SLAM : 공간 데이타의 공간적 근접성을 이용한 효율적인 버퍼관리기법

(SLAM : An Efficient Buffer Management Strategy using Spatial Locality of Spatial Data)

안 재 용 † 민 준 기 † 정 진 완 ††
(Jae-Yong Ahn) (Jun-Ki Min) (Chin-Wan Chung)

요 약 데이타베이스 관리시스템에서 중요한 문제중의 하나는 효율적인 버퍼관리이다. 데이타베이스 관리시스템에서 객체를 디스크에서 읽어오는 작업은 많은 비용을 필요로 하기 때문에 시스템의 성능을 향상시키기 위해서는 디스크 I/O의 횟수를 최소화하는 것이 매우 중요하다. 지금까지 디스크 I/O 횟수를 줄이기 위한 많은 버퍼관리기법들이 제안되었지만, 그 기법들은 시간적 근접성만을 고려하기 때문에 공간적 근접성도 존재하는 공간데이타베이스 환경에서는 좋은 성능을 보여주지 못했다.

본 논문에서는 공간데이타베이스의 시간적 근접성과 공간적 근접성을 동시에 고려하는 새로운 버퍼관리기법인 Spatial Locality Area Measure(SLAM) 기법을 제안한다. 제안한 버퍼관리기법은 SLM-tree와 M-LRU, 두 개의 구조체로 구성되었으며 공간데이타베이스 환경에서의 다양한 버퍼크기와 참조빈도에 대한 실험에서 뛰어난 성능을 보여준다.

키워드 : 공간데이타베이스, 버퍼관리, 공간적 근접성, 시간적 근접성

Abstract One of the major issues of DBMS is the buffer management. Because fetching data from the database disk is costly, the number of disk I/O's must be minimized in order to improve the DBMS performance. Although there have been many buffer management strategies to minimize the disk I/O, those strategies usually focused on just the temporal locality. Since there are the spatial locality as well as the temporal locality in the spatial database, strategies using only the temporal locality cannot achieve the optimal performance in the spatial database.

In this paper, we propose a new buffer management strategy, the Spatial Locality Area Measure(SLAM) strategy, that considers not only the temporal locality but also the spatial locality. The SLAM buffer management strategy consists of two core structures, the SLM-tree and the M-LRU. We show the efficiency of the proposed strategy through experiments over various buffer sizes and reference frequencies.

Key words : spatial database, buffer management, spatial locality, temporal locality

1. 서 론

기존에는 데이타베이스 관리시스템(Database Management System: DBMS)이 문자, 숫자와 같은 간단한 구조의 정보들만을 관리해 왔기 때문에, 기존의 DBMS를 이용하여 지리정보 시스템 분야에서 필요로 하는 복잡한 공간 객체 정보를 저장 관리하는 데는 많은 제약이 존재한다. 따라서 공간 객체를 효율적으로 처리할 수 있는 새로운 형태의 DBMS가 필요하게 되었으며 이런 요구에 의해 등장하게 된 것이 공간데이타베이스 관리시스템(Spatial Database Management System: SDBMS)이다. SDBMS는 관리하는 공간 객체들이 다양한 복잡도를 갖고 있으며, 전체 데이타의 크기가 100 테라바이트(terabyte)에 이르기도 하는 등 대용량의 데이타를 다룬다는 특징을 갖는다. 이와 같은 SDBMS의 특징들은 시

· 본 연구는 한국과학재단 특정기초연구(과제번호:1999-2-315-001-3) 지원으로 수행되었음.
† 비 회 원 : 한국과학기술원 전산학과
jyahn@islab.kaist.ac.kr
jkmin@islab.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
chungcw@islab.kaist.ac.kr
논문접수 : 2001년 8월 11일
심사완료 : 2002년 7월 25일

스템의 성능을 저하시키는 요인이 되기도 한다.

DBMS는 참조하고자 하는 객체가 버퍼에 존재하지 않을 때, 디스크의 객체에 접근하여 버퍼에 저장하는 작업이 실행되는데 이 작업은 많은 비용이 들기 때문에 객체를 가져오는 횟수를 줄이기 위한 효율적인 버퍼관리에 대한 연구는 매우 중요하다. SDBMS는 앞에서 말했듯이 대용량의 데이타를 관리하므로 효율적인 버퍼관리 문제는 일반 DBMS 환경에서 보다 훨씬 중요한 문제가 된다.

지금까지 연구된 버퍼관리기법들은 DBMS의 버퍼를 관리 하기 위한 정보로써 시간적 근접성만을 고려하고 있다. 그러나 SDBMS에는 시간적 근접성 뿐만 아니라 공간적 근접성(spatial locality)[1, 2, 3]도 동시에 존재하기 때문에 이 두 가지 근접성 정보를 통합하여 이용하면 보다 효율적으로 데이타를 관리할 수 있다.

본 논문에서는 시간적 근접성과 공간적 근접성을 동시에 고려하여 버퍼를 관리함으로써 SDBMS의 성능을 향상시키는 SLAM 버퍼관리기법을 제안한다. SLAM은 실행시간(Run-Time)중에 공간적 근접성 영역(spatial locality area)을 추정하기 위한 수단으로 SLM-tree를 이용하며 버퍼의 객체를 관리하기 위한 구조체로 M-LRU를 이용한다. 공간 데이타 특성을 가장 잘 표현하는 것이 객체지향모델(Object-Oriented Model)[4]이므로 본 논문에서 제안하고 있는 기법은 객체지향데이타베이스 관리시스템(Object-Oriented Database Management System:OODBMS)을 기반으로 하고 있다[5, 6].

본 논문의 구성은 다음과 같다. 2장 관련연구에서는 기존의 버퍼관리기법들과 본 논문에서 고려하는 SDBMS의 특징에 대해 설명하고 있으며 3장에서는 본 논문에서 제안하는 SLAM 버퍼관리기법에 대해 기술하고 있다. SLAM이 가진 문제를 보완하기위해 고안된 수정된 SLAM 버퍼관리기법은 4장에서 다루고 있다. 5장에서는 여러 가지 다양한 환경에서 SLAM 버퍼관리기법이 기존의 방법들에 비해 얼마나 성능 향상을 보이는지 실험한 내용을 다루고 있으며 마지막으로 6장에서는 결론에 대하여 논의한다.

2. 관련 연구

지금까지 버퍼를 효율적으로 관리하기 위한 버퍼 교체 알고리즘에 대하여 많은 연구들이 진행되어 왔다. 대부분의 상업적인 목적의 DBMS들은 LRU[7] 알고리즘을 사용하는 것으로 알려져 있다[8]. LRU 알고리즘은 '마지막에 참조된 시간'이라는 너무 한정된 정보만을 이용하여 버퍼에서 교체할 대상을 찾는 한계가 있었기 때문에 LRU-k[9] 알고리즘이 LRU 알고리즘의 단점을 보완하기 위해서 제안되었다. LRU-K 알고리즘은 데이타가 참조되는 빈도를 통계적으로 예측하고, 이 정보를 이용해서 참조가 되는 빈도가 가장 적은 데이타를 교체하게 된다[9]. 그러나 LRU-k 알고리즘도 각각의 데이타에 접근할 때마다 우선순위 큐(priority queue)를 관리하기 위해서 많은 비용이 든다는 문제를 가지고 있다. 2Q[10] 알고리즘은 이런 LRU-k 알고리즘의 문제를 해결하면서 가장 효율적인 LRU-k 알고리즘인 LRU-2[9] 수준의 성능을 보여 줄 수 있는 알고리즘으로 제안되었다. 2Q 버퍼관리기법은 2개의 LRU 큐(queue)와 하나의 FIFO 큐를 이용하여 별도의 데이타를 저장하기 위한 부담 없이 많이 사용되는 데이타들만을 버퍼에 남게 하는 방법을 취한다.

그러나 위의 기법들은 SDBMS 환경에서 나타나는 다양한 특징들을 고려하지 않고 있으므로 SDBMS 환경에서의 버퍼 교체 알고리즘으로는 적당하지 않다.

SDBMS는 공간 데이타를 관리하기 위하여 고안되었기 때문에 기존의 DBMS들과는 많은 차이점을 보인다[11]. SDBMS의 질의들은 지도 전체에 고르게 분포되어 있지 않고 지도에서 특정한 부분에 집중된다는 특징을 보여준다. 더욱이 데이타의 분포 또한 특정 부분에 집중되어 나타난다. 그림 1은 전체 지도상에서 질의 영역이 집중되는 예를 보여 준다. 이러한 질의 영역의 집중 현상을 공간적 근접성[1, 2, 3]이라 한다.

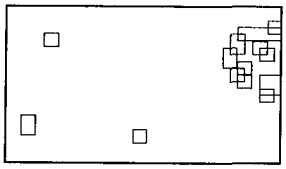


그림 1 SDBMS에서 질의의 분포

일반적으로 근접성(locality)은 공간적 근접성(Spatial locality)과 시간적 근접성(temporal locality)의 두 가지로 나눌 수 있다[1, 3]. 공간 데이타를 관리할 때 서로 다른 속성의 두 가지 근접성을 잘 고려하여 이용하면 시스템의 성능을 향상시킬 수 있다. [1]은 공간적 근접성을 이용하여 공간 인덱스(spatial index)의 구조를 최적화하였으며, [2]는 공간적으로 근접한 데이타들을 연속된 페이지에 클러스터링(clustering) 함으로써 공간적 근접성을 이용하였다. 본 논문에서는 버퍼관리에서 시간적 근접성과 함께 공간적 근접성을 효과적으로 이용하는 방법에 대하여 연구한다.

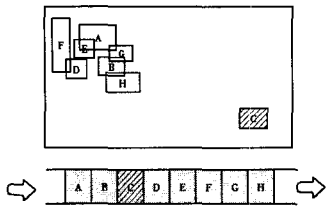


그림 2 LRU에서의 버퍼교체전략

3. SLAM 버퍼관리기법

본 논문에서 제안하는 SLAM 버퍼관리기법은 SDBMS에서 시간적 근접성과 공간적 근접성을 통합 관리함으로써 위의 문제를 해결하고 버퍼의 효율성을 향상시키고 있다.

그림 2는 LRU에서의 비효율적인 버퍼교체전략을 보여준다. 그림에서 볼 수 있듯이 버퍼에 존재하는 객체들 중에서 객체 **C**를 제외한 객체들은 공간적으로 근접해 있다. 이 객체들은 공간적 근접성을 나타내게 되고 재참조될 가능성이 높으므로 객체 **C**를 먼저 교체하는 것이 바람직하다. 그러나 LRU의 교체 전략에 의하면 객체 **H**가 사용된 지 가장 오래 되었으므로 객체 **H**를 교체하게 된다. 이 경우 당연히 재 참조될 가능성이 높은 객체를 교체 대상으로 선택하기 때문에 시스템의 성능이 저하된다.

SLAM은 가장 최근에 사용된 객체들이 존재하는 위치가 대개의 경우 현재의 공간적 근접성 영역과 유사하다는 관찰 결과를 이용한다. SLAM은 최근에 사용된 객체들이 존재하는 영역을 공간적 근접성 영역으로 추정하고, 버퍼의 교체 대상 객체를 선정하는 과정에서 추정된 공간적 근접성 영역에 속하지 않는 객체들을 먼저 교체한다. 이는 공간적 근접성 영역에 속하는 객체가 앞으로 참조될 가능성이 더 많다는 사실을 기반으로 한다. 위와 같이 버퍼를 관리하는 SLAM 은, SDBMS에서 재참조될 가능성이 적은 객체들을 교체 대상으로 선정하기 때문에, 기존의 방법들보다 뛰어난 성능을 보여준다.

SLAM 버퍼관리시스템은 그림 3과 같이 SLM-tree와 M-LRU의 두 가지 구조체로 이루어져 있다.

- **SLM-tree** - 최근에 사용한 ***k***개 객체들의 MBR (Minimum Bounding Rectangle)[11] 정보들을 별도로 저장 관리하여 최근에 사용된 객체들을 모두 포함하는 MBR 정보를 '현재의 공간적 근접성 영역'으로 제공한다.
- **M-LRU** - ***n***개 객체를 버퍼에 관리하고 '현재의 공간적 근접성 영역'을 이용하여 저장된 객체들 중에서 추정된 공간적 근접성 영역 안에 속하지 않는 객체들을 가장 먼저 교체되어야 할 대상으로 선정하는 역할을 수행한다.

각 구조체들에 대한 세부적인 내용은 3.1절과 3.2에서 설명하고 있고 3.3절에서는 SLAM에서 버퍼를 관리하는 알고리즘을 제시하고 있다.

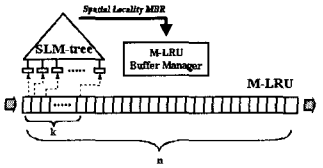


그림 3 SLAM 버퍼관리 시스템

3.1 SLM-tree

3.1.1 SLM-tree의 구조

본 논문에서는 공간적 근접성을 추정하기 위하여 버퍼에 있는 m개의 객체들 중에서 가장 최근에 참조된 k개의 객체들의 MBR정보를 이용한다. 즉, k개의 MBR을 포함하는 MBR을 '현재의 공간적 근접성 영역'으로 추정한다.

SLM-tree는 이진 트리 구조로 구성되었다. 말단 노드에는 최근에 이용된 ***k***개의 객체 MBR들이 저장되며, 트리(tree)의 내부 노드는 자식 노드들의 MBR들을 포함하는 MBR을 저장하도록 트리를 구성하여 루트 노드에는 말단 노드에 있는 모든 객체들의 MBR을 포함하는 MBR이 만들어 진다. 또한 SLM-tree는 이진 트리를 이용하므로 말단 노드의 MBR 정보가 바뀔 때 마다 ***O(log k)*** (단, ***k***는 말단 노드의 수)의 적은 비용으로 루트 노드에 변화를 반영시킬 수 있다는 장점을 가지고 있다.

최근에 이용된 8개의 객체들의 MBR들을 고려하는 SLM-tree는 그림 4와 같이 구성된다. (A)는 (B)의 SLM-tree가 고려하고 있는 실제 지도상의 객체들과 MBR들이다. ***O1***에서 ***O8***까지는 최근에 사용된 8개의 객체들의 MBR들이며 그것들을 포함하는 ***A***에서 ***G***까지의 MBR

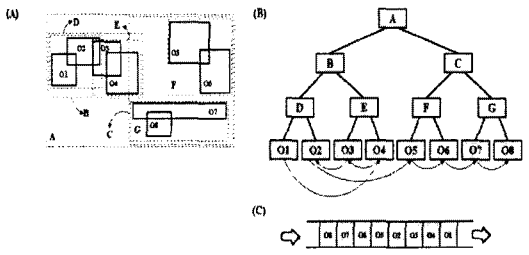


그림 4 SLM-tree 의 구조와 구성

은 SLM-tree를 구성하는 내부 노드들의 MBR들이다. (B)는 (A)의 객체들이 SLM-tree를 이루고 있는 모습이다.

그림 4의 (B)에서 말단 노드들은 모두 최근에 참조된 객체들이다. 각 내부 노드들은 두개의 자식 노드들을 모두 포함하는 MBR을 유지하므로 루트 노드 ***A***의 MBR은 모든 말단 노드의 객체를 모두 포함하는 MBR을 가지게 된다. 이는 그림 4의 (A)와 (B)를 비교하여 확인할 수 있다. (A)의 노드 ***D***는 (B)에서 자식노드에 해당하는 객체 ***O1***과 ***O2***를 포함하는 MBR이 되며 노드 ***B***는 노드 ***D***의 MBR과 노드 ***E***의 MBR을 포함하는 MBR이 된다. 그리고 마지막으로 루트 노드인 ***A***는 노드 ***B***와 노드 ***C***의 MBR을 모두 포함하는 MBR이 되고 이 MBR은 말단 노드에 존재하는 모든 객체의 MBR을 포함하는 MBR이 된다. 그리고 이 루트 노드 ***A***는 최근에 참조된 객체들의 분포를 반영하고 있으므로 최근의 공간적 근접성을 반영하는 공간적 근접성 영역으로 가정하고 사용할 수 있다.

그림 4 (B)의 SLM-tree 말단 노드들간의 실선은 객체들의 사용된 순서를 나타낸다. SLM-tree는 가장 최근에 사용된 ***n***개 객체들의 MBR들만을 고려하므로 SLM-tree가 새로 참조된 객체의 MBR을 고려해야 할 때 SLM-tree에서 가장 오래된(Least recently referenced) 객체의 MBR을 트리에서 제거해야 한다. 이를 위해서 트리의 말단에 존재하는 객체들의 참조 순서를 관리한다. 그림 4의 (C)는 B의 말단 노드 객체들이 사용된 순서를 큐의 형태로 보여주고 있다.

3.1.2 SLM-tree의 생성

SLM-tree가 공간적 근접성 영역을 추정하기 위해 이용할 객체의 수 ***k***가 정해지면 ***k***개의 말단노드를 가질 수 있는 가장 낮은 레벨의 트리를 생성하게 되며 이 트리는 노드의 생성, 삭제, 혹은 이동과 같은 구조변경 없이 이용된다. 이러한 특징들 때문에 SLM-tree는 실제 구현에서 배열(array)구조를 이용하여 고정된 형태로 구현될 수 있으며, 따라서 포인터(pointer)를 이용하여 동적으로 트리를 관리하는 것보다 트리 유지 비용을 많이 줄일 수 있다. 물론 그림 5에서 보여지듯이 트리에 사용되지 않는 노드들이 존재하기 때문에 메모리가 낭비된다는 단점들이 있지만, SLM-tree 각각의 노드는 MBR만을 저장하기 때문에 메모리의 낭비도 크지 않다.

SLM-tree는 이진 트리이기 때문에 식을 통해 트리의 레벨 ***n***을 찾을 수 있다. 그림 5는 SLAM이 공간적 근접성 영역을 추정하기 위해서 5개의 최근 사용 객체 정보를 이용할 경우에 생성되는 SLM-tree의 구조를 보여주고 있다. 5개의 말단 노드를 가져야 하기 때문에 높이

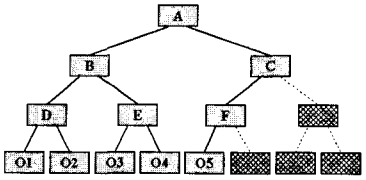


그림 5 5개의 말단노드를 가진 SLM-tree

가 3인 트리를 생성한다. 생성된 트리는 말단 노드가 $8(=2^3)$개 이기 때문에 필요한 5개의 말단노드 보다 3개가 더 많게 된다. 남는 노드들은 사용되지 않을 것이므로 말단 노드들중에서 왼쪽에서부터 5개의 노드들만 사용하고, 나머지 3개는 사용하지 않는다. 만일 트리의 내부노드들중에 자식노드들이 모두 사용되지 않는 노드가 있다면 그 노드들도 사용하지 않는다. 그림 5의 빗금쳐진 노드들은 이런 이유들로 사용되지 않을 노드들이다. 버퍼가 사용될 때 참조되는 객체의 정보들은 ***O1*** ~ ***O5***의 노드들에 저장되게 된다.

3.1.3 객체 참조

처음 SLM-tree가 생성된 후에 객체 참조가 시작되었을 때는 SLM-tree의 말단 노드들이 모두 채워질 때까지 참조된 객체들의 MBR 정보를 SLM-tree에 삽입만 한다. 이후에 SLM-tree의 말단 노드가 모두 채워지면 그때부터 SLM-tree의 추정된 공간적 근접성 영역을 사용하게 된다.

그림 6은 SLM-tree가 구축되어 있는 상황에서 새로운 객체가 참조 되었을 때 SLM-tree가 추정하는 공간적 근접성 영역에 어떻게 반영되는 지를 보여준다. 그림 6의 Step 1은 SLM-tree가 유지되는 중의 상태이다. 말단 노드들간의 참조 순서 리스트를 통해 가장 최근에 참조된 객체는 ***O8***이고 가장 오래된 객체가 ***O1***임을 알 수 있다. Step2는 새로운 객체인 ***O9***이 참조되었을 때의 변경된 SLM-tree이다. 항상 최근에 참조된 8개의 객체들만을 고려한다고 하였으므로 새로운 객체인 ***O9***을 트리에 반영하기 위해서 지금까지 고려했던 8개의 객체들 중에 참조된 지 가장 오래된 객체인 ***O1***을 트리에서 제거해야 한다. Step2의 큐는 고려대상중인 객체들의 큐에서 가장 오래된 객체 ***O1***이 나가고 새로운 객체 ***O9***이 추가되는 모습을 보여준다.

일반적으로 균형 잡힌 동적인 이진 트리에 데이타를 삽입하는 것은 적지 않은 비용이 든다. 그러나 SLM-tree에서는 항상 한 개의 객체가 나가고 하나의 객체가 들어오므로 삭제되는 객체의 MBR 위치에 새로운 객체의 MBR 정보만을 교체 저장하여 구조 변화 없이 트리를 유지하고 ***O(log n)***의 적은 비용으로 객체를 삽입,

삭제할 수 있다.

새로 삽입된 객체는 큐의 가장 끝에 놓여야 하기 때문에 트리 말단 노드들간의 순서 리스트는 그림 6의 step2와 같이 바뀌게 된다. 일단 ***O1***이 있던 자리에 ***O9*** 이 들어가게 되면, 변경된 말단 노드의 상태를 루트 노드 MBR에 반영해야 한다. 이를 위해서 바텀-업 업데이트(bottom-up update) 방법이 사용된다. ***O9***의 부모 노드인 ***D***는 자식 노드의 정보가 변경되었으므로 새로 ***O9*** 과 ***O2***를 포함하는 MBR을 가지는 ***D'***으로 바뀌게 된다. ***D***가 ***D'***으로 바뀌었으므로 ***D***의 부모 노드인 ***B***는 ***D'***과 ***E***의 MBR을 포함하는 새로운 MBR을 갖는 ***B'***이 된다. 이와 같이 반복적으로 부모 노드들에 자식노드의 변경을 반영하면 루트 노드***A***는 ***A'*** 노드로 바뀌게 된다. 이 외의 노드들은 자식 노드들에 아무런 변화가 없으므로 수정될 필요가 없다. 그림 6의 step2에서 실선으로 채워져 있는 노드들만 값이 반영되면 모든 작업이 끝난다. 작업이 끝나면 최종적으로 루트 노드인 ***A'*** 노드에는 말단노드의 변화를 반영하는 MBR이 생긴다.

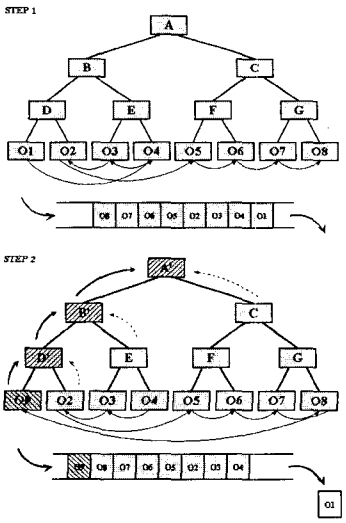


그림 6 SLM-tree의 새로운 객체 참조

```
Procedure insertNewObj (Onew)
{
1: newObjMBR := the MBR of Onew
2: currentNode := the oldest node among the SLMtree leaf nodes
3: Write the newObjMBR on the currnetNode
4: Reset the currentNode as the newest Node in the 'leaf node list'
5: while (the currentNode has a parent)
6:     { parentNode := the parent node of the currentNode
7:       parentNode.MBR := MBR including MBRs of two children
8:     }
}
```

그림 7 SLM-tree의 객체 참조 반영 알고리즘

그림 7은 새로운 객체가 참조되었을 때 SLMtree에 그 내용이 반영되는 알고리즘이다.

3.2 M-LRU

M-LRU는 버퍼의 객체들을 LRU 큐 형태로 유지하고 버퍼에서 객체를 교체해야 할 때 큐의 뒤에서부터 검사하여 SLM-tree에서 제공되는 '추정된 공간적 근접성 영역'과 겹치지 않는 객체를 버퍼에서 교체한다. 따라서, 처음 객체를 삽입할 때는 LRU 큐처럼 한 쪽 끝에서 삽입하지만 객체를 빼낼 때에는 중간에 있는 객체를 빼낼 수 있게 고안 되었으며 리스트 중간의 객체를 교체 대상으로 선정할 수 있도록 하기위해서 객체를 리스트의 뒤에서부터 탐색하는 기능을 가지고 있다.

그림 8은 교체 대상을 선정하는 과정이 어떻게 동작하는 지를 보여주고 있다. 그림 8의 (A)에서 점선으로 된 부분은 3.2절의 SLM-tree를 이용해서 공간적 근접성 영역으로 추정된 영역이며 A에서 F까지의 사각형은 버퍼에 저장된 각 객체의 MBR들을 의미한다. 또한 그림 8 (B)의 큐는 M-LRU에 어떤 순서로 객체들이 저장되어 있는 지를 보여 주고 있다. 큐에서 A가 있는 쪽이 버퍼에서 가장 오래 사용되지 않은 객체가 있는 부분이다.

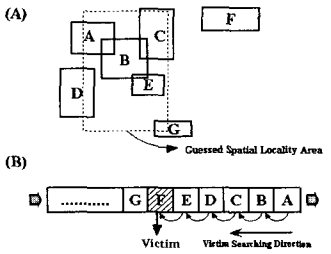


그림 8 M-LRU의 교체 대상 선정

교체 대상을 선정하는 작업은 다음과 같은 단계로 진행된다. 먼저 시간적 근접성에 근거하여 교체대상 객체를 큐의 뒤에서부터 찾게 된다. 처음에는 ***A***가 SLM-tree를 통해 추정된 공간적 근접성 영역과 겹치는 지를 확인한다. 그림 8의 (A)에서 객체 ***A***의 MBR이 추정된 공간적 근접성 영역과 겹치는 것을 알 수 있다. ***A*** 객체는 공간적 근접성 영역에 속하는 객체이므로 교체 대상으로 선정하지 않는다. 그리고 다음의 객체들에 대하여도 순서대로 겹침 여부 테스트를 한다. 그림 8의 (B)처럼 순차적으로 ***A***에서 ***E***까지 모든 객체가 추정된 공간적 근접성 영역과 겹치므로 ***F***까지 조사를 하게 된다. 객체 ***F***의 경우 그림 8의 (A)에서 확인할 수 있듯이

공간적 근접성 영역과 겹치지 않는다. 따라서 객체 F는 공간적 근접성 영역에 포함되지 않는 객체이므로 교체 대상으로 선정하게 된다.

이와 같은 방법을 통해서 공간적 근접성 영역 밖에 존재하는 객체를 쉽게 버퍼에서 교체할 수 있다.

```
Procedure object getObject (oid)
{
1: if  (the object is found in the buffer)
2:    then return the found object
3:    else {
4:        fetch the requested object from the database
5:        while (the size of the fetched object is bigger than the
                  space in the buffer)
6:              replaceObject()
7:        insertObject(the fetched object) }
8: return the fetched object
}

Procedure replaceObject()
{
1: mbr := the 'Root MBR' of the SLMtree
2: for each object oi in o1, o2, ..., on which are located in the
          M-LRU in order
3:  {let r  be the MBR of oi
4:   test whether r  is overlapping with 'mbr'
5:   if (r overlaps with 'mbr')
6:      then go to 3 and repeat
7:      else { replace oi
8:             return;   }
9:  }
}

Procedure insertObject (Onew)
{
1: Insert Onew into the buffer
2: SLMtree.insertNewObj(Onew)
}
```

그림 9 SLAM 버퍼 관리 알고리즘

3.3 버퍼 관리 알고리즘

표1은 앞에서 설명한 버퍼 교체 전략을 이용하여 버퍼를 관리하는 알고리즘을 나타낸 것이다. 참조는 get Object() 함수를 이용한다. 만일 객체가 버퍼에 있다면 그 객체를 찾아서 제공한다(1~2단계). 그러나 그렇지 않다면, 객체를 데이타베이스에서 읽어온 후 객체를 제공한다(3~8단계). 이 과정에서 새로 가져올 객체를 저장할 공간을 확보해야 하는데 이는 replaceObject()함수를 통해 이루어지며(5~6단계), 공간이 확보된 후에 객체를 버퍼에 저장하는 작업은 insertObject()함수를 통해이루어진다(7단계).

replaceObject() 함수는 앞에서 설명한 M-LRU의 관리 방법에 따라 객체를 교체하는 역할을 한다. Insert Object() 함수는 버퍼에 새로운 객체를 저장하고 객체가 버퍼에 저장되었다는 정보를 SLMtree에 추가한다.

3.4 성능 분석

객체 버퍼 아키텍쳐(architecture)[12]에서의 객체 참조 비용은 기존에 알려진 다음의 식 (1)을 통해 계산될 수 있다.

(1) $C=(O_{hit}\times C_{Ohit})+(O_{miss}\times C_{Omiss})$

(단, $O_{miss}=1-O_{hit}$)

O_{hit} : 객체가 버퍼에 있을 확률
C_{Ohit} : 버퍼의 객체를 참조하기 위한 비용
O_{miss} : 객체가 버퍼에 없을 확률
C_{Omiss} : 버퍼에 없는 객체를 참조하기 위한 비용

C_{Omiss}의 값은 디스크 I/O 비용과 클라이언트와 서버 간의 상호 작업을 위한 비용을 포함하게 되므로 직접 메모리를 참조하는 C_{Ohit}값보다 매우 큰 값을 갖게 된다. 따라서 전체적인 비용을 줄이기 위해서는 O_{hit}를 높여야 한다.

일반적으로 SDBMS에는 공간적 근접성이 존재하므로 객체들을 공간적 근접성 영역에 속하는 객체들의 집합과 공간적 근접성 이외의 영역에 속하는 객체들의 집합으로 나누어 볼 수 있다. 이와 같은 경우에 O_{miss}와 O_{hit}는 (2)의 식과 같이 표현 될 수 있다.

(2) $\begin{cases} O_{miss}=1-O_{hit} \\ O_{hit}=\sum_{i=1}^{s}(P_{buff}(O_i)\times\frac{X_{O_i}}{X})\times\sum_{m=s+1}^{t}(P_{buff}(O_m)\times\frac{X_{O_m}}{X}) \end{cases}$

t : 전체 객체의 수
$O_1\sim O_s$: 공간적 근접성 영역에 속하는 객체
$O_{s+1}\sim O_t$: 공간적 근접성 영역에 속하는 객체
$P_{buff}(O_k)$: 객체 O_k가 버퍼에 있을 확률값
X : 전체 객체 참조 횟수
X_{O_k} : 객체 O_k가 참조되는 횟수

(2)의 식을 통하여 O_{hit}는 공간적 근접성 영역에 속하는 객체들에 대한 객체 적중률과 공간적 근접성 영역에 속하지 않는 객체들에 대한 객체 적중률의 합임을 알 수 있다.

일반적으로 공간적 근접성이 나타나는 경우는 공간적 근접성 영역 내에 속하는 객체가 참조되는 횟수가 상대적으로 매우 높게 된다. 넓은 영역의 데이타가 저장된 대용량 공간 데이타베이스라면 그 차이는 더욱 더 커질 것이다. 따라서 전체적인 객체 적중률을 높이기 위해서는 (2)식에서 확인할 수 있듯이 공간적 근접성 영역에 속하는 객체들이 버퍼에 존재할 가능성을 높이는 것이 중요하다.

이 논문에서 제안하고 있는 SLAM은 공간적 근접성 영역 안에 속하는 객체들이 더 오래 버퍼에 저장되도록 한다. 따라서 (2)의 식에서 공간적 근접성 영역에 대한 객체 적중률 계산의 $P_{buff}(O_i)$를 높이는 효과를 얻게 되며, 결과적으로 버퍼에서의 객체 적중률이 커지게 된다.

그러나 SLAM을 사용하게 되면 버퍼 교체를 위해서 버퍼에서 공간적 근접성 영역 밖에 존재하는 객체를 큐에서 찾는 작업이 필요하기 때문에 부가적인 비용 ε

만큼이 더 증가하게 된다. 따라서 객체 적중률을 향상시켜서 얻는 비용 절감이 δ인 경우에 $\varepsilon < \delta$이 된다면 전체 시스템의 성능이 향상될 것이다. 4장에서의 실험은 실제 환경에서 $\varepsilon < \delta$이 되어 시스템 성능이 향상된다는 것을 보여준다.

4. 수정된 SLAM 버퍼관리기법

그림 10과 같이 지도에서 공간적 근접성 영역이 오른쪽 상단으로 잡혀 있을 때 대부분의 객체 참조는 오른쪽 상단에서 이루어지지만 가끔 공간적 근접성 영역과 많이 떨어져 있는 객체를 참조할 경우가 발생한다. 이때 SLM-tree는 공간적 근접성 영역을 추정하기 위해서 멀리 떨어져 있는 객체도 포함하는 MBR을 생성할 수 밖에 없게 된다. 이렇게 공간적 근접성 영역이 크게 추정되면 클라이언트 버퍼에서 추정된 공간적 근접성 영역에 속하지 않는 객체를 찾기가 힘들어 진다. 따라서 버퍼의 객체들을 교체해야 할 때 버퍼에서 끝까지 탐색해야 하는 상황이 자주 발생한다.

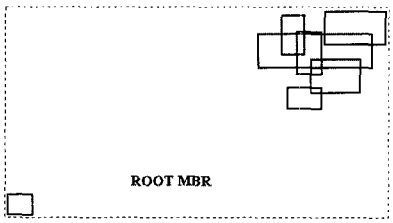


그림 10 M-LRU의 교체 대상 선정

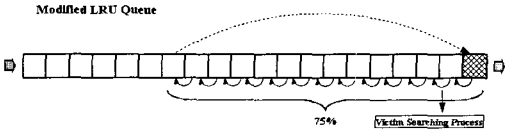


그림 11 수정된 SLAM 버퍼관리기법의 객체 탐색

위와 같은 문제를 해결하기 위하여 M-LRU의 기능을 수정하였다. 그림 11의 수정된 SLAM 버퍼관리기법은 M-LRU에서 공간적 근접성 영역에 속하지 않는 객체를 찾을 때 버퍼의 뒤에서 버퍼 전체의 x%를 탐색하여 공간적 근접성 영역 밖에 속하는 객체가 발견되지 않는다면 현재 공간적 근접성 영역이 너무 크게 추정되어 남은 영역에서도 찾기 힘들 것으로 간주하고 탐색을 중지하고 LRU와 같은 정책으로 교체대상을 선정한다. 탐색과정을 분석하여 불필요한 탐색시간을 절약하자는 것이 수정된 SLAM 버퍼관리기법의 기본 생각이다. 그러나 실험을 해본 결과 원래의 SLAM 보다 개선된 성능은 보여주지 못하였다. 변수 x의 값을 여러가지로 변화시키면서 실험하였는데 75%에서 상대적으로 가장 좋은 성능을 보여주었으므로 실험에서는 75%로 고정하여 실험하였다.

5. 실험과 분석

본 절은 제안하고 있는 SLAM 버퍼관리기법의 성능 실험을 다루고 있다. 시스템의 성능은 크게 두 가지로 측정하고 있다. 그 첫번째는 적중률(hit ratio)이다. 적중률은 전체 객체 요청 횟수 중에 클라이언트 버퍼에서 객체를 찾아 내는 비율을 나타내는 것으로 얼마나 효율적으로 객체 교체를 했는가를 나타내는 척도가 된다. 따라서 본 논문은 다음과 같이 기존에 알려진 적중률을 조사하여, 성능평가의 기준으로 삼는다.

$$HitRatio = \frac{N(R_{hit})}{N(R_{tot})}$$

$N(R_{hit})$: 디스크에 접근하지 않는 참조의 수
$N(R_{tot})$: 모든 참조의 수

두 번째는 일정 개수의 객체들에 대한 요청에 대하여 시스템이 얼마나 빠르게 요청한 객체들을 제공할 수 있는 가를 나타내는 응답 시간(response time)이다. 본 논문에서 제안하고 있는 SLAM 버퍼관리기법은 교체 대상을 선정하는 과정의 오버헤드(overhead)가 있기 때문에 응답 시간도 성능을 평가하는 중요한 척도가 된다.

실험결과는 실제 성능을 나타내는 그래프 또는 아래의 식과 같이 표현되는 기존의 방법에 대한 상대적인 성능을 나타내는 그래프로 보여준다.

$$P_{HitRatio} = \frac{HR_{SLAM}}{HR_{LRU}}$$

$P_{HitRatio}$: 상대적인 적중 이득률
HR_{SLAM} : SLAM의 적중률
HR_{LRU} : LRU의 적중률

$$P_{response} = \frac{RT_{SLAM}}{RT_{LRU}}$$

$P_{response}$: 성능향상률
RT_{SLAM} : SLAM의 응답시간
RT_{LRU} : LRU의 응답시간

5.1 실험 환경

5.1.1 실험 데이타

실험에서는 합성(synthetic) 데이타와 실제 지도 데이타, 두 가지 종류의 데이타를 사용하였다. 먼저 합성 데이타는 인위적으로 만든 공간 객체들이다. 표 2는 실험에 사용된 데이타들의 속성을 보여주고 있다.

표 2에서 볼 수 있듯이 전체 데이타 중 약 3%가 참조되는 빈도가 높은 공간적 근접성 영역 안의 데이타이다.

표 2 합성 데이타 속성

데이터베이스크기	100 MB
평균객체크기	1KB (최소 100B, 최대 1.9KB)
평균객체MBR크기	10 x 10 (최소: 5, 최대: 20)
객체의 수	100,000
지도의 크기	10,000 x 10,000
공간적근접성영역 설정크기	1,700 x 1,700
객체의 분포	균등분포 (uniform distribution)

실제 데이타는 공간 데이타베이스 연구에서 많이 사용되는 Tiger/lines[13]을 이용하였다. 그림 12는 미국의 캘리포니아 주 지역의 도로 정보를 나타낸 것으로 본 논문에서 실험한 데이타의 크기와 분포를 보여준다. 전체 978,428 x 1,115,916 크기의 지도에서 50,000 x 80,000의 영역을 공간적 근접성 영역으로 설정하였으며 이는 전체 객체 101,144개의 2%를 포함하는 영역이다. 전체 데이타베이스 크기는 100MB이다.

시크 타임(Seek time)은 9msec으로 가정하였고, 평균 래이턴시 타임(latency time)은 6msec으로 가정하였으며 4KB를 전송하는 데 소요되는 전송 시간(transfer time)은 1msec으로 가정하였다[14].

그림 12 Tiger/lines 데이터

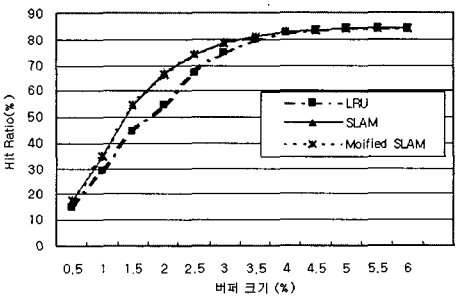
5.1.2 실험 방법

먼저 합성 데이타를 이용한 실험은 5.2절부터 5.4절까지 기존의 LRU 버퍼관리기법, SLAM 버퍼관리기법, 수정된 SLAM 버퍼관리기법의 세가지 방법들에 대해서 세가지 매개 변수(parameter)들의 값을 다르게 설정하면서 성능의 변화를 분석하였다. 첫번째로 전체 데이타베이스의 크기에 대한 버퍼의 상대적인 크기에 따른 성능 변화를 실험하였다. 두 번째로 공간적 근접성 영역을 참조하는 빈도의 변화에 따른 성능 변화를 실험하였다. 이 값은 공간적 근접성이 얼마나 강하게 나타나는 지를 나타내는 값이다. 마지막으로 SLM-tree에서 공간적 근접성 영역을 추측할 때 고려하는 객체들의 수에 따라 나타나는 성능 차이를 조사하였다. 실제 데이타에 대한 실험도 위와 같은 방식으로 실행하였으나 유사한 결과를 보이기 때문에 간략히 5.5절에서 설명하였다.

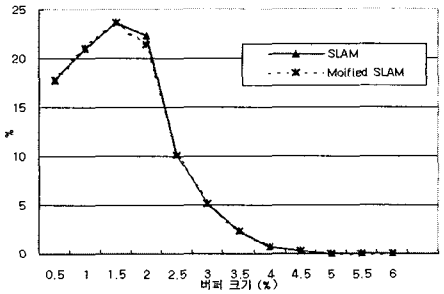
5.2절, 5.4절, 5.5절의 실험에서는 85% 정도의 객체 요청이 공간적 근접성 영역에 집중되도록 설정한 후, 각 20만번의 객체를 요청한 실험을 통하여 적중률과 응답 시간을 측정하였다. 또한, 5.3절의 실험에서는 공간적 근접성 영역의 객체들에 대한 객체 요청 비율을 다양하게 변화시키면서, 각 20만번의 객체 요청을 통해 적중률과 응답시간을 측정하였다.

5.2 버퍼의 상대적인 크기에 따른 성능

이 실험은 전체 데이타베이스에 대한 상대적인 클라이언트의 버퍼 크기가 데이타 적중률과 응답 시간에 어떤 영향을 미치는 지를 알아보기 위한 실험이다. SLM-tree가 공간적 근접성 측정을 위해 고려하는 객체의 수를 20으로 설정하였다.



(A) 적중률



(B) 상대적 적중이득률

그림 13 버퍼 크기에 따른 적중률과 상대적 적중이득률

그림 13은 버퍼 크기에 따른 적중률의 변화를 나타낸 것이다. 그림 13-(B)에서 보여지듯이 버퍼 크기가 데이타베이스 사이즈의 0.5%정도일 때는 모두 비슷한 성능을 보이며 버퍼 크기가 1.5% 일 때 SLAM의 상대적인 적중률이 최고조에 이르러 약 25%정도의 버퍼 적중률 성능 향상을 가져온다. 그러나 버퍼 크기가 더 커지게 되면 LRU의 적중률이 점차로 증가하여 기존의 LRU 버퍼관리기법과 비슷한 수준의 적중률을 보여 주었다.

버퍼 크기가 너무 작을 때 좋은 성능을 보여 주지 못하는 이유는 버퍼 크기가 너무 작아서 공간적 근접성에 속하는 객체들을 충분히 저장할 수 없게 되기 때문에 쓰래싱(thrashing) 현상이 빈번히 발생하기 때문이다. 또한 버퍼가 커졌을 때 되었을 때는 버퍼가 충분하여 공간적 근접성 영역 안에 들어 있는 모든 객체를 저장할 수 있기 때문에 LRU의 성능이 충분히 향상된다. 수정된 SLAM의 경우는 M-LRU의 끝까지 탐색을 하지 않기 때문에 적중률이 SLAM에 비해 저하되었다.

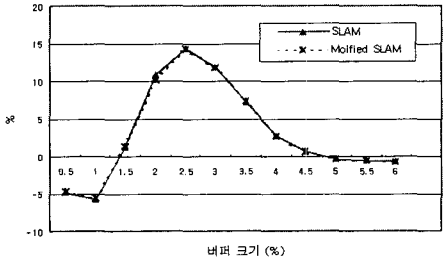


그림 14 버퍼 크기에 따른 응답시간 성능향상률

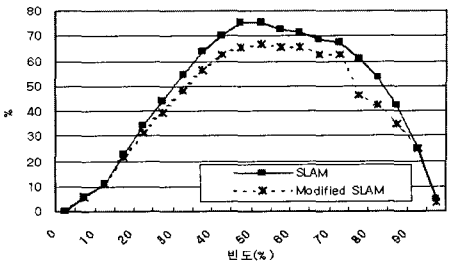


그림 15 빈도에 따른 상대적 적중이득률

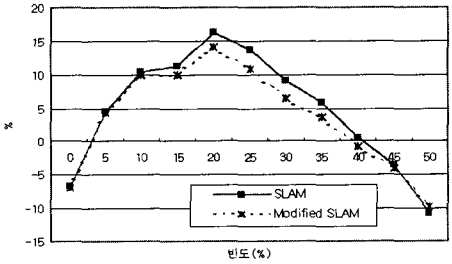


그림 16 빈도에 따른 응답시간 성능향상률

그림 14는 버퍼의 크기에 따라서 각각의 기법들이 어느 정도의 응답시간을 보여주는가를 나타내는 그래프이다. 버퍼가 데이타베이스의 2.5% 정도일 때 최고 14% 정도 성능향상을 보임을 알 수 있다. 그러나 버퍼의 크기가 1.5% 이하로 작을 때는 오히려 LRU의 성능이 좋게 나타나는데, 이것은 SLAM의 경우 버퍼가 작아서 버퍼내의 거의 모든 객체가 공간적 근접성안에 속하는 객체들이고, 따라서 불필요한 탐색을 하기 때문이다. 버퍼의 크기가 4%가 넘을 때는 적중률의 향상이 없기 때문에 성능향상이 나타나지 않는다. 수정된 SLAM의 경우 전체적으로 비슷한 수준의 응답 시간을 보여준다.

5.3 공간적 근접성 영역 참조 빈도에 따른 성능

이 실험은 공간적 근접성 영역으로 설정된 영역 안에 있는 객체에 대한 참조 빈도를 달리하면서 그 변화가 적중률 및 성능에 어떤 영향을 주는 지를 알아보기 위한 것이다. 버퍼의 사이즈는 전체 데이타베이스 사이즈의 2%로 설정을 하였으며, 공간적 근접성 영역을 추정하기 위해 이용하는 객체의 수는 20으로 설정하였다.

빈도에 따른 상대적 적중 이득률을 보여주는 그림 15를 보면 외부 객체 참조 비율이 35% 내외인 경우에 SLAM이 70%정도 성능향상을 보여주는데 이는 공간적 근접성 영역 외부 영역에 대한 참조 빈도가 높아질수록 외부 객체들이 버퍼의 많은 부분을 차지하게 되고 SLAM 버퍼관리기법은 쉽게 이들 외부 객체들을 교체 대상으로 찾을 수 있기 때문이다. 그러나 외부 객체 참조 비율이 35% 이상으로 높아지면 현재의 공간적 근접성 영역을 추정하는 과정에서 공간적 근접성 영역 외부에 존재하는 객체의 MBR에 의해 영향을 많이 받게 되기 때문에 LRU와 비슷한 성능을 보여준다.

그림 16은 LRU에 대한 상대적인 성능을 보여주고 있다.빈도가 40% 정도가 될 때까지는 SLAM 버퍼관리기법이 LRU 버퍼관리기법보다 상대적으로 짧은 응답시간을 보여 주어 최고 17% 정도까지 성능을 향상시킴을 보여준다. 빈도가 적절한 수준의 공간적 근접성으로 유지되면 SLAM이 좋은 성능을 보여준다. 그러나 빈도가 40%가 넘으면 앞에서 언급하였듯이 부정확한 공간적 근접성 추정으로 SLAM의 성능이 저하된다.

SLM-tree가 고려하는 객체 수에 따른 성능

이 실험은 SLM-tree가 고려하는 객체의 수가 성능에 어떤 영향을 주는 지를 조사하기 위한 실험이다. 공간적 근접성 영역의 객체를 참조하는 빈도를 85%로 하고 이외의 영역을 15% 참조하도록 하였으며 버퍼의 크기는 2%로 설정하였다.

그림 17은 SLAM 알고리즘에서 SLM-tree에서 고려하는 객체의 수가 적중률에 어떤 영향을 미치는 지를 실험한 결과이다. SLAM의 경우에 고려하는 객체의 수가 많아지면 처음에는 성능이 향상되지만 어느 수준 이상으로 많아지면 점차로 성능이 저하되는 것을 볼 수

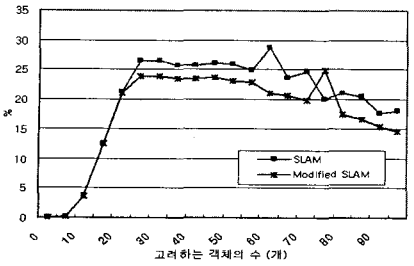


그림 17 고려하는 객체수에 따른 상대적 적중이득률

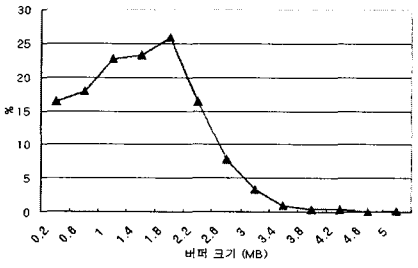


그림 18 버퍼크기에 따른 상대적 적중이득률(Tiger/lines)

있다. 고려하는 객체의 수가 적을 때는 추정된 공간적 근접성 영역이 정확하지 않기 때문에 성능이 나쁘고 고려하는 객체의 수가 25정도로 늘 때까지 계속 성능이 개선되어 25%까지 성능향상을 보인다. 그러나 고려하는 객체의 수가 50을 넘어서 계속 늘어나면 공간적 근접성 영역이 너무 크게 추정되는 상황이 자주 발생하여 점차로 성능이 저하되는 것을 발견할 수 있다.

응답시간의 상대적 성능을 보여주는 그림 18은 고려하는 객체의 수가 약 25개정도가 될 때까지 성능이 향상되는 것을 보여준다. 이는 앞에서 적중률 실험에서도 설명했듯이 고려하는 객체의 수가 적절한 수가 되어야 보다 신뢰할 수 있는 공간적 근접성 영역을 추정할 수 있기 때문이다. 또한 고려하는 객체의 수가 25개 이상이 될 때는 적중률의 저하에 따라 성능이 저하된다.

5.5 실제 데이타를 이용한 실험

이 장에서는 Tiger/lines[13] 데이타를 이용하여 실험한 결과에 대하여 설명한다. 이 데이타는 실제 지리 정보이기 때문에 특정지역에 객체들이 밀집되는 등 다양한 객체들의 비균등 분포가 나타난다. 앞에서 설명한 합성 데이타에 대해 실행한 실험들과 같이 네 가지 파라미터 값들을 변경시켜 가면서 실험을 하였다. 실험 결과들이 합성 데이타에 대해 실행한 실험 결과와 거의 유사하므로 여기에서는 버퍼의 크기에 따라서 적중률과 응답시간이 어떻게 변하는 지에 대한 결과만을 보여준다.

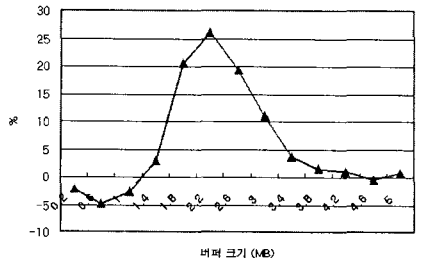


그림 19 버퍼크기에 따른 성능향상률 (Tiger/lines)

그림 19는 버퍼의 크기 변화에 따라 적중률이 어떻게 변하는 지를 보여주고 있으며, 그림 20는 응답시간이 어떻게 달라지는 지를 보여 주고 있다. 두 그래프는 각각 일반적인 LRU에 대한 상대적인 적중률과 응답시간 성능을 보여주고 있다. 적중률과 응답시간 모두 LRU 방법과 비교하여 최고 26% 정도 성능 향상을 나타냄을 확인할 수 있다. 따라서 앞에서의 합성 데이타에 대한 실험이 실세계 데이타 패턴과 많이 다르지 않다는 것을 확인할 수 있으며, SLAM 버퍼관리 방법이 다양한 패턴 및 분포를 나타내는 데이타에서도 효율적으로 이용될 수 있다는 것을 확인할 수 있다.

6. 결 론

본 논문에서는 SDBMS 환경에서 디스크 I/O 횟수를 줄여 시스템 성능을 향상시키는 버퍼관리기법에 관하여 다루고 있다.

기존의 버퍼관리기법들은 일반적으로 대부분의 DBMS에서 나타나는 시간적 근접성만을 고려하고 있기 때문에 SDBMS에만 존재하는 공간적 근접성이라는 특성이 고려되지 않았으며 SDBMS 환경에서 좋은 성능을 보여 주지 못하였다. 하지만 본 논문에서 제안하는 SLAM 버퍼관리기법은 SDBMS 환경에서 버퍼를 관리할 때 시간적 근접성과 공간적 근접성을 같이 고려하여 버퍼에서 적절한 교체 대상을 선정한다. SLAM 버퍼관리기법은 공간적 근접성과 시간적 근접성을 고려하기 위하여 SLM-tree와 M-LRU의 두 가지 구조체를 이용한다.

LRU 버퍼관리기법과 성능을 비교한 실험을 통해 SLAM 버퍼관리기법이 적중률 측면에서 많은 향상을 보이며, 응답 시간 측면에서도 적지 않은 성능 향상을 보여 줌을 확인하였다.

참 고 문 헌

[1] 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.
[2] Brinkhoff, T., Kriegel, H.-P., Seeger, B., "Parallel processing of spatial joins using R-trees," Data Engineering, Proceedings of the Twelfth International Conference on, pp.258-265, 1996.
[3] T. Brinkhoff, H. Horn, H. Kriegel, R. Schneider "A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems," Symposium on Large Spatial Databases, pp.357-376, 1993.
[4] E. Bertino, L. Martino "Object Oriented Database Management Systems: Concepts and Issues" IEEE Computer, Vol. 24, No.4, pp.31-47, April 1991.
[5] R. Breitl et al., "The Gemstone Data Management System," in Object-Oriented Concepts, Databases and Applications, W.Kim and F. Lochovsky eds., Addison-Wesley Readings, Mass., pp.283-308, 1989.
[6] K. Wilkinson, P. Lyngbaek and W. Hassan, "The Iris Architecture and Implementation," IEEE Trans. Knowledge and Data Eng., Vol.2, No. 1, pp.91-108, 1990.
[7] W. Effelsberg, T. Harder "Principles of Database Buffer Management," ACM Transactions on Database Systems, Vol. 9, No. 4, pp.560-595, December 1984.
[8] H. Chou and D.J.DeWitt "An Evaluation of Buffer Management Strategies for Relational Database Systems," Proceedings of the 11th VLDB Conference, pp.127-141, August 1985.
[9] E. J. O'Neil, P. E. O'Neil, G. Weikum "The LRU-K Page Replacement Algorithm For Database Disk Buffering," ACM SIGMOD Conference, pp. 297-306, May, 1993.
[10] T. Johnson and D. Shasha "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm," Proceedings of the 20th VLDB Conference, pp.439-450, 1994.
[11] R. Guting "An Introduction to Spatial Database Systems," the VLDB Journal, Vol. 3, No. 4, pp. 357-399, October 1994.
[12] D. J. DeWitt and D. Mater "A Study of Three Alternative Workstation-Server Architectures for Object-Oriented Database Systems," Proceedings of the 16th VLDB Conference, pp.107-121, 1990.
[13] Tiger/lines precensus files: 1994 technical documentation, Technical report, *U.S. Breau of Census*, 1994.
[14] T. Brinkhoff and H. Kriegel "The Impact of Global Clustering on Spatial Database System," Proceedings of the 20th VLDB Conference, pp.168-179, 1994.

안 재 용
1999년 고려대학교 컴퓨터학과(학사). 2001년 한국과학기술원 전산학과(석사). 2001년 ~ 현재 한국과학기술원 박사과정 재학중. 관심분야는 공간데이타베이스, GIS, 버퍼관리, XML

민 준 기
1995년 숭실대학교 전자계산학과(학사). 1997년 한국과학기술원 전사학과(석사). 1998년 한국과학기술원 위촉연구원. 2002년 8월 한국과학기술원 전산학전공(박사). 관심분야는 XML, GIS, 데이타마이닝

정 진 완
정보과학회논문지 : 데이타베이스
제 29 권 제 1 호 참조