DOI: 10.3745/KIPSTD.2009.16-D.2.153

공간 시간 근접성을 이용한 효율적인 버퍼 관리 기법

민 준 기†

요 약

효율적인 버퍼 관리는 시스템의 성능과 밀접한 관련이 있다. 따라서 다양한 버퍼 관리 기법들에 대한 많은 연구가 진행되어 왔다. 그러나 많은 제안된 기법들의 대부분은 시간 근접성만을 고려하고 있다. 공간 데이터베이스와 같은 환경에서는 시간 근접성뿐 만 아니라, 유사한위치에 있는 공간 객체들은 서로 같이 접근 될 가능성이 높다는 공간 근접성도 존재한다. 따라서, 본 논문에서는 공간 데이터베이스 환경에서 시간 근접성과 공간 근접성을 같이 효과적으로 고려하는 버퍼 관리 기법인BEAT를 제안한다. 실제 데이터와 가상 데이터를 이용한 실험 결과는 BEAT의 효율성을 보인다.

키워드 : 공간 데이터, 버퍼 교체, 근접성

An Efficient Buffer Management Technique Using Spatial and Temporal Locality

Jun-Ki Min†

ABSTRACT

Efficient buffer management is closely related to system performance. Thus, much research has been performed on various buffer management techniques. However, many of the proposed techniques utilize the temporal locality of access patterns. In spatial database environments, there exists not only the temporal locality but also spatial locality, where the objects in the recently accessed regions will be accessed again in the near future. Thus, in this paper, we present a buffer management technique, called BEAT, which utilizes both the temporal locality and spatial locality in spatial database environments. The experimental results with real-life and synthetic data demonstrate the efficiency of BEAT.

Keywords : Spatial Data, Buffer Replace, Locality

1. 서 론

전통적인 데이터베이스 관리 시스템(Database Management System: DBMS)은 문자열, 숫자와 같은 단순한 구조의 데이터만을 처리한다. 건물, 도로와 같은 공간 객체를 처리하기 위하여 DBMS의 기능을 확장한, 공간 데이터베이스 관리 시스템(Spatial DBMS: SDBMS)이 제안되었다[1].

공간 관리 시스템(Geographic information System: GIS)과 CAD, 멀티미디어 시스템, 인공위성 이미지 시스템, 그리고 위치 기반 서비스(Location Based Service: LBS) 등의 응용 분야의 폭넓은 사용에 따라 공간 데이터베이스 관리는 매우 활동적인 연구 분야가 되었다. 많은 SDBMS에 대한 연구는 시스템의 성능 향상을 위한 공간 데이터 인덱스[2], 질의 처리 방법[3, 4] 등에 집중되었다.

메인 메모리의 크기는 디스크에 비하여 상당히 작기 때문에 디스크 I/O의 발생을 피할 수 없다. 이러한 디스크 I/O의 횟수를 줄이기 위하여 버퍼가 사용된다. 특히, 공간 데이터는 공간 객체의 큰 크기 때문에 디스크 공간을 많이 사용함으로 효율적인 버퍼 관리가 매우 중요하다. 버퍼의 효율적인 관리는 데이터베이스의 성능과 밀접하게 관련되어 있기 때문에 많은 연구자들이 다양한 버퍼 관리 기법들을 제안하였다[5-7, 15, 17].

특히, 지금까지 잘 알려진 버퍼 관리 기법들은 현재 자주 접근되는 데이터 아이템은 앞으로도 자주 사용될 가능성이 높다라고 하는 시간 근접성(temporal locality) 을 고려하고 있다. 그러나 공간 데이터베이스에서는 위치적으로 근접한 데이터 아이템이 앞으로도 자주 접근될 가능성이 높다는 공간 근접성 (spatial locality) 이 존재한다. 따라서, SDBMS에서는 효율적인 버퍼 관리를 위한 공간 근접성도 고려하여야

※ 이 논문은 2008년도 정부(교육과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업으로 수행된 연구임(No.R0A-2008-000-10225-0).
† 종신회원 : 한국기술교육대학교 인터넷미디어공학부 조교수
논문접수 : 2008년 1월 14일
수 정 일 : 1차 2008년 10월 21일
심사완료 : 2008년 11월 11일

한다. 그러나 전통적인 버퍼 관리 기법들은 시간 근접성만을 고려하고 있다[5-7].

본 논문에서는 기존의 버퍼 관리에서 유용한 척도로 사용되었던 시간 근접성과 공간 데이터베이스에서 발생되는 공간 근접성을 통합적으로 활용하여 공간 데이터베이스의 버퍼 관리를 효율적으로 처리할 수 있는 BEAT (Buffer rEplace mAnagement Technique)이라는 새로운 버퍼 관리 기법을 제안한다. 본 논문에서 제안한 기법은 많이 사용되는 LRU 큐(Queue)를 활용한다. 또한 우리는 BEAT를 구현하고 다양한 접근 유형 및 버퍼 크기에 따른 실험을 수행하였다. 실험의 결과로써 본 논문에서 제안한 BEAT 기법이 공간 데이터베이스 환경에서 기존의 버퍼 관리 기법들에 비하여 보다 효율적임을 확인하였다.

본 논문의 구성은 다음과 같다. 2절에서 시간 근접성만을 고려하는 전통적인 버퍼 관리 기법들과 공간 데이터베이스 시스템을 위하여 고안된 버퍼관리 기법들에 대하여 살펴본다. 3절에서는 본 논문에서 제안하는 BEAT 기법의 세부 사항에 대하여 다룬다. 4절에서는 실험을 통한 BEAT 기법과 기존의 다른 버퍼 관리 기법들과의 비교에 대하여 다루고 마지막으로 5절에서 본 연구의 결론을 맺는다.

2. 관련 연구

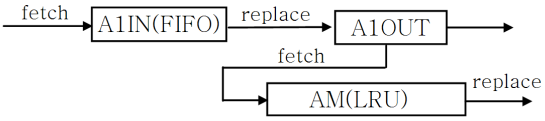
버퍼 관리 기법의 핵심은 버퍼 교체 알고리즘이다. 버퍼가 가득 차 있는 상황에서, 새로운 데이터 아이템을 버퍼에 적재하고자 할 때, 버퍼 관리자는 버퍼 교체 알고리즘을 이용하여 접근 패턴을 분석하고 교체할 희생자(victim)을 선정한다. 버퍼 관리 관련 연구는 오래되고 풍부한 역사를 가지고 있다. 이 절에서는 다양한 버퍼 교체 알고리즘들 중 대표적인 기법들에 대하여 설명하도록 하겠다.

2.1 전통적인 버퍼 교체 알고리즘

지금까지 제안된 버퍼 관리 기법 등 중에서 가장 많이 알려진 기법이 LRU (Least Recently Used) [5] 이다. LRU 기법은 버퍼에 존재하는 데이터 아이템 중 가장 오랫동안 사용되지 않은 데이터 아이템 (즉 LRU 데이터 아이템)을 희생자로 삼는다. LRU 기법은 최근에 사용된 데이터 아이템은 앞으로 다시 사용된다 라고 하는 단순한 경험론적 규칙(heuristic rule)을 기반으로 하고 있어서 다양한 형태의 데이터 접근 패턴을 효과적으로 지원하지 못한다는 단점이 존재한다.

이러한 문제점을 해결하기 위하여, LRU-K 기법[6]이 제안되었다. LRU-K 기법은 각 데이터아이템에 대한 최근 k번의 참조 시간들을 유지하고, 가장 오래 전에 k번째 참조가 발생된 데이터 아이템을 희생자로 삼는다. LRU-K 기법은 각 데이터 아이템마다 k개의 접근 이력을 유지하고 희생자를 찾는데 많은 비용이 발생한다는 단점이 존재한다.

따라서, John과 Shasha는 LRU-2처럼 동작하지만 보다 효율적인 2Q 기법[7]을 제안하였다. 2Q 기법의 구조는 (그림 1)과 같다.



(그림 1) 2Q 기법의 구조

2Q는 버퍼를 2개의 독립된 큐- A1IN 큐와 AM 큐-로 관리한다. A1IN 큐는 선입선출 방식(FIRST-IN-FIRST-OUT: FIFO)으로 동작하며 AM큐는 LRU 큐와 같이 동작한다. 가까운 과거에 사용되지 않았던 새로운 데이터 아이템이 버퍼에 적재되면, 그 데이터 아이템 정보가 A1IN 큐에 삽입된다. 그렇지 않다면 새로운 데이터 아이템은 AM 큐에 삽입된다. 2Q에서 데이터 아이템의 교체 이력은 A1OUT 큐에서 관리 된다. A1OUT 큐는 데이터 아이템 자체는 포함하지 않고 해당 데이터 아이템에 대한 정보만을 유지하고 있다. 따라서, A1OUT 큐의 정보를 이용하여 데이터 아이템이 가까운 과거에 사용되었었는지 아닌지를 결정할 수 있다. 2Q 기법은 (그림 1)에서 보이는 각 큐의 크기 값에 따라서 그 성능 좌우된다는 단점이 존재한다.

최근에 각광 받고 있는 데이터 마이닝 기법을 활용한 BROOM [8] 이라는 기법도 제안되었다. 이 기법은 데이터 아이템 접근에 있어서 임의의 데이터 아이템집합이 접근되면 어떤 데이터 아이템도 같이 접근된다 라는 연관 규칙(association rule)을 이용한 것으로 이러한연관 규칙을 추출하기 위하여 offline 상에서의 학습 단계가 필요하다는 단점이 존재하며, 대화형 시스템(interactive system)과 같이 데이터 접근 패턴이 동적으로 변화하는 환경에서는 적용이 어렵다는 단점이 존재한다.

접근 빈도와 접근 이력은 시간 근접성의 주요한 지시자(indicator) 이다. LRFU[9]는 이 두 가지 지시자를 통합한 기법이다. LRFU에서, 버퍼 상에 존재하는 각 데이터 아이템 x는 다음과 같은 C(x) 값을 지닌다. 우선 각 데이터 아이템 x에 대하여 처음 버퍼에 적재될 때 C(x)값에 0 을 할당한다. 그리고 각 시간 t에 대하여 C(x)를 다음과 같이 갱신한다.

$$C(x) = \begin{cases} 1+2^{-\lambda}C(x), & \text{만약 } x\text{가 시간 } t\text{에 참조되면} \\ 2^{-\lambda}C(x), & \text{그렇지 않다면} \end{cases}$$

여기서 λ 는 조정 변수(tunable parameter)임.

버퍼 교체가 필요할 때, C(x) 값이 가장 작은 데이터 아이템을 찾아서 희생자로 삼는다. 위의 수식에서 값이 1에 접근할수록 C(x)값은 최근 데이터 아이템 접근에 더 많은 가중치를 두게 되어 LRFU의 동작이 LRU와 비슷하게 된다. 또한 값이 0에 접근하면 C(x)값은 단순히 각 데이터 아이템에 대한 접근 횟수만을 나타냄으로 LRFU는 접근 빈도에 기반한 LFU (least frequency used)와 유사하게 동작한다. 따라서, LRFU의 성능은 에 의하여 결정된다.

LRFU의 값, 2Q의 버퍼 크기 비율 등 과 같은 조정 변수

의 최적 값을 효율적으로 찾아내는 것은 현실적으로 불가능하다. 따라서, 이러한 조정 변수 없이 데이터 접근 패턴에 따라서 버퍼의 동작이 바뀌는 ARC [10] 기법이 제안되었다. ARC 기법은 c 크기의 버퍼를 p 크기의 LRU 버퍼 B1과 c-p 크기의 LRU 버퍼 B2 로 나누어 관리한다. 여기서 버퍼 B1과 B2에 존재하지 않는 새로운 데이터 아이템이 접근되면 B1에 적재하고, B1 또는 B2에 존재하는 데이터 아이템에 대한 접근이 발생하면 그 데이터 아이템을 B2에 적재한다. 여기서 버퍼 B1, B2의 크기는 변수 p에 의하여 결정되고 p는 지속적으로 변경된다. 즉, 버퍼 B1은 최근에 한번 사용된 데이터 아이템들을 관리하며, B2는 적어도 2번 이상 접근된 데이터 아이템들을 관리한다. 버퍼가 가득 차서 버퍼 교체가 필요할 경우에는 버퍼 B1의 크기가 변수 p 보다 클 경우 B1에서 LRU 정책에 따라서 희생자가 결정되며, 그렇지 않을 경우 버퍼 B2에서 LRU 정책에 따라서 희생자가 결정된다.

변수 p 값은 B1 버퍼에 존재하는 데이터 아이템에 대한 접근이 발생되면 증가되고 B2 버퍼에 대한 접근이 발생되면 감소한다. 여기서, ARC 기법은 학습율(learning rate) 에 따라서 p 값의 증감량을 조정할 할 수 있다. 즉 ARC의 기본 목표인 조정 변수의 제거를 이룩하지 못하고 있다.

최근에, 총 접근 회수만을 고려하는 LFU의 일반화된 형태로 단위 시간 단의 접근 횟수(즉, 속도), 또는 단위 시간당의 접근 횟수의 증가량 (즉, 가속도)등을 이용한 LFU-K [11] 기법 과 가장 최근의 참조 시간과 가장 최근의 참조 시간과 그 이전 참조 시간 사이의 총 데이터 아이템 참조 횟수를 이용하여 해당 데이터 아이템의 참조 시간을 예측하고 그 정보를 활용하여 교체 대상을 선정하는 TNRP [12] 기법이 제안되었다.

또한, 일반적인 데이터 아이템이 아닌 인덱스 아이템에 대한 버퍼 관리 기법으로 ILRU [13], GHOST [14] 등이 있다.

2.2 공간 데이터를 위한 버퍼 교체 알고리즘

지금까지 알려진 버퍼 관리 기법들은 시간 근접성만을 고려한다. 그러나, 공간 데이터베이스의 특성을 파악하고 이를 이용한 공간 데이터베이스에 보다 적합한 버퍼 관리 기법들이 제안되었다.

Papadopoulos와 Manolopoulous는 LRD-Manhattan [15] 기법을 제안하였다. 공간 데이터베이스의 공간 객체는 계산 복잡도를 줄이기 위하여 MBR(minimum bounded rectangle)로 단순화 되어 표현된다. 균등 분포 가정하에서는, 단위 공간에서 임의의 점을 찍을 때 큰 공간 객체가 해당 점을 포함할 확률이 작은 객체가 해당 점을 포함할 확률 보다 크다 [16]. 즉, 확률적으로, MBR의 크기가 클수록 해당 데이터를 접근할 확률이 작은객체를 접근할 확률 보다 높다는 것이다. 따라서, 이 기법은 각 공간 객체의 접근 밀도 (즉 전체 공간 데이터 접근 중 해당 공간 객체의 접근 비율)와 MBR의 정규화 크기의 산술 평균을 이용하여 최소 값을 가지는 공간 객체를 교체 대상으로 삼는다.

또한, 최근에 LRU 규칙과 공간 데이터의 크기를 통합하여 고려하는 ASB [17] 기법이 제안되었다. ASB기법은 버퍼를 ARC 기법처럼 논리적으로 구분된 2개의 버퍼로 구성한다. 즉, 버퍼의 크기가 M일 때, P 크기의 B1과 M-P 크기의 B2로 구성한다. 이 중 B1 은 LRU 특성을 이용하여 관리되며 B2 에서는 MBR 크기와 같은 공간 데이터 특성을 이용하여 관리된다. 새로운 공간 객체는 B1 부분에 적재되며 이로 인하여 B1 공간 부분이 모자라면 LRU 객체를 B2으로 넘긴다. B2 에는 P 크기의 버퍼(즉, B1)에서 관리되지 못한 LRU 객체들이 유지되어 있다. ASB에서 객체 교체가 필요한 상황에서는 버퍼의 B2 에서 (즉, LRU객체들 중에서) 가장 작은 MBR의 공간 객체를 찾아서 희생자로 삼는다. 또한, ASB 접근되는 공간 객체의 특성을 파악하여 B1부분의 크기 및 B2 부분의 크기를 가변적으로 조정한다.

간단히 설명하며, 접근하고자 하는 공간 객체가 B2에 존재한다면 B1의 크기가 작아서 시간근접성을 유지하지 못한다고 판단하여 B1의 크기를 증가시킨다. 또한 B1에 접근하고자 하는 객체가 존재한다면 B1,B2의 크기가 적절하다고 생각하여 그 크기를 그대로 유지하고, 접근하고자 하는 객체가 B1이나 B2에 존재하지 않는다면 앞으로의 접근 패턴이 현재 B1에 유지된 시간 근접성과 다를 것이라고 판단하여 B1의 크기를 감소 시킨다 (자세한 사항은 [17] 참조).

그러나 지금까지 살펴본 공간 데이터베이스를 위한 버퍼 관리 기법들은 공간 객체의 정적 특성 (즉, MBR의 크기) 등만을 고려 함으로써, 동적인 객체 접근 패턴에 따른 효과적인 버퍼 관리 기법을 제공하지 못하고 있다.

3. 시간-공간 근접성을 통합한 공간 객체 버퍼 관리 기법

공간 데이터베이스에서는 최근 사용된 객체가 앞으로 다시 사용될 확률이 높다는 시간 근접성뿐 만 아니라, 특정 지역의 데이터가 자주 사용되는 공간 근접성[18]도 나타난다. 즉, 임의의 영역이 자주 접근된다면, 해당영역 안에 존재하는 공간 객체들도 자주 사용될 확률이 높게 된다.

공간 데이터베이스에서 시간 근접성만을 고려한다면 자주 접근되는 영역 안에 존재하는 공간 객체를 희생자로 선정할 경우가 발생되어 버퍼의 효율성을 떨어뜨리게 된다. 또한, 공간 근접성만을 고려할 경우에는 무작위 접근과 같은 형태의 접근 패턴을 효율적으로 지원할 수 없게 된다.

따라서, 본 논문에서 제안하는 버퍼 관리 기법인 BEAT는 기존의 데이터베이스를 위한 버퍼 관리기법들과 달리, 공간 근접성과 시간 근접성을 통합적으로 고려하는 특징을 지닌다. BEAT 관리 기법에서의 기본적인 경험적 규칙은 " 자주 접근되는 영역 안의 공간 객체들은 앞으로도 자주 사용된다" 라는 것이다.

3.1 BEAT의 기본 개념

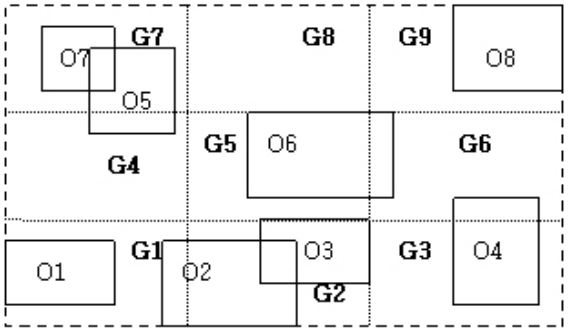
BEAT 기법에서는 우선 (그림 2)와 같이 공간 객체들이 존재하는 작업 공간을 균등 크기의 NxN 격자(grid)들로 분할한다.

여기서, 각 공간 객체의 MBR의 중심점이 어느 격자에 안에 존재하는 가에 따라서 각 공간 객체가 소속된 영역이 결정된다. 예로써. 격자 G2에는 공간 객체 O2와 O3가 포함되며, G3에는 공간 객체 O4가 포함된다. 즉. 임의의 공간 객체가 어느 영역에 포함되는지를 손쉽게 파악할 수 있다.

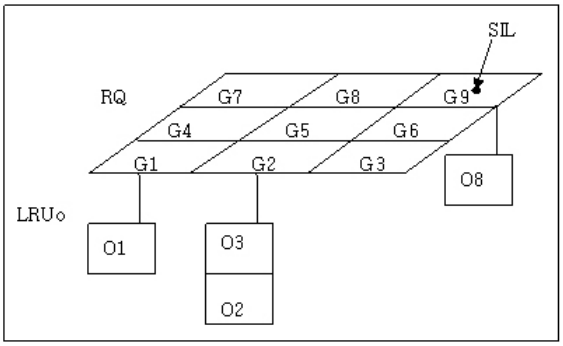
이렇게 분할된 공간 영역에서 어떠한 영역이 자주 사용되고 있는지를 파악하기 위하여 BEAT에서는 분할 된 공간 영역의 집합을 큐를 이용하여 관리 한다. 또한 각 영역 안에서 어떠한 공간 객체가 자주 사용되고 있는지를 파악하기 위하여 각 영역 안의 공간 객체들을 LRU 큐를 이용하여 관리한다. 즉, (그림 3)에서 보이는 바와 같이, BEAT에서는 자주 사용되는 영역들의 정보를 관리 및 각 영역에서 자주 사용되는 공간 객체들의 정보 관리를 하는 2 단계 버퍼 관리 기법을 이용한다. 여기서 RQ는 분할된 공간 영역을 관리하기 위한 큐 구조체이며 LRUo는 각 영역에서 관리하는 LRU 큐이다.

(그림 3)의 SIL (spatially interesting location) 은 공간적 관심 위치로써 현재 공간 데이터베이스에 자주 접근되는 위치를 나타낸다. SIL에 대한 추정 방법은 다음의 3.2절에서 자세히 설명한다.

BEAT기법에서 버퍼교체가 필요할 경우 LRUo가 empty가 아닌 RQ의 원소들 중에서 SIL에서 가장 먼 위치의 영역



(그림 2) 작업 공간의 분할



(그림 3) BEAT의 구조

```
SIL // 공간적 관심 위치
Procedure BEAT(Spatial_Object s){
        if a buffer is full
            replace(SIL)
        load s into a buffer
        compute_SIL(SIL, s); //3.2절 참조
}
Procedure replace(point SIL){
        find R from RQ which is the farthest from SIL
                    and whose LRUo is not empty
        find a victim v from R's LRUo
        drop v
}
```

(그림 4) BEAT 알고리즘

을 선택하고 이 영역에서 가장 오랫동안 사용되지 않은 (즉, least recently used) 공간 객체를 희생자로 선정한다. (그림 3)과 같은 상태에서 버퍼 상에 4개의 공간 객체만을 유지할 수 있다고 가정할 때, G1 영역의 중심점이 SIL에서 가장먼 영역임으로 G1의 객체 O1이 희생자로 선정된다. (그림 4)는 위에서 설명한 BEAT 기법의 개략적인 프로시져(Procedure)를 나타낸다.

따라서, 새로운 공간 객체를 접근 시에 BEAT 기법은 각 분할 영역의 LRU 큐를 시간 근접성과 공간 근접성을 통합적으로 고려하면서도 효율적으로 공간 데이터베이스의 버퍼를 관리한다.

3.2 공간 근접성의 측정

BEAT에서는 위에서 언급한 바와 같이 공간 근접성을 이용한다. 이 공간 근접성을 이용하기 위해서 접근하는 공간 객체들의 위치들에 따른 공간적 관심 위치(Spatial Interesting Location: SIL)를 추정하여야 한다.

BEAT 에서 초기 SIL 값은 작업 공간의 중심점으로 설정한다. 이후, SIL는 사용자의 관심의 변화에 따라서 지속적으로 변화한다. 현재 추정된 SIL 이 <xsil, ysil>이고 현재 접근한 공간 객체 p의 MBR 중심점이 <xp, yp> 라고 할 때, 새로운 SIL은 다음과 같은 수식을 통하여 예측된다.

$$xsil = xsil + wx(xsil - xp)$$
$$ysil = ysil + wy(ysil - yp)$$

SIL를 예측하는데 있어서, 일반적으로 사용자가 관심 있는 위치는 천천히 변화함으로, 일탈자(outlier)들을 고려해야만 한다. 현재 SIL와 새로 접근하는 객체의 거리가 멀어질수록 새로 접근한 객체의 위치가 사용자 관심 위치일 확률이 지수적으로 감소한다. 따라서 위의 수식에서 가중치 wx와 wy를 사용하여 이를 반영하며, 가중치 계산 함수는 간단한 감소함수 (1/e) 를 이용하였다. 가중치 계산식은 다음과 같다.

$$w_x = \left(\frac{1}{e}\right)^{|x_{sil} - x_p| / Domain_x}$$

위의 식에서 e는 자연 지수이고, Domainx는 작업공간의 x축 길이이다. 가중치 wy에 대한 계산식도 위의 수식과 유사하다. 위의 식에서 현재 SIL의 위치와 새로 접근한 객체의 거리가 증가될수록 가중치의 값이 감소한다.

3.3 작업 공간의 분할

BEAT는 영역의 개수에 따라서 흥미로운 특징을 나타낸다. 만약 분할된 영역이 한 개라면 (즉, 작업 공간 자체가 하나의 영역이다.), 하나의 영역 안의 객체들은 LRU 큐를 이용하여 관리함으로 BEAT는 LRU처럼 동작할 것이다. 즉, 시간 근접성 만을 고려하게 된다. 또한, 분할된 영역의 개수가 매우 많다면 각 분할된 영역에 하나의 공간 객체만이 포함될 것임으로 SIL에서 가장 먼 공간 객체를 희생자로 선정함으로 공간 근접성만을 고려하게 된다. 따라서, 공간 근접성과 시간 근접성을 동시에 고려하기 위해서는 적절한 수로 작업 공간을 분할하여야 한다.

그러나, 분할된 영역의 개수의 최적 값을 효율적으로 구하는 것은 매우 어려운 문제이다. 따라서, 우리는 간단한 경험적 규칙을 사용한다.

버퍼가 가득 찾을 때, 버퍼 상에 존재하는 객체의 개수를 X라고 하자. 공간 객체들의 평균크기는 일반적으로 SDBMS에서 메타 데이터로 관리되고 있으므로, X 값은 손쉽게 계산할 수 있다.

만약, 분할된 영역이 한 개이면, RQ에 원소는 하나이고, LRUo의 원소는 X 일 것이다. 이에 반하여, 분할된 영역의 개수가 매우 크다면, RQ의 원소는 X이고 LRUo의 원소 개수는 1 일 것이다. 따라서, 우리는 평균적으로 RQ의 원소 개수와 LRUo의 원소 개수가 같도록 작업 공간을 분할하여, 시간 근접성과 공간 근접성을 동등하게 고려하도록 하였다.

따라서, 분할된 영역의 개수를 N2 이라고 할 때 (3.1절에서 언급한 바와 같이 작업 공간은 NxN의 균등 크기로 분할하였으므로), N2를 $\sqrt{X}$ 로 설정한다. 따라서, 평균적으로, RQ의 원소 개수는 $\sqrt{X}$ 이고, 각 LRUo에 속한 공간 객체의 수도 $\sqrt{X}$ 가 된다. 따라서, 버퍼 상에 존재하는 공간 객체의 수는 X (=$\sqrt{X}$ x $\sqrt{X}$)가 된다.

4. 실험과 분석

본 절에서는 제안된 BEAT 기법의 성능 실험을 다룬다. 실험에서는 합성 데이터와 실제 지도 데이터를 이용하여 BEAT 기법과 기존의 다양한 버퍼 관리 기법들-LRU, 2Q, ARC 그리고 ASB-과의 적중률(hit ratio)을 비교하여 BEAT 기법의 우수성을 보인다. 위에서 언급한 바와 같이 LRU,2Q, ARC는 시간 근접성만을 고려하며, ASB는 시간 근접성과 공간 객체의 특성(즉, MBR의 크기)을 고려 한다. 우리는 다양한 크기의 버퍼에서 실제 데이터와 합성 데이터를 이용하여 BEAT의 성능을 측정하였다.

합성 데이터의 특징은 <표 1>에 나타나 있다.

〈표 1〉 합성 데이터의 특성

객체 수	10000
평균객체 크기	80byte(최대 128byte)
작업공간 크기	100000x100000
객체 분포	균등분포

공간 편향(skew)은 실제 데이터에 나타남으로 합성 데이터에서는 균등 분포를 사용하였다. 본 실험에서 사용한 실제 데이터는 미국 Bureau of Census[19]의 Tiger/Line 데이터에서 추출하였다. 실험 데이터는 켈리포니아 주 King County의 도로 정보를 이용하였으며 공간 객체의 수는 21,853 개이고 작업 공간의 크기는 840,681x700,366이다.

BEAT의 효율성을 평가하기 위하여 세가지 데이터 접근 유형을 만들었다. 첫 번째는 균등 접근 유형- 실험 결과에 Uniform으로 표기- 으로 모든 공간 객체의 접근 확률이 동일하다. 두 번째로 시간 근접의 효과를 측정하기 위하여 Zipf 분포를 이용한 시간적 편중 접근 유형- 실험 결과에 Time Skew로 표기-을 작성하였다. 시간적 편중 접근 유형에서는 전체 데이터 접근 중 80%의 접근이 약 20%의 공간 객체에 집중되어 있다. 마지막으로, 공간 편중 접근 유형- 실험 결과에 Spatial Skew로 표기-를 작성하였다. 이 접근 유형에서는90%의 데이터 접근이 작업 공간의 약 10% 크기의 영역에 집중되도록 하였다. 따라서, 공간 편중 접근 유형에서는 시간 근접성과 공간 근접성이 동시에 나타난다. 각 접근 유형에서 1,000,000번의 공간 객체 접근이 발생된다.

본 실험의 수행 환경으로 버퍼의 크기를 전체 공간 객체 크기의 총합의 5%, 10%, 20%로 설정하여 기존의 버퍼 관리 기법과 본 논문에서 제안하는 BEAT기법의 성능을 비교하였다.

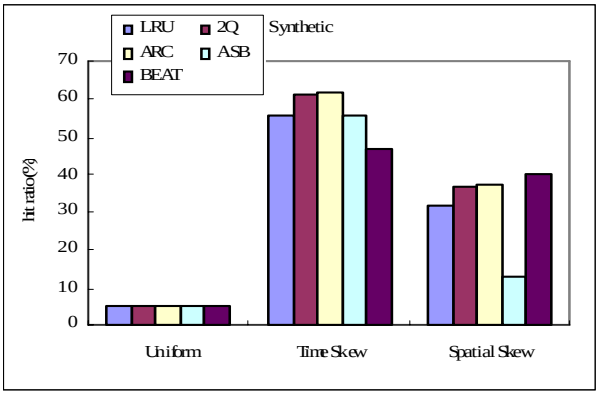
(그림 5), (그림 6)에서 보는 바와 같이, BEAT 기법은 모든 경우에 가장 좋은 적중률을 보이지는 않는다. 그러나, BEAT 기법은 Spatial Skew 유형에서 LRU, 2Q, ARC, ASB 보다 좋은 적중률을 보임을 알 수 있다. 또한 BEAT 기법은 LRU, 2Q, ARC에 비교하여, 시간 근접성 및 공간 근접성을 동시에 고려함에 있어서도 BEAT 기법은 Uniform 및 Time Skew 접근 유형에서도 뒤떨어지지 않는 우수한 성능을 보이고 있다.

ASB는 정적 특성인 MBR 크기를 고려함으로 대부분의 경우에 가장 나쁜 성능을 보인다. LRU 기법은 모든 경우에 가장 좋은 성능을 보이지 않지만 또한 가장 나쁜 성능을 보이지도 않는다. 2Q와 ARC는 시간적을 편중된 접근 유형을 위하여 개발되었기 때문에, Time Skew 유형일 때 좋은 적중률을 보인다.

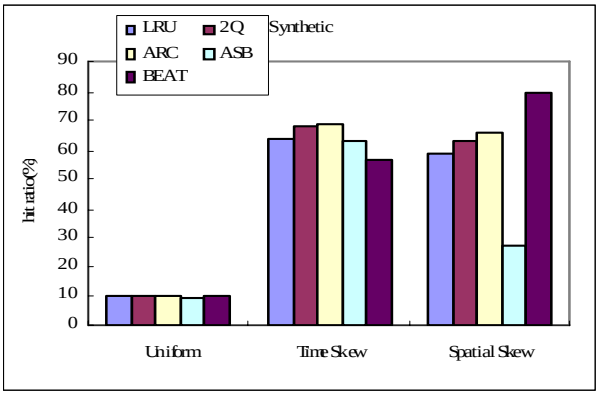
(그림 5), 합성 데이터의 Time Skew 접근 유형에서, BEAT 기법은 좋은 성능을 보이지 않는다. BEAT 기법은 분할된 영역 안에서 가장 오랫동안 사용되지 않은 공간 객체를 희생자로 선정한다. 여기서, 합성 데이터는 공간 객체

들은 균등 분포에 따라서 배치되어 있고 데이터 접근 유형은 공간 객체의 분포와는 독립적이다. 따라서, LRU와 같이 시간 근접성만을 고려하는 버퍼 관리 기법은 버퍼 상에 존재하는 모든 객체들 중에서 희생자를 고른다. 그러나 BEAT는 하나의 분할 영역 내에 존재하는 공간 객체들 중에서 희생자를 선정함으로써 합성 데이터의 Time Skew 유형에서 다른 기법들에 비하여 좋은 성능을 보이지 못한다.

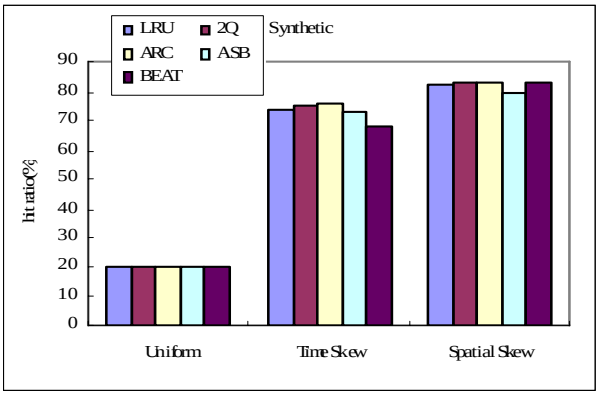
그러나 (그림 6)의 실제 데이터의 경우에는 Time Skew 접근 유형 시, BEAT가 좋은 성능을 보여준다. 실제 데이터에서는 공간 데이터의 분포에 편향(Skew)가 존재하며, 의도하지 않았지만 Time Skew 접근 유형에 공간 근접성이 발생된다. 따라서, BEAT가 다른 기법들과 비교하여 적절한 성능을 보인다. 더욱이 (그림 6)의 (a),(b),(c)에서 보듯이 실제 데이터를 이용한 실험에서는 다양한 접근 유형에서 BEAT가 시간 근접성과 공간 근접성을 동시에 활용함으로써 좋은 성능을 보임을 알 수 있다. 특히 Spatial Skew 유형에 대하여서는, 합성 데이터 및 실제 데이터에 대하여 뛰어난 성능을 보임을 알 수 있다.



(a) 5% 크기의 버퍼

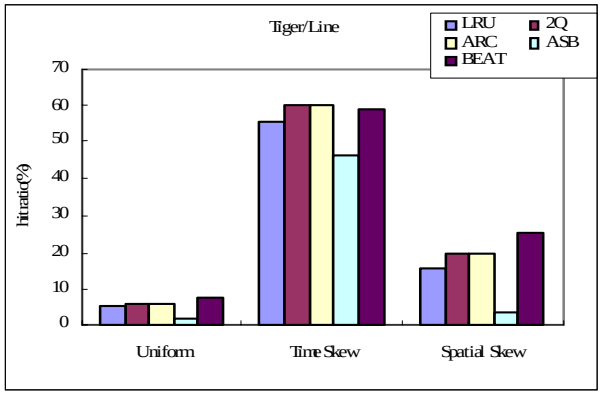


(b) 10% 크기의 버퍼

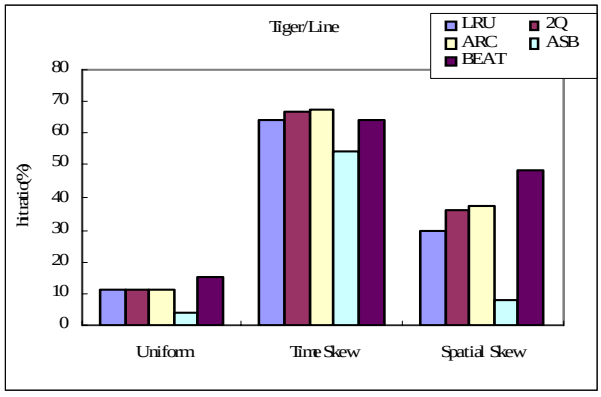


(c) 20% 크기의 버퍼

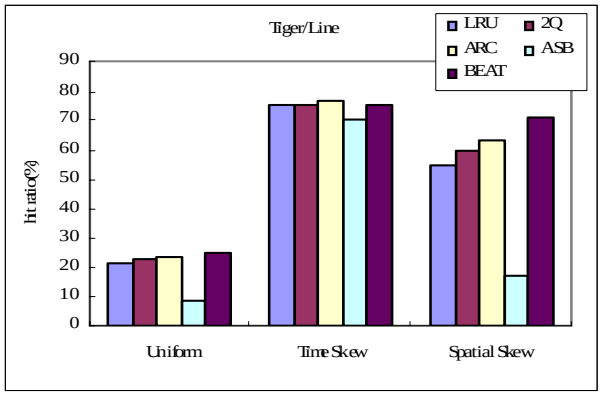
(그림 5) 합성 데이터의 적중률



(a) 5% 크기의 버퍼



(b) 10% 크기의 버퍼



(c) 20% 크기의 버퍼

(그림 6) 실제 데이터의 적중률

결론적으로 BEAT는 다양한 버퍼 크기에서 다양한 접근 유형에 대하여 기존의 버퍼 관리 기법에 뒤떨어지지 않는 성능을 보이며, 특히, 공간 근접성과 시간 근접성이 동시에 나타나는 공간 데이터베이스 환경에서는 BEAT 기법이 다른 기법들보다 뛰어난 성능을 보인다.

5. 결 론

데이터베이스 분야에서, 효율적인 버퍼관리 기법은 디스크 I/O를 줄일 수 있는 주요 요소 중 하나임으로 많은 연구자들이 이에 대한 연구를 진행해 왔다. 이러한 대부분의 버퍼 관리 기법은 시간 근접성만을 고려한다. 따라서 본 논문에서는 BEAT라는 새로운 버퍼 관리 기법을 제안한다. 공간 데이터베이스 환경에서는 최근 접근한 영영 안에 존재하는 공간 객체가 가까운 미래에 접근된다는 공간 근접성이 존재한다.

따라서, 시간 근접성과 공간 근접성을 동시에 고려하기 위하여 BEAT에서는 2단계 버퍼 관리 기법을 사용한다. BEAT에서 작업 공간을 분할된 영역들로 분리하고 공간 근접성은 SIL이라는 공간 관심 위치를 추정하여 관리한다. 또한 각 분할된 영역에서는 LRU 큐를 이용하여 영역 내에 존재하는 공간 객체들의 시간 근접성을 관리한다.

BEAT 기법의 효율성을 보이기 위하여, 실제 데이터와 합성 데이터를 이용하여 다양한 버퍼 크기와 접근 유형에 대하여 실험을 수행하였다. 실험 결과, BEAT 기법이 공간 데이터베이스 환경에서 기존의 버퍼 관리 기법들에 비하여 우수함을 보였다.

본 연구에서 우리는 간단한 경험 규칙을 이용하여 작업 공간을 버퍼의 크기에 따라서 분할하도록 하였다. 따라서, 후속 연구로써, 접근 유형에 따라서 분할되는 영역의 개수를 변형시키는 방법에 대하여 연구하고자 한다.

참 고 문 헌

[1] R. H. Guting, “An Introduction to Spatial Databases Systems,” VLDB 3, pp.357-399, 1994.

[2] T. Brinkhoff, H. Kriegel, R. Scheneider, B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", in proceedings of ACM SIGMOD Conference, pp.322-331, 1990.

[3] J.K. Min, H.H. Park, C.W. Chung, "Multi-way spatial join selectivity for the ring join graph", Information and Software Technology 47(12), pp.785-795, 2005.

[4] A. Papadopoulos, and Y. Manolopoulos, “Global Page Replacement in Spatial Databases,” in proceedings of DEXA, 1996.

[5] W. Effelsberg, “Principles of Database buffer Management,” ACM TODS, 9(4), pp.560-595, 1984.

[6] E. J. O'Neil, P. E. O'Neil, and G. Weikum, “The LRU-K Page Replacement algorithm for database disk buffering,” in proceedings of ACM SIGMOD, pp.297-306, 1993.

[7] T. Johnson, and D. Shasha, “2Q: a Low Overhead High Performance Buffer Management Replacement Algorithm,” In proceedings of VLDB Conference, pp.439-450, 1994.

[8] A.J. Tung, Y.C. Yay, and H. Lu, “BROOM: Buffer Replacement using Online Optimization by Mining,” in proceedings of CIKM, pp.185-192, 1998.

[9] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, “LRFU: A Spectrum of Policies that subsumes the Least Recently Used and Least Frequently Used Policies,” IEEE Trans. Computers, 50(12), pp.1352-1360, 2001.

[10] N. Megiddo and D. S. Modha, “ARC: A Self-tuning, Low Overhead Replacement Cache,”in proceedings of USENIX FAST Conference, 2003.

[11] L. B. Sokolinsky, “LFU-K: An Effective Buffer Management Replacement Algorithm,” in proceedings of DASFAA Conference, pp.670-681, 2004.

[12] B. Juurlink, “Approximating the Optimal Replacement Algorithm,” in proceedings of ACM CF Conference, 2004.

[13] G. M. Sacco, “Index Access with a Finite Buffer,” in proceedings of VLDB, 1987.

[14] C. H. Goh, B. C. Ooi, D. Sim, K. Tan, “GHOST: Fine Granularity Buffering of Index,” in proceedings of VLDB, 1999.

[15] A. Papadopoulos, Y. Manolopoulos, “Global Page Replacement in Spatial Databases,” in proceedings of DEXA. 1996.

[16] I. Kamel, C. Faloutsos, "On Packing R-Trees," in proceedings of CIKM, pp.490-499, 1993.

[17] T. Brinkhoff, “A Robust and Self-tuning Page Replacement Strategy for Spatial Database Systems,” in proceedings of EDBT, pp.533-552, 2002.

[18] L. Ki-Joune and L. Robert, “The Spatial Locality and a Spatial Indexing Method by Dynamic Clustering in Hypermap System,” Advances in Spatial Databases, pp.207-223, 1990.

[19] U.S. Census Bureau, “UA Census 2000 TIGER/Line Files,” http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html.

민 준 기

e-mail : jkmin@kut.ac.kr

1995년 숭실대학교 전자계산학과(공학사)

1997년 한국과학기술원 전자전산학과(공학석사)

2002년 한국과학기술원 전자전산학과(공학박사)

2002년~2003년 한국과학기술원 연수연구원(Post Doc.)

2003년~2004년 한국과학기술원 초빙교수

2004년~2005년 전자통신연구원 선임연구원

2005년~현 재 한국기술교육대학교 인터넷미디어공학부 조교수

관심분야 : Query Processing, XML, Stream Data, Sensor Network