는 사실을 반환
}
```

〈알고리즘 1〉은 입력이 요구되는 세그먼트를 위해 이진 버디 시스템으로 관리되는 버퍼 영역에서 버퍼를 할당하는 알고리즘이다. 알고리즘을 단순화 시키기 위해 입력이 요구되는 세그먼트의 크기를 변형하여 size 인자를 통해 버퍼 할당 알고리즘에 전달한다. 세그먼트는 2의 급수개의 디스크 페이지로 구성되기 때문에 세그먼트의 크기에 대하여 size는 다음과 같은 식으로 구한다.

$size = \log_2(\text{세그먼트의 페이지 수})$

예를 들어, 32페이지로 구성된 세그먼트의 입력을 위해 버퍼 할당이 요구되면 size는 $\log_2 32 = 5$가 된다.

〈알고리즘 1〉은 입력되는 디스크 세그먼트를 위해 전달된 size 값을 통해 버퍼 영역에서 메모리 세그먼트를 버퍼로 할당한다. 이진 버디 시스템으로 관리되는 버퍼 영역에서 입력되는 디스크 세그먼트의 크기와 같은 메모리 세그먼트가 존재하면 그 메모리 세그먼트를 디스크 세그먼트를 위한 버퍼로 할당한다. 만약 입력되는 디스크 세그먼트와 크기가 같은 메모리 세그먼트가 존재하지 않을 때는 버퍼의 할당을 위해 디스크 세그먼트보다 큰 메모리 세그먼트들 중에서 가장 작은 것을 선택한다. 선택된 메모리 세그먼트 중에서 디스크 세그먼트 크기만큼만 버퍼로 할당하고 나머지는 버퍼 영역으로 반환하여 다음의 버퍼 할당에 대비한다.

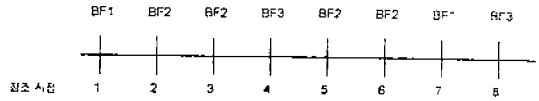
3.2 버퍼 교체 기법

멀티미디어 데이타에 대한 버퍼 교체 정책은 앞으로 참조될 가능성이 가장 적은 데이타에 대한 버퍼를 교체하여 디스크 입출력을 최소로 하는 것을 목표로 삼는다.

기존의 버퍼 관리자는 대부분 LRU를 기반으로 버퍼를 관리한다. 이는 LRU가 구현이 간단하며 비교적 좋은 성능을 보이기 때문이다. 그러나 LRU는 데이타가 마지막으로 참조된 시점만을 이용하여 버퍼를 관리하기 때문에 상대적으로 자주 참조되는 데이타와 자주 참조되지 않는 데이타를 구별하지 못하는 단점을 가지고 있다[3].

LRU-K는 LRU의 성능을 개선하기 위해 버퍼에 대한 참조 간격을 이용하여 버퍼를 교체한다. K는 버퍼에 대한 참조 간격을 구하는데 사용되는 버퍼의 참조 수이다. 따라서 K가 1일 경우에는 LRU-K 기법은 LRU 기법과 동일하다. LRU-K는 데이타에 대한 참조 지역성을 잘 반영하며 다중 사용자 환경에서도 성능이 뛰어나다[3]. 따라서 본 버퍼 관리자는 모든 버퍼들을 참조 간격에 따라 전역 LRU-K 링크로 연결시켜 관리하며, 전역 LRU-K 링크를 통해 참조 간격이 가장 큰 버퍼를 교체한다.

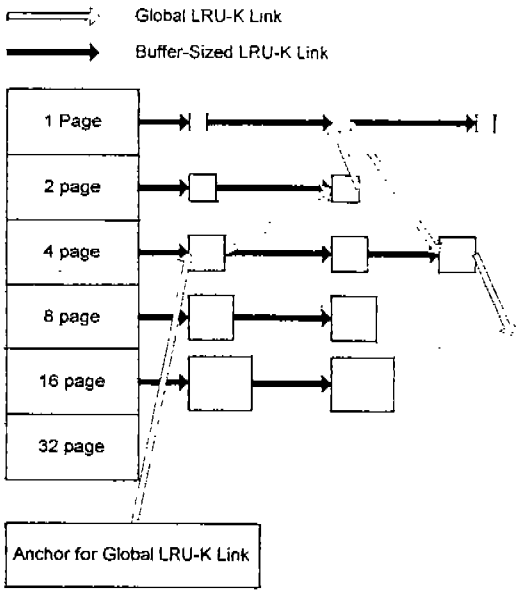
(그림 2)는 버퍼 BF1, BF2, BF3에 저장되어 있는 데이타의 참조를 시간 순으로 보여준다. LRU 기법에서는 마지막 참조 시점이 가장 오래된 BF2가 교체 대상이 된다. BF2는 BF1이나 BF3에 비해 빈번히 참조되는 데이타이며 앞으로 참조될 가능성이 가장 많은 데이타이다. 따라서 BF2를 교체하는 것은 디스크 입출력을 발생시킬 가능성이 높다. K가 2라고 가정하면, LRU-K는 각각의 버퍼에 대한 두 번의 참조 간격을 이용하여 버퍼를 교체한다. BF1의 참조 간격은 6, BF2의 참조 간격은 1, BF3의 참조 간격은 4이다. 따라서 LRU-K는 참조 간격이 가장 큰 BF1을 교체한다.



(그림 2) 버퍼 데이타의 참조 예
(Fig. 2) Example of buffer data reference

그런데 앞 절에서 보았듯이 멀티미디어 데이타를 위한 버퍼의 할당은 세그먼트 단위로 이루어진다. 세그먼트는 그 크기가 가변적이므로 버퍼의 크기 역시 가변적이다. 따라서 입력이 요구되는 세그먼트의 크기와 전역 LRU-K 링크를 통해 교체 대상으로 선택된 버퍼의 크기가 다를 수 있다. 전역 LRU-K 링크를 통해 교체 대상으로 선택된 버퍼의 크기가 입력이 요구되는 세그먼트의 크기보다 작으면 그 버퍼는 교체될 수가 없다. 따라서 전역 LRU-K 링크를 통해 두번째로 참조 간격이 긴 버퍼가 교체 대상이 될 수 있다. 그러나 이 버퍼 역시 입력이 요구되는 세그먼트의 크기보다 작으면 교체 대상이 될 수 없다. 따라서 전역 링크 탐색의 횟수를 제약하지 않으면 최악의 경우 전역 링크를 모두 탐색하여야 되는 경우가 발생할 수 있다. 즉 탐색에 소요되는 시간이 커지며 소요되는 시간의 변동(variation)가 커지게 된다. 이러한 시간의 변동이 커지면 시스템 응답의 예측이 더욱 어려워지는 문제를 발생시킨다. 따라서 본 멀티미디어 데이타 버퍼 관리자는 버퍼를 교체할 때, 전역 LRU-K 링크가 계속 탐색되어 많은 탐색 시간이 소비되는 것을 방지하기 위해 버퍼를 크기에 따라 버퍼 크기별 LRU-K 링크로 연결 관리한다. 일정한 횟수까지는 전역 LRU-K 링크를 탐색하며 그 때까지 적당한 크기의 버퍼를 찾지 못하면 버퍼 크기별 LRU-K 링크를 탐색하여 교체할 버퍼를 탐색한다. 버퍼 크기별 LRU-K 링크는 버퍼의 크기가 페이지 크기의 2의 급수로 증가하기 때문에 테이블 구조를 이용하여 간단하게 구성된다.

(그림 3)은 버퍼들이 전역 LRU-K 링크와 버퍼 크기별 LRU-K 링크로 관리되는 것을 보여준다.



(그림 3) 전역 LRU-K 링크와 버퍼 크기별 LRU-K 링크로 관리되는 버퍼

(Fig. 3) Buffers managed by global LRU-K Link and Buffer-sized LRU-K Link

〈알고리즘 2〉 버퍼 교체 알고리즘

```
ReplaceBuffer(Size, BufferAddr)
입력 : 입력되는 세그먼트의 크기
출력 : 할당된 버퍼의 주소
{
  /*전역 LRU-K 링크 탐색. */
  일정 횟수까지 전역 LRU-K 링크를 탐색
  if (교체될 버퍼가 존재) {
    if (버퍼의 내용이 변경)
      버퍼의 내용을 디스크에 반영
  버퍼의 주소를 반환
  }
  /* 버퍼 크기별 LRU-K 링크 탐색. */
  테이블로 관리되는 버퍼 크기별 LRU-K 링크를 탐색
  if (적당한 버퍼가 존재) {
    if (버퍼의 내용이 변경)
      버퍼의 내용을 디스크에 반영
    if (버퍼의 크기 > 입력되는 세그먼트의 크기)
      버퍼에서 입력되는 세그먼트의 크기를 뺀
      부분을 버퍼 영역으로 반환
    버퍼의 주소를 반환
  }
  /* 버퍼를 병합하여 할당. */
  전역 LRU-K 링크에서 교체 대상이 되는 버퍼들을
  병합
  갱신된 버퍼들이 있으면 디스크에 반영
  병합된 버퍼들을 세그먼트의 입력을 위해 버퍼로
  할당
  버퍼의 주소를 반환
}
```

〈알고리즘 2〉는 세그먼트 단위의 데이타 입력을 위해 버퍼를 교체하는 버퍼 교체 알고리즘이다. 교체 알고리즘은 다음과 같이 수행된다.

버퍼의 교체를 위해 전역적인 LRU-K 링크를 이용하여 교체될 버퍼를 선택한다. 전역 LRU-K 링크는 버퍼의 참조 간격이 가장 큰 것으로부터 가장 작은 것의 순으로 이중 연결(double linked) 구조로 연결되어 있으며, 가장 큰 참조 간격을 가지는 것이 우선적으로 교체 대상으로 선택된다. 만약 선택된 버퍼의 크기가 입력되는 세그먼트 크기보다 작으면 전역 LRU-K 링크를 탐색하면서 입력되는 세그먼트 크기보다 크거나 같은 버퍼가 결정될 때까지 반복한다. 탐색된 버퍼의 크기가 요구되는 세그먼트 크기보다 크면, 탐색된 버퍼 중에서 세그먼트 크기를 제외한 나머지 부분은 버퍼 영역으로 반환된다. 이렇게 버퍼 영역으로 반환된 부분은 다음에 그것이 저장할 수 있는 세그먼트의 입력이 요구될 경우 버퍼로 할당된다.

교체를 위한 적당한 버퍼를 찾기 위한 탐색 횟수가 전역 LRU-K 링크에서의 제한된 탐색 횟수보다 커지면, 버퍼 크기별로 유지되는 버퍼 크기별 LRU-K 링크를 탐색한다. 입력되는 세그먼트 크기와 동일한 크기의 버퍼가 존재하면 그 버퍼를 교체한다. 만약에 입력되는 세그먼트 크기에 해당하는 버퍼가 존재하지 않으면, 다음 크기의 버퍼들에 대한 버퍼 크기별 LRU-K 링크를 탐색한다. 그리고 탐색된 버퍼 중에서 세그먼트 크기만큼만 버퍼로 할당하고 나머지 부분은 버퍼 영역으로 반환하여 다음 세그먼트의 입력

에 대비한다.

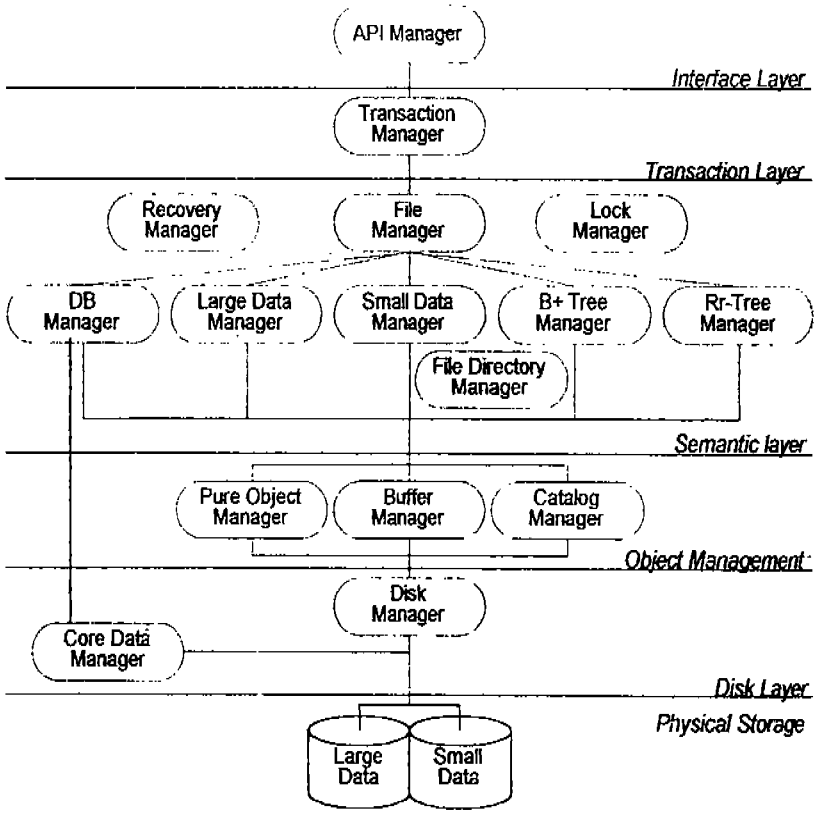
또한 버퍼 관리가 허용되는 크기의 세그먼트이지만 교체할 버퍼가 존재하지 않을 경우에는, 전역 LRU-K 링크에 의해 첫번째 교체 대상이 되는 버퍼와 버퍼 영역에서 그 주위에 있는 버퍼들을 병합하여 입력이 요구되는 세그먼트를 저장할 수 있도록 하나의 버퍼로 만들어 세그먼트를 입력한다. 버퍼의 병합은 이진 버디 시스템의 병합알고리즘을 이용하여 수행한다[9].

4. 멀티미디어 데이타 버퍼 관리자의 설계 및 구현

4.1 멀티미디어 데이타 저장 관리자 KORED/STORM의 구조

(그림 4)는 멀티미디어 데이타를 처리하는 저장 관리자인 KORED/STORM의 구조이다. (그림 4)에서 디스크 관리자는 디스크에 대한 모든 입출력과 저장 공간의 할당 및 반환을 담당한다. 스몰 데이타 관리자는 페이지 내에서 슬롯(slot) 구조를 이용하여 크기가 작은 정형 데이타를 관리한다. 멀티미디어 데이타 관리자는 대용량 멀티미디어 데이타를 관리한다. 인덱스 관리자는 데이타에 대한 B^+-tree와 공간 데이타에 대한 Rr-tree를 지원한다. 록 관리자(lock manager)는 트랜잭션들 사이의 동시성 제어를 수행하며 회복 관리자(recovery manager)는 고장 회복 기능을 갖는다. API 관리자(Application Program Interface Manager)는 상위 응용 프로그램을 위한 인터페이스를 제공한다[19].

KORED/STORM은 스몰 데이타를 위한 버퍼 관리자와 멀티미디어 데이타를 위한 버퍼 관리자를 분리하



(그림 4) KORED/STORM의 구조

(Fig. 4) The architecture of KORED/STORM

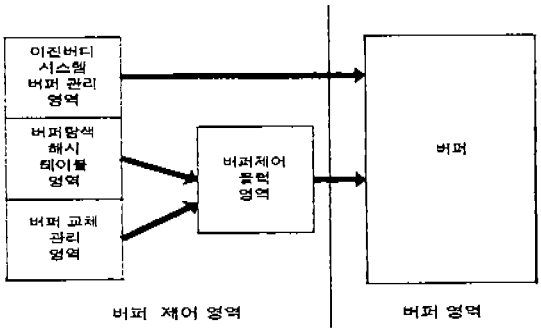
여 스몰 데이타와 멀티미디어 데이타를 별도로 버퍼 관리한다. 멀티미디어 데이타 버퍼 관리자는 하부의 디스크 관리자를 통해서 멀티미디어 데이타의 버퍼 관리를 수행한다.

4.2 멀티미디어 데이타 버퍼 관리자의 구현

4.2.1 버퍼 풀의 영역 구성

본 멀티미디어 데이타 버퍼 관리자가 관리하는 메모리 영역은 버퍼 영역과 버퍼 제어 영역(buffer control area)으로 구성된다. 버퍼 영역은 세그먼트의 입출력을 위해 버퍼를 할당하는 영역이다. 버퍼 제어 영역은 버퍼 제어 블럭을 위한 영역, 버퍼 탐색 해시 테이블(hash table) 영역, 이진 버디 시스템 관리 영역과 버퍼 교체를 위한 영역으로 구성된다. 버퍼 제어 블럭 영역은 버퍼에 입력된 세그먼트에 대한 세그먼트 식별자, 버퍼를 공유하는 트랜잭션의 수, 버퍼내 데이타의 갱신 여부, 버퍼의 참조 간격 등과 같은 버퍼에 대한 제어 정보로 구성된 버퍼 제어 블럭을 관리하기 위한 영역이다. 버퍼 탐색 해시 테이블 영역은 버퍼에 대한 빠른 탐색을 위해 관리되는 영역이다. 이진 버디 시스템 관리 영역은 버퍼 영역을 이진 버디 시스템으로 관리하기 위해 필요한 영역이다. 버퍼 교체를 위한 영역은 버퍼의 크기별 LRU-K 링크와 전역 LRU-K 링크를 관리하기 위해 필요한 영역이다. 이와 같은 작업을 위해 본 버퍼 관리자의 메모리 영역은 시스템의 공유 메모리(shared memory) 영역으로부터 할당받는다.

(그림 5)는 본 버퍼 관리자의 메모리 구성을 보여준다.



(그림 5) 버퍼 관리자에서 관리하는 메모리 영역의 구성
(Fig. 5) Memory area managed by buffer manager

4.2.2 버퍼 탐색

세그먼트가 디스크에서 입력될 때, 세그먼트가 이미 버퍼로 관리되어 있다면 세그먼트를 디스크에서 읽어올 필요가 없다. 따라서 세그먼트의 입력을 수행할 때 세그먼트의 내용을 이미 저장하고 있는 버퍼에 대한 탐색 작업은 버퍼 할당과 교체 작업에 선행되는 작업이다. 본 버퍼 관리자는 해시 테이블을 통해 버퍼 제어 블럭들을 관리함으로써 버퍼의 빠른 탐색 기능을 제공한다.

〈알고리즘 3〉은 해시 함수를 이용하여 버퍼를 탐색하는 기능을 다음과 같이 수행한다. 먼저 세그먼트 식별자와 세그먼트가 속해 있는 데이타베이스 인덱스를 해시 함수에 전달하여 해시 테이블 인덱스를 구한다. 해시 테이블의 각 인덱스 별로 관리되는 버퍼 제어 블럭 링크를 따라 버퍼 제어 블럭을 차례로 탐색하면서 입력하려는 세그먼트를 저장하는 버퍼가 존재하는가를 결정한다. 세그먼트를 저장하고 있는 버퍼가 존재하면 그 버퍼를 가리키는 버퍼 제어 블럭을 반환한다. 세그먼트를 저장하고 있는 버퍼가 존재하지 않으면 버퍼가 존재하지 않다는 것을 버퍼 탐색 함수의 호출자에게 알린다.

버퍼 탐색 알고리즘의 구현을 단순하게 하기 위해 해시 함수는 나머지 연산(modular operation)을 이용하여 구현한다. 해시 함수에 전달되는 인자는 세그먼트 식별자와 데이타베이스 인덱스이다.

〈알고리즘 3〉 버퍼 탐색 알고리즘

```
SearchBuffer(Dbidx, SegmentID, BCB)
입력 : 데이타베이스 인덱스
       세그먼트 식별자
출력 : 버퍼에 대한 버퍼 제어 블럭
{
  데이타베이스 인덱스와 세그먼트 식별자를 해시
  함수에 전달하여 해시 테이블의 인덱스를 구함
  해시 테이블의 인덱스가 가리키는 버퍼 제어 블
  럭을 선택
  While (무한 루프) {
    if (버퍼 제어 블럭이 가리키는 버퍼가 세그먼
        트를 저장하고 있음)
      버퍼 제어 블럭을 반환
    버퍼 제어 블럭 링크 상에서 현재 버퍼 제어
```

```
        블럭의 다음 버퍼 제어 블럭을 선택
        if (버퍼 제어 블럭이 존재하지 않음)
            While 루프를 빠져 나감
    }
    세그먼트를 저장하는 버퍼가 존재하지 않는다는
    사실을 반환
}
```

4.2.3 멀티미디어 데이타의 읽기 쓰기

응용 프로그램에서 사용하고 있는 버퍼가 다른 세그먼트의 입력 때문에 교체되어서는 안된다. 따라서 현재 사용되고 있는 버퍼는 교체 대상에서 제외되어야 한다. 이와 같은 기능을 수행하기 위해 본 버퍼 관리자는 버퍼 고정(fix) 함수와 버퍼 해제(unfix) 함수를 지원한다. 버퍼 고정 함수는 세그먼트를 버퍼로 입력하고, 응용 프로그램에서 버퍼의 사용이 종료될 때까지 버퍼가 교체되는 것을 방지하는 기능을 수행한다. 버퍼 해제 함수는 응용 프로그램에서 버퍼의 사용이 종료된 후에, 버퍼가 교체 대상이 될 수 있도록 하는 기능을 수행한다.

〈알고리즘 4〉는 다음과 같은 순서로 버퍼를 고정시키는 기능을 수행한다. 우선 입력되는 세그먼트가 이미 버퍼로 관리되어 있으면 단순히 버퍼 제어 블럭의 내용만 갱신한다. 만약 세그먼트에 대한 버퍼가 존재하지 않으면 버퍼 영역에서 버퍼를 할당받아 세그먼트를 입력한다. 버퍼의 할당이 실패하면 전역 LRU-K 링크와 버퍼 크기별 LRU-K 링크를 이용하여 버퍼를 교체한다. 디스크에서 버퍼로 세그먼트를 읽어오기 전에 버퍼 제어 블럭의 내용을 조정하여 버퍼가 교체 대상이 되는 것을 방지한다.

버퍼 해제 함수는 사용이 끝난 세그먼트를 저장하고 있는 버퍼가 교체 대상이 될 수 있도록 버퍼 제어 블럭의 내용을 갱신하고 전역 LRU-K 링크와 버퍼 크기별 LRU-K 링크를 조정한다.

버퍼로 입력된 세그먼트가 갱신되었다면 이것은 디스크에 반영되어야 한다. 갱신된 세그먼트가 디스크에 반영되는 시점은 세그먼트를 저장하는 버퍼에 대한 버퍼 해제 함수를 호출할 때가 아니라 다른 세그먼트의 입력을 위해 버퍼가 교체될 때 수행된다.

〈알고리즘 4〉 버퍼 고정 알고리즘

```
FixBuffer(Dbidx, SegmentID, BufferAddress)
입력 : 데이타베이스 인덱스
       세그먼트 식별자
출력 : 입력되는 세그먼트를 저장하는 버퍼의 주소
{
  버퍼 탐색 함수를 이용하여 세그먼트를 저장하는
  버퍼 탐색
  if (버퍼가 존재) {
     버퍼를 관리하는 버퍼 제어 블럭의 내용을 갱신
     버퍼를 전역, 버퍼 크기별 LRU-K 링크에서 삭제
     버퍼의 주소를 반환
  }
  버퍼 할당 알고리즘을 이용하여 세그먼트를 위한
  버퍼를 할당
  if (버퍼 할당이 성공) {
     버퍼를 위한 버퍼 제어 블럭을 만들고 그 내
     용을 갱신
     세그먼트를 버퍼로 입력
     버퍼의 주소를 반환
  }
  버퍼 교체 알고리즘을 이용하여 교체될 버퍼를 선택
  버퍼를 위한 버퍼 제어 블럭을 만들고 그 내용을 갱신
  세그먼트를 버퍼로 입력
  버퍼의 주소를 반환
}
```

5. 결 론

본 논문은 멀티미디어 데이타의 디스크 입출력을 줄이므로써 시스템의 성능을 향상시킬 수 있는 멀티미디어 데이타 버퍼 관리 기법을 제안하였다.

제안된 버퍼 관리 기법은 멀티미디어 데이타의 세그먼트 단위 디스크 입출력을 효율적으로 처리하기 위하여 세그먼트 크기와 동일한 크기로 버퍼를 할당하여 디스크 입출력에 소요되는 시간이 감소시켰다. 또한 이진 버디 시스템을 이용하여 버퍼 영역을 관리하여 신속하게 버퍼를 할당하며 반환하였다. 멀티미디어 데이타의 부분 입력에 대하여 멀티미디어 데이타 중에서 입력되는 데이타 부분을 포함하는 세그먼트들만을 버퍼로 관리하여 멀티미디어 데이타 중에서 입력이 요구되는 부분 외에 나머지 부분에 대한

디스크 입출력을 방지하여 버퍼 영역의 낭비를 막고 멀티미디어 데이타에 대한 빈번한 부분 입력 발생시 버퍼 영역이 낭비되는 문제점을 해결하였다. 제안된 버퍼 관리 기법은 버퍼 크기의 가변 특성을 고려하여 버퍼를 교체할 때, 버퍼의 참조 행동뿐만 아니라 버퍼의 크기를 고려하였으며 버퍼의 참조 간격을 이용하는 LRU-K를 사용하여 버퍼 교체의 성능을 향상시켰다.

향후, 디스크 상에서 세그먼트 단위로 관리되지 않는 멀티미디어 데이타의 버퍼 관리에 대한 연구가 필요하다.

참 고 문 헌

[1] A. Biliris, "The Performance of Three Database Storage Structures for Managing Large Objects," Proc. of 1992 ACM SIGMOD Int. Conf. on Management of Data, pp.276-285, 1992.

[2] A. Biliris, "The EOS Large Object Manager," AT &T Technical Report, ATT-DB-92-2, 1992.

[3] E. J. O'Neil, et al., "The LRU-K Page Replacement Algorithm For Database Disk Buffering," Proc. of 1993 ACM SIGMOD Int Conf. on Management of Data, pp.297-306, 1993.

[4] EXODUS Project Group, "Using the EXODUS Storage System V3.1," EXODUS Project Document, University of Wisconsin-Madison, 1993.

[5] G. M. Sacco, et al., "Buffer Management in Relational Database Systems," ACM TODS, Vol. 24, No. 7, pp.412-418, 1982.

[6] H. T. Chou, et al., "An Evaluation of Buffer Management Strategies for Relational Database System," Proc. of the 11th VLDB, pp.127-141, 1985.

[7] H. T. Chou, et al., "Design and Implementation of the Wisconsin Storage System," Software Practice and Experience, Vol. 15, No. 10, pp.943-962, 1985.

[8] J. Gray, et al., "The Five Minute Rule for Trading Memory for Disk Accesses and The 10 Byte Rule for Trading Memory for CPU Time," Proc. of 1987 ACM SIGMOD Int. Conf. on Management of Data, pp.395-398, 1987.

[9] J. J. Peterson, et al., "Buddy Systems," Communications of the ACM, Vol. 20, No. 6, pp. 421-431, 1977.

[10] J. Waterworth, Multimedia, Ellis Horwood, pp. 114-150, 1991.

[11] J. Z. Teng, et al., "Managing IBM Database 2 Buffers to Maximize Performance," IBM Systems Journal, Vol. 23, No. 2. pp.211-218, 1984.

[12] M. J. Carey, et al., "Object and File Management in the EXDOUS Extensible Database System," Proc. of the 12th VLDB, pp.375-383, 1986.

[13] M. Stonebraker, "Operating System Support for Database Management," Communications of the ACM, pp.412-418, 1981.

[14] P. G. Selinger, et al., "Predictions and Challenges for Databae Systems in the Year 2000," Proc. of 19th VLDB, pp.667-675, 1993.

[15] T. J. Lehman, B. G. Lindsay, "The Starburst Long Field Manager," Proc. of 15th VLDB, pp. 375-383, 1989.

[16] W. Effelsberg, et al., "Principles of Database Buffer Management," ACM TODS, Vol. 9. No. 4, pp.560-595, 1984.

[17] 김종훈, 김재홍, 배해영, "KORED/GEO의 공간 데이타 처리기의 설계 및 구현," 한국정보과학회, '92 가을 학술발표논문집, Vol. 19, No. 2, pp. 55-58, 1992.

[18] 김홍연, 김재홍, 배해영, "유닉스 환경 하에서 효율적인 저장 관리자의 설계," 한국정보과학회, '93 봄 학술 발표논문집, Vol. 20, No. 1, pp.47-50, 1993.

[19] 방기식, 조영섭, 김재홍, 배해영, "멀티미디어 응용을 위한 멀티미디어 데이타 관리 기법," 한국정보처리응용학회 '94 가을 학술 발표논문집, Vol. 1, No. 2, pp.303-306, 1994.

[20] 조영섭, 김재홍, 배해영, "통 데이타의 효율적 처리를 위한 KORED/STORM의 버퍼 관리기", 한국정보과학회, '94 봄 학술 발표논문집, Vol. 21, No. 1, pp.77-80, 1994.

조 영 섭

1993년 인하대학교 공과대학 전자계산공학과 졸업
1995년 인하대학교 대학원 전자계산공학과 졸업
1995년~현재 인하대학교 대학원 전자계산공학과 박사과정

김 재 홍

1988년 인하대학교 전자계산학과(이학사)
1990년 인하대학교 대학원 전자계산학과(이학석사)
1994년 인하대학교 대학원 수학과 전자계산학(이학박사)
1994년 인하대학교 전자계산공학과 전임대우
1995년~현재 영동공과대학 컴퓨터공학과 전임강사

배 해 영

1974년 인하대학교 응용물리학과(공학사)
1978년 연세대학교 대학원 전자계산학과(공학석사)
1989년 숭실대학교 대학원 전자계산학과(공학박사)
1985년 Univ. of Houston 객원교수
1992년~1994년 인하대학교 전자계산소 소장
1982년~현재 인하대학교 전자계산공학과 교수