다중 존 디스크 환경에서 다차원 인덱스 구조의 효율적 저장 기법

(Efficient Storage Techniques for Multidimensional Index Structures in Multi-Zoned Disk Environments)

유 병 구 † 김 선 호 †† 장 재 영 †††
(Byung-gu Yu) (Seon-ho Kim) (Jae-young Chang)

요 약 대용량의 다차원 데이타를 다루는 데이타베이스 응용분야에서는 접근 방법 및 기반 디스크 시스템이 전반적인 성능에 중요한 영향을 미친다. 현재 생산되고 있는 많은 디스크들은 다중의 물리적 존을 갖도록 설계되고 있다. 그러나 기존의 접근 방법에 대한 연구는 단순한 가정의 전통적인 디스크 모델에 기반을 두고 진행되어 왔고, 다중 존 디스크를 고려한 접근 방법에 대한 연구는 현재까지 거의 이루어 지지 않고 있다. 본 논문에서는 다중 존 디스크 환경에서 실질적인 데이타 전송률을 향상시키기 위해, 정적 및 동적 환경 모두를 고려한 다차원 인덱스 구조의 디스크 저장 기법을 제안한다. 이를 위해 다차원 인덱스 구조를 다중 존 디스크에 효과적으로 배치하는 알고리즘을 제시하고, 범위 질의에 대해 지역화된 질의 처리 기법을 제안한다. 또한 실험을 통하여 본 논문이 제안한 기술이 질의 성능을 획기적으로 향상시킨다는 것을 증명한다.

키워드 : 데이타베이스, 다중 존 디스크, 접근 방법, 다차원 인덱스, 질의 처리

Abstract The performance of database applications with large sets of multidimensional data depends on the performance of its access methods and the underlying disk system. In modeling the disk system, even though modern disks are manufactured with multiple physical zones, conventional access methods have been developed based on a traditional disk model with many simplifying assumptions. Thus, there is a marked lack of investigation on how to enhance the performance of access methods given a zoned disk model. The paper proposes novel zoning techniques that can be applied to any multidimensional access methods, both static and dynamic, enhancing the effective data transfer rate of underlying disk system by fully utilizing its zone characteristics. Our zoning techniques include data placement algorithms for multidimensional index structures and accompanying localized query processing algorithms for range queries. The experimental results show that our zoning techniques significantly improve the query performance.

Key words : database, multi-zoned disk, access method, multidimensional index, query processing

1. 서 론

대용량 데이타를 처리해야하는 데이타베이스 응용 시스템들이 늘어남에 따라 이 분야의 질의 성능을 향상시키기 위한 연구가 활발히 이루어지고 있다. 특히 대용량 데이타 환경에서의 질의 성능은 데이타에 대한 접근 방법(access method)과 기반 디스크 시스템에 많은 영향을 받는다. 최근 들어 인접한 여러 개의 디스크 실린더들을 존(zone) 단위로 묶어 전체 디스크 영역을 여러 개의 존으로 관리하는 기술이 사용되고 있다[1,2]. 디스크는 물리적 특성으로 인해 디스크 중심에서 바깥쪽으로 갈수록 트랙(track)의 길이가 더 길어짐에 따라 이론적으로 바깥쪽의 트랙에 더 많은 양의 데이타를 저장할 수 있다. 따라서 디스크를 여러 개의 물리적 존으로 나눈 후에, 존마다 탐색 시간(seek time)과 데이타 전송률(transfer rate)을 다양화함으로써, 디스크의 성능 및 저장 공간을 극대화 할 수 있다. 그러나 현재까지 이러한

· 본 연구는 2007년도 한성대학교 교내연구비 지원과제임
† 정 회 원 : National University Computer Science and Information Systems Department 교수
byu@nu.edu
†† 정 회 원 : University of Denver Computer Science Department 교수
seonkim@cs.du.edu
††† 종신회원 : 한성대학교 컴퓨터공학과 교수
jychang@hansung.ac.kr
논문접수 : 2006년 9월 4일
심사완료 : 2007년 4월 16일

다중 존(multi-zone)으로 분할된 디스크 모델을 적용하여 데이타 접근 방법(예를 들어 인덱스 저장 및 검색)을 최적화하기 위한 연구는 거의 이루어지지 않고 모든 트랙이 동일한 탐색 시간과 데이타 전송률을 갖는 전통적인 디스크 모델에 기반을 둔 연구만이 대부분을 차지하고 있다.

본 논문에서는 다중 존 디스크 환경에서 다차원 인덱스와 같은 트리 형태를 갖는 접근 방법의 성능을 향상시키기 위한 인덱스 저장 및 질의 처리 기술을 제안한다. 다중 존 디스크 환경에서는 다음과 같은 조건이 만족될 경우 접근 방법의 성능 향상을 기대할 수 있으며, 본 논문에서는 이에 대한 구체적인 방안을 제시한다.

조건 1. 인덱스 구조에서 자주 접근되는 페이지들을 보다 빠른 존에 저장한다.

조건 2. 실행중인 모든 쓰레드(thread)는 한 순간에 오직 하나의 디스크 존만을 접근하며, 각 쓰레드는 한 존의 모든 필요한 작업이 완료된 후에 다른 존으로 이동하여 접근한다.

우선 조건 1에 대해서 트리 형태의 인덱스 구조와 임의의 질의를 가정하자. 이 경우 인덱스 구조에서 각 페이지 - 트리의 각 노드(node) - 의 상대적 접근 빈도는 (access frequency) 트리에서 해당 페이지의 레벨(level)과 비례한다고 가정할 수 있다. 예를 들어 인덱스의 모든 검색은 루트 페이지(root page)부터 시작되므로 루트 페이지가 가장 높은 접근 빈도를 갖는다. 같은 이유로 단말 페이지(leaf page)들이 가장 낮은 접근 빈도를 갖는다고 가정할 수 있다. 따라서 각 페이지들이 저장될 디스크의 존은 해당 페이지의 레벨로 결정할 수 있다. 이러한 가정은 모든 트리 형태의 인덱스 구조에 적용될 수 있는 가장 간단하고도 직관적인 접근 방법이며, 버퍼 시스템에서 트리 형태 인덱스의 상위 구조를 버퍼에 상주시키는 기법의 기본적인 가정과도 일치하는 것이다[3]. 그러나 이 기법은 인덱스 내의 데이타 분포를 고려하지 않았을 뿐만 아니라, 각 질의에 대해서 모든 존을 방문해야만 하는 문제점을 안고 있다. 이러한 접근 방법은 기본적으로 동일 레벨에 있는 모든 페이지들은 동일한 접근 확률 및 빈도를 갖는다는 가정을 배경으로 한다. 하지만 실제 환경에서 이러한 경우는 거의 발생되지 않는다.

일반적으로 임의의 범위 질의(range query)에 대해서, 큰 데이타 공간을 갖는 페이지일수록 그렇지 못한 페이지에 비해 접근될 확률도 높아진다. 따라서 조상-자손(ancestor-descendant) 관계가 아닌 두 페이지에 대해서, 레벨이 낮은 페이지가 높은 페이지에 비해 더 큰 데이타 공간을 가짐으로써 접근 빈도가 오히려 높은 경우도 얼마든지 존재할 수 있다. 이와 더불어, 질의의 범위가 더 많은 데이타 객체를 보유한 페이지 쪽에 집중될수록 해당 페이지는 그렇지 못한 페이지에 비해 높은 접근 확률을 갖게 된다. 즉, 페이지의 접근 빈도는 각 페이지가 보유한 데이타 객체의 수로도 결정될 수 있다.

따라서 각 페이지의 접근 빈도는 각 페이지에 저장된 실제 데이타 공간의 크기에 의해 결정될 수 있으며, 추가적으로 해당 페이지가 보유하는 데이타 객체의 수로도 결정될 수 있다. 결론적으로 각 페이지에 대한 디스크 존의 배정은 각 페이지의 확률적 접근 빈도에 의해 결정되는 것이 가장 이상적이다. 본 논문에서는 각 페이지의 접근 빈도를 고려하여 질의 성능을 극대화 하기위해 인덱스 페이지를 다중 존 디스크에 효율적으로 저장 관리하는 기법을 제안한다.

다음으로 조건 2를 만족할 경우 존 단위로 질의 처리를 지역화(localization)할 수 있다. 즉, 각 존 내에서는 페이지들이 클러스터링되는 효과를 나타냄으로 디스크에서 탐색 시간을 최소화 하여 질의 처리 성능을 획기적으로 향상 시킬 수 있다. 이 조건을 만족시키기 위한 가장 단순한 방법은 잠금(lock) 기법을 이용하는 것이다. 하나 이상의 쓰레드가 특정 존을 접근중일 경우 동일 디스크에 있는 모든 다른 존을 잠금으로써 접근을 막는 것이다. 그러나 이러한 접근 방법은 각 쓰레드가 다른 존으로 이동할 때 잠금이 해제되기를 기다려야하는 시간 때문에 오히려 성능에는 악영향을 가져올 수 있다. 따라서 잠금 기법을 사용하지 않으면서 조건 2를 만족시키기 위한 새로운 기법이 필요하다. 본 논문에서는 질의 처리 과정에서 한 존 내에서 접근해야하는 페이지를 효율적으로 탐색하여 질의 처리 성능을 향상시키는 기법을 제안한다.

본 논문에서 제안한 기법의 타당성 및 실용성을 검증하기 위해 다양한 환경에서의 실험 결과를 제시한다. 실험은 R^*-트리[4]를 이용하여 실시하였고 다중 존 디스크와 기존 디스크 환경에 대해서 성능을 비교 분석하였다. 본 논문에서는 문제를 단순화하기 위해 우선 하나의 디스크만을 가정하였다. 하지만 이 기법은 RAID와 같은 다중 디스크 환경에 대해서도 쉽게 확장될 수 있다.

본 논문에서 제시된 일부 기법들은 [5]와 [6]에서 특정 환경에 대한 부분적인 해결방안으로 소개되었다. [5]에서는 정적 인덱스 환경에서의 저장 기법을 제안하였고 [6]에서는 삽입과 삭제가 수시로 발생하는 동적 인덱스 환경을 고려하였다. 본 논문에서는 이러한 내용들을 하나의 이론적 프레임워크로 통합 확장하였다. 특히 실제 환경에서의 효과를 검증하기 위해 다양한 환경을 가정한 대용량의 모의 데이타에 대해 전반적으로 실험을 새롭게 실시하였다. 또한 통신회사의 운용 과정 중에 얻어진 대용량의 실제 데이타에 대한 실험도 추가하여 제

안된 기법의 실용성 및 우수성을 검증하였다.

본 논문의 구성은 다음과 같다. 우선 2장에서는 다중 존 디스크 모델에 대한 간단한 소개와 트리 형태의 인덱스 구조를 다중 존에 구성하는 기법을 소개한다. 3장에서는 존 단위로 지역화된(localized) 질의 처리 방법에 대해 제안하고 4장에서는 실험 결과를 제시한다. 마지막으로 5장에서는 결론 및 향후 연구 과제에 대해 논한다.

2. 다중 존 구조에서의 인덱스 저장 기법

2.1 다중 존 디스크

최근 들어 자기 디스크 드라이브에서의 가장 중요한 물리적 특징 중 하나가 바로 다중 존 기술이다. 다중 존의 기본 개념은 원형으로 구성된 자기 디스크의 물리적 특성에 기인한다. 각 트랙은 디스크의 중심으로부터 외부 방향으로 갈수록 더 길어진다. 따라서 디스크 상에서 동일한 저장밀도를 갖는다는 가정 하에 내부보다는 외부 트랙에 더 많은 데이타를 저장할 수 있다. 이러한 특성을 기반으로 디스크의 모든 트랙을 여러 개의 배타적 구역으로 분할한다. 즉, 하나의 존을 연속적인 디스크 트랙의 집합으로 구성한다[1,2]. 이때 한 존 내의 각 트랙은 동일한 저장용량을 갖도록 한다. 즉 같은 존 내의 각 트랙당 섹터 수는 동일하다. 따라서 중심으로 부터 외부에 위치한 존은 내부의 존보다 보다 많은 섹터수를 갖게 설계할 수 있으므로 외부 존에 더 많은 양의 데이타를 저장할 수 있다. 결과적으로 전체 디스크의 저장용량이 획기적으로 증가하게 된다.

디스크 당 존의 수는 디스크 모델에 따라 다양하다. 예를 들어, Seagate Cheetah X15는 9개의 존으로 구성되며 Seagate Barracuda 7200.7은 15개의 존을 갖는다. 한 디스크 내의 서로 다른 존은 서로 다른 전송률을 갖는다. 그 이유는 우선 각 존마다 트랙의 저장용량이 다르지만 디스크의 회전 속도는 존에 관계없이 일정하기 때문이다. 따라서 디스크 외부 존의 트랙이 내부 존의 트랙보다 더 많은 섹터로 구성되지만 하나의 트랙을 읽는 속도는 위치에 관계없이 일정하므로 외부 존의 데이타 전송률이 더 높게 된다. 예를 들어 표 1은 존 별 저장용량과 데이타 전송률의 예를 보여준다. 이 표에서 보는 바와 같이 디스크 외부 존의 용량 및 데이타 전송률이 내부 존보다 높은 것을 알 수 있다.

표 1 Seagate Cheetah X 15의 존 구성

Zone #	Size (GB)	*Read Rate (MB/s)*
0	12	*57.5*
1	3.5	*55.4*
2	3.0	*54.7*
3	4.0	*52.7*
4	3.0	*50.6*
5	2.5	*48.1*
6	3.0	*45.6*
7	2.5	*43.6*
8	*2.5*	*41.9*

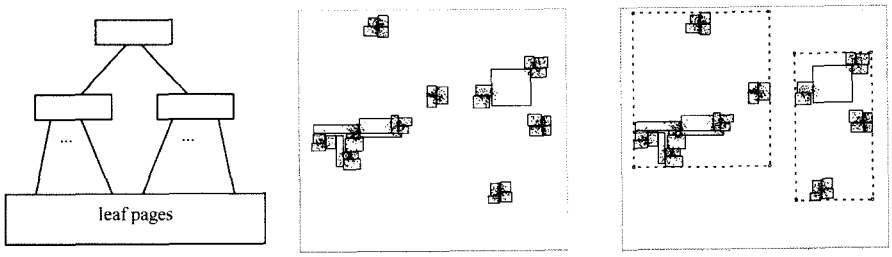
2.2 계층적 인덱스 구조

디스크와 같은 보조기억 장치에서 인덱스는 대부분 균형 트리(balanced tree) 형태의 계층적 구조를 갖는다. 트리의 각 페이지들은 자식 페이지들의 차지하는 데이타 영역을 모두 포함하는 영역으로 정의된다. 따라서 루트 페이지는 인덱스에 저장된 모든 데이타 객체의 위치 또는 영역을 포함하는 전체 영역을 나타낸다. 단말 페이지들에는 실제 데이타 객체들이나 이에 대한 간접 정보인 데이타 엔트리(data entry)들이 저장된다. 예를 들어 데이타 엔트리는 <coordinate, tuple_pointer>의 형태로 나타낼 수 있는데, 여기서 coordinate는 다차원 공간에서 데이타 객체의 위치 또는 영역을 나타내며, tuple_pointer는 실제 데이타 객체가 저장된 주소를 나타낸다.[1] 중간 페이지들에는 인덱스 엔트리의 집합으로 구성된다. 즉, 인덱스 엔트리는 <region_descriptor, child_pointer>로 구성되는데, region_descriptor는 child_pointer의 하위에 존재하는 모든 단말페이지에 있는 데이타 객체를 포함하는 최소의 영역을 나타낸다. 예를 들어 그림 1은 포인트 데이타에 대한 R*-트리의 인덱스 구조에 대한 예를 보여준다.

일반적으로 인덱스는 일괄 갱신(bulk update)과 동적 갱신(dynamic update)의 두 가지 방법으로 구축된다. 일괄 갱신은 미리 주어진 데이타 집합에 대해서 인덱스 구조를 하나의 알고리즘으로 단시간에 구축하는 것을 말하며, 동적 갱신은 개별 데이타의 생성 및 삭제에 따라 인덱스에 삽입, 삭제 연산을 실시간으로 실행하는 것을 의미한다.

동적 갱신은 주로 OLTP(On-Line Transaction Processing)와 같이 규모가 작은 다수의 트랜잭션들에 의해 데이타의 삽입 삭제가 빈번히 발생하는 환경에서 성능에 중요한 영향을 미친다. 반면에 일괄 갱신은 데이타 웨어하우스(data warehouse) 또는 OLAP과 같은 환경 즉, 대용량이면서 갱신이 거의 발생되지 않는 정적 환경에서 자주 사용된다. 인덱스 구축 및 갱신에 관련된 알고리즘은 동적 갱신뿐만 아니라 일괄 갱신의 경우에 대해서도 이미 이전에 많은 연구가 이루어졌다[7-10]. 하지만 이전의 어떠한 인덱스 관련 연구에서도 다중 존을 갖는 디스크 모델을 가정한 경우는 없었다.

1) Coordinate는 데이타 객체의 타입 즉, 포인트 데이타냐 혹은 영역 데이타냐에 따라 달라진다.



(a) (b) (c)

그림 1 계층적 인덱스 구조의 예: (a) 페이지로 구성된 인덱스 구조, (b) 단말 페이지의 영역, (c) 단말 페이지의 부모 페이지 영역(점선 사각형)

인덱스 구조를 다중 존에 저장하기 위해서는, 우선 디스크에 존재하는 모든 존들 중에서 인덱스를 저장하기 위한 존 집합을 결정해야한다. 예를 들어 극단적으로 디스크의 모든 존이 인덱스 저장을 위해 사용될 수도 있고, 극히 일부 존들만이 사용될 수도 있다. 이러한 존 집합의 선택은 존의 활용도나 저장할 인덱스 구조의 중요도 등 시스템 관리 정책에 따라 달라질 수 있으나 이 문제는 본 논문의 범위를 벗어나므로 더 이상 논하지 않는다. 따라서 본 논문에서는 인덱스를 저장할 주어진 존 집합이 이미 선택되어졌다는 가정 하에 다음의 두 조건을 만족하는 인덱스 저장 알고리즘을 제시한다.

1) 모든 선택된 존은 공평하게 활용된다(equally utilized).

2) 존 *zid*에 저장된 모든 인덱스 페이지 p에 대해서, *zid*보다 전송률이 느린 존 *zid'*에 저장된 어떠한 페이지 p'에 대해서도 p'의 접근 확률은 항상 p보다 크지 않다.

다중 존에서의 인덱스 저장 기술은 위에서 언급한 일괄 갱신과 동적 갱신 모두에 대해 고려될 수 있으며 다음 절에서 각각에 대한 알고리즘을 제시한다.

2.3 다중 존에서의 인덱스 저장 알고리즘

본 절에서는 우선 일괄 구축으로 생성되는 정적 인덱스 환경에서 페이지들을 배치하는 정적 인덱스 저장 알고리즘 - 이하 SMD-Zoning(Static Multi-Dimensional Zoning) - 을 소개하고, 다음으로 인덱스 페이지의 삽입과 삭제가 수시로 발생하는 동적 환경에서의 인덱스 저장 알고리즘 - 이하 DMD-Zoning(Dynamic Multi-Dimensional Zoning) - 을 소개한다.

이를 위해 주어진 인덱스 구조에서 페이지들의 접근 확률에 대한 정의가 필요한데 우선 다음의 함수를 가정하자. *Measure*(p)를 페이지 p의 *area* 또는 *margin*이라 하고, *Pop*(p)를 페이지 p의 자손이 되는 모든 단말 페이지에 포함된 데이타 객체들의 수라고 하자. 여기서 *area*나 *margin*은 해당 페이지가 나타내는 영역의 크기를 측정하는 방법으로 각각 다음과 같이 정의할 수 있다.

정의 1. 차원의 크기가 d인 다차원 인덱스 페이지 p에 대해서, p가 나타내는 공간의 각 차원 $i(1 \le i \le d)$의 범위가 l_i부터 h_i까지라면, p의 *area*는 $\prod_{i=1}^{d}(h_i - l_i)$로 정의한다.

정의 2. 차원의 크기가 d인 다차원 인덱스 페이지 p에 대해서, p가 나타내는 공간의 각 차원 $i(1 \le i \le d)$의 범위가 l_i부터 h_i까지라면, p의 *margin*은 $2^{d-1} \cdot \sum_{i=1}^{d}(h_i - l_i)$로 정의한다.

위의 정의에서 *area*는 페이지 p가 나타내는 영역의 크기를 넓이 또는 부피로 나타낸 것이고, *margin*은 영역의 크기를 각 차원의 길이를 모두 더한 수치로 나타낸 것이다. 예를 들어, 이차원의 경우 *margin*은 사각형 둘레의 길이를 의미한다. 다음으로 *MeasureU*와 *PopU*를 각각 루트 페이지의 *area*(또는 *margin*)와 트리에서 전체 데이타 객체의 총 수라 하자. 그러면 임의의 질의에 대해서 페이지 p가 접근될 확률(access probability)은 다음의 가중치를 이용한 수식으로 나타낼 수 있다.

$$RI(p) = W1 \times \frac{Measure(p)}{MeasureU} + W2 \times \frac{Pop(p)}{PopU} \qquad (1)$$

단, $0 \le W1 \le 1, 0 \le W2 \le 1$ 그리고 $W1 + W2 = 1$

이 식에서 $RI(p)$는 페이지 p의 상대 중요도(*RI*: Relative Importance)를 나타내며 접근 확률과 같은 의미로 사용될 수 있다. 이 수식을 적용할 경우 규모가 큰 인덱스 트리에서는 단말 페이지의 접근 확률은 0에 가까운 매우 작은 값을 갖게 되며 루트 페이지의 경우는 확률 1의 값으로 모든 질의에 대해 필연적으로 접근하게 된다. 또한 만약 임의의 질의에 대해서도 질의의 범위가 저장된 데이타의 분포에 영향을 받지 않는다면[2], $W1$과

2) 예를 들어 데이타가 집중된 영역에 대해 질의가 집중된다면, 질의의 범위가 저장된 데이타 분포에 영향을 받는다고 할 수 있다.

$W2$는 각각 1과 0의 값을 갖게 되어 $pop(p)$의 값이 무의미하게 된다. 일단 본 논문의 나머지 부분에서도 논의의 편의를 위해서 $W1$=1, $W2$=0의 경우만을 고려한다. 따라서 식 (1)은 다음과 같이 단순화될 수 있다.

$$RI(p)=\begin{cases}\dfrac{Measure(p)-MinM}{MaxM-MinM} & MaxM\neq MinM\text{의 경우,}\\ 1 & \text{나머지 경우}\end{cases} \quad (2)$$

이 식에서 $MinM$와 $MaxM$는 각각 $Measure$의 최소값과 최대값을 의미한다. 여기서 $MaxM$은 결국 루트 페이지의 $Measure$ 값이 된다.

알고리즘을 기술하기 위해 몇 가지 추가적인 정의가 필요하다. NZ와 CAP를 각각 인덱스를 저장하기 위해 선택된 존의 수와 존들의 총 저장 용량이라 하자. 그리고 NZ개의 각 존에 대해서 0부터 NZ-1까지의 순차적인 번호를 부여하는데 작은 값일수록 데이타 전송률이 높다고 가정하자. 따라서 존 0와 존 NZ-1은 각각 가장 빠른 존과 가장 느린 존을 의미한다. 그리고 $ZCap[zid]$를 존 zid의 저장 용량이라고 하자. 그러면 총 저장 용량에 대한 $ZCap[zid]$의 상대적인 비율인 $ZR[zid]$는 $ZCap[zid]/Cap$로 정의할 수 있다.

마지막으로 각 존을 동등하게 활용하기 위한 알고리즘을 위해서는 각 존의 활용 정도를 나타내는 기준이 필요하다. TS를 인덱스 트리를 구성하는 페이지의 총 수라고 가정할 때 존 zid에 대한 존 활용도(zone utilization indicator: ZUI)는 다음과 같이 정의될 수 있다.

$$ZUI[zid]=ZC[zid]/(TS\times ZR[zid]),\ zid=0,\ldots,(NZ-1) \quad (3)$$

단, $ZC[zid]$는 존 zid에 저장된 실제 페이지의 수이고, $(TS\times ZR[zid])$는 모든 존이 균등하게 활용될 경우 존 zid에 저장될 페이지의 수를 의미한다.

예를 들어 $ZUI[zid]>1$일 경우 존 zid에는 페이지들이 적정수보다 과도하게 할당되었음을 의미하며 반대로 $ZUI[zid]<1$은 적정수보다 적게 저장되었음을 의미한다.

이러한 정의들을 이용하여 SMD-Zoning 알고리즘은 비교적 간단하게 다음 두 단계로 구현할 수 있다.

단계 1. 인덱스 구조에 있는 모든 페이지를 식 (2)의 RI 값의 내림차순으로 정렬한다.

단계 2. 0부터 NZ-1까지의 각 존 zid에 대해서, 가장 큰 RI 값을 갖는 페이지부터 차례로 $ZR[zid]\times TS$개의 페이지들을 존 zid에 할당한다.

앞서 언급한 바와 같이 zid 0은 가장 빠른 존을 나타낸다. 이 알고리즘에서 모든 페이지들은 해당 페이지의 접근 확률에 따라 존에 배정된다. 즉, 높은 접근 확률을 갖는 페이지 일수록 보다 빠른 존에 배정한다. 또한 이 알고리즘은 특정 존에 페이지들이 몰리는 현상을 방지하기 위하여 존의 활용도에 따라 균등한 배분이 이루어지는 것을 보장하게 된다. 즉 각 존 zid에 대한 $ZUI[zid]$는 항상 1이 된다.

OLTP와 같이 데이타베이스에 대한 갱신이 자주 발생하는 환경에서는 필연적으로 각 인덱스 페이지에 대한 갱신이 발생한다. 따라서 갱신된 페이지에 대해서 저장될 존도 새롭게 결정해야한다. 그러나 이러한 작업은 SMD-Zoning 알고리즘에 비해 간단한 문제가 아니다. 그 이유는 우선 인덱스 페이지들에 대한 삽입과 삭제가 실행시간(run-time)에 수시로 발생되고, 수정된 페이지를 존에 할당하기 위해서는 이미 저장된 다른 페이지들에 대한 정보가 필요하나 실시간으로 그 것들을 알아내기가 간단하지 않기 때문이다. 이러한 가정을 기반으로 각 인덱스 페이지를 동적으로 존에 할당하는 알고리즘인 DMD-Zoning은 그림 2와 같다.

```
int Dynamic_Zoning(index_page)
{
  double C_ZR; //Cumulative ratio of the following measure:
  double Measure;

  Measure = Volume(index_page); // or Margin(index_page);

  //Update MinMeasure and MaxMeasure
  if(Measure < MinMeasure) MinMeasure = Measure;
  if(Measure > MaxMeasure) MaxMeasure = Measure;

  //Define relative importance (RI) of index_page
  if(MinMeasure == MaxMeasure) RI = 1.0;
  else RI=(Measure-MinMeasure)/(MaxMeasure-MinMeasure);

  //Assign a zone to an index page
  for(C_ZR=0, i=0; i<NZ; i++) { //-----------------------------------(A)
        ZUI = ZC[NZ-i-1]/ (TS*ZR[NZ-i-1]);
        C_ZR += ZR[NZ-i-1];
        if(RI <= C_ZR && ZUI <= 1)
              return(NZ-i-1);
  }
  return(0);
}
```

그림 2 DMD-Zoning 알고리즘

이 알고리즘은 인덱스 페이지를 인자로 받아들여 이 페이지가 저장될 존의 번호를 반환한다. 이 알고리즘에서 (A)의 for loop 연산에서 보자. 여기서 존의 탐색을 zid의 내림차순 즉, 속도가 느린 존부터 탐색하여 페이지의 RI와 각 존의 저장 용량 및 활용도에 따라 가장 적절하게 할당될 존을 찾아나간다. 이때 for loop내의 if 조건식에서 ZUI를 고려하지 않는다면 대부분의 하위 페이지들이 느린 존들에 집중적으로 할당되게 된다. 이렇게 일부분의 존에 집중적으로 할당되면 성능의 저하뿐만 아니라 많은 존들의 활용도가 현저하게 떨어지는 결과를 낳게 된다. 따라서 ZUI값이 1보다 큰 경우는 그렇지 않은 존이 발견될 때 까지 zid의 내림차순으로 존을 검색함으로써 가능한 한 모든 존이 균등하게 활용할 수 있도록 한다.

3. 다중 존에서의 질의 처리

다차원 인덱스에 대한 범위 질의를 처리하기 위해서는 루트 페이지로 부터 단말 페이지 방향으로 검색(search)이 진행되며, 대부분 하나 이상의 경로(path)를 따라 검색하게 된다. 각 중간 페이지에서는 엔트리를 하나씩 비교하여 다음 검색 경로가 되는 자식 페이지들을 선택하게 된다. 일반적으로 검색 경로는 질의 조건으로 주어진 영역과 자식 페이지들이 표현하는 영역이 겹치는 부분이 있는 가를 검사함으로써 이루어진다. 최종적으로 단말 페이지에서는 질의 영역에 완전히 포함되는 데이타 엔트리나 객체를 선택함으로써 질의의 결과를 생성하게 된다. 이와 같은 검색 방법은 트리 형태의 다차원 인덱스에서 질의 결과를 생성하는 가장 일반적인 방법으로[11,12] 본 장에서는 이와 같은 범위 질의에 대한 처리 기법을 중심으로 다중 존을 고려한 질의 처리 기법을 소개한다.

앞서 언급한 바와 같이 인덱스 검색은 순차적인 페이지들에 대한 접근을 통해 이루어진다. 다중 존에 저장된 인덱스 환경에서 이러한 순차적 접근을 가장 효율적으로 수행하는 방법은 인덱스 페이지에 대한 검색을 존 단위로 지역화(localization)하는 것이다. 즉, 하나의 존 내에서 필요한 페이지들을 모두 검색한 후에 다른 존을 검색한다. 결국 존 단위로 순차적인 검색이 가능해져 존 간의 이동이 최소화된다. 따라서 디스크의 탐색 시간을 최소화함으로써 획기적인 질의 성능의 향상을 기대할 수 있다. 이 때 고려해야 할 중요한 사항은 트리에서 각 페이지들 간의 관계이다. 즉, 직접적인 조상-자손의 관계가 아닌 이상 인덱스 트리에서 레벨이 낮은 페이지가 더 높은 페이지 보다 더 큰 *RI* 값을 가질 수 있으며, 같은 레벨에 존재하는 페이지 간에도 서로 다른 *RI* 값을 가질 수 있다. 따라서 기존의 깊이우선 검색(depth-first search)이나, 너비우선 검색(breadth-first search) 같은 단순한 트리 검색 알고리즘으로는 지역화된 검색이 불가능하며 새로운 검색 알고리즘이 요구된다.

2장에서 제안한 SMD-Zoning 알고리즘은 임의의 페이지에 대해서도 그의 조상 페이지들보다 항상 작거나 같은 *RI* 값을 갖는 것을 보장한다. 따라서 이 알고리즘에서는 트리의 루트부터 단말 페이지까지의 모든 경로에 대해서 방문하는 각 페이지의 존 번호는 항상 오름차순이 된다. 이러한 조건이 DMD-Zoning 알고리즘에서도 만족되기 위해서는 모든 수정된 페이지들은 이 알고리즘을 이용하여 배정된 새로운 존에 저장되고, 이와 더불어 이 페이지의 모든 조상 페이지들에 대해서도 루트 방향으로 차례로 같은 함수를 이용하여 저장될 새로운 존을 배정 받도록 재계산되어야 한다.

이러한 조건하에 그림 3의 질의 처리 알고리즘은 각 인덱스 검색 연산에 대해서 존에 대한 순차적 검색, 즉 지역화된 검색이 보장된다. 우선 이 알고리즘에서 각 존마다 IPA라는 FIFO 큐 배열이 하나씩 존재한다. 이 자료구조는 각 존에서 질의 조건을 만족하는 단말 페이지를 찾아가기 위해 조사해야하는 페이지들을 저장하고 있다. 따라서 가장 빠른 존부터 시작해서 각 존의 IPA에 저장된 페이지들을 검색하고, IPA가 빈(empty) 상태가 되면 다음 존의 IPA를 검색한다. 이러한 방식으로 페이지들을 검색하게 되면 각 존에 대한 지역화된 검색이 보장된다. 그림 3에서 (B)의 for-loop에서 호출되는 각 zone_search 함수는 하나의 존에 대한 검색을 나타낸다. 함수 zone_search에서는 IPA에 저장된 각 페이지들을 디스크로 부터 읽은 후에 단말 페이지일 경우 질의 조건이 만족하는지 검사 후에 질의 결과를 생성하고, 중간 페이지일 경우 질의 조건을 만족하는 각 엔트리에 대해 그 자식 페이지를 add_queue 함수를 이용하여 해당 IPA에 저장하게 된다.

4. 실험 결과

본 논문에서 제안한 다중 존 디스크 환경에서의 알고리즘들에 대한 우수성 및 실용성을 검증하기 위해 다양한 조건에서의 실험을 실시하였다. 실험은 기존의 디스크 존을 고려하지 않은 순수한 R^*-트리와 본 논문에서 제안한 Area-Zoned R^*-트리 및 Margin-Zoned R^*-트리에 대해서 실시하였다. 즉, Area-Zoned R^*-트리와 Margin-Zoned R^*-트리는 디스크 존을 고려하여 저장된 트리를 나타내며, 각각은 *RI*를 계산할 때 정의 1과 정의 2에서 정의한 area와 margin을 이용하여 존에 저장된 구조를 의미한다. 또한 SMD-Zoning 알고리즘과 DMD-Zoning 알고리즘 모두 Area-Zoned R^*-트리와 Margin-Zoned R^*-트리에서 구현되었다.

본 실험에서는 각 페이지의 디스크 접근시간을 계산하기 위해 표 2에 있는 Seagate Barracuda 7200.7의 디스크 모델을 이용하였다. 이 표에서 마지막 열은 특정 존에서 디스크 접근이 지역화되었을 경우 평균적인 페이지의 읽기/쓰기 시간을 millisecond 단위로 나타낸다. 여기서 특정 존 내에서 일정시간 동안 디스크 접근이 발생된다고 가정하자. 물리적 존 0은 가장 빠른 데이타 전송률을 갖고 있다 하더라도 그 존 내에서의 실제 페이지 접근 시간이 가장 빠르다는 것을 보장하지 않는다. 그 이유는 존 0이 가장 많은 수의 실린더로 구성되어 있으므로 탐색 시간이 다른 존에 비해 그 만큼 크기 때문이다. 같은 이유로 가장 작은 용량을 갖는 존이 그만큼 작은 수의 실린더가 할당되므로 실제 페이지 접근시간이 오히려 빠를 수 있다. 표 2는 이와 같은 존 용량과 페이지 접근 시간과의 관계를 잘 보여주고 있다.

```
NZ // the total number of existing zones (an environmental constant)
zone_locks [NZ] // an integer semaphore array; one for each zone
struct index_queue {
        PID ; //a page ID
        Level ; //an index tree level
}
index_queue IPA [NZ][ ]; // each zone is associated with a FIFO
                         // queue of the IDs of the pages to be accessed
                         // by currently running threads.
```

```
search_main (q) //q is a range query
{
  current_depth = 1; //the root level is level 1
  read_page(root);
  if(current_depth==tree_height) { //root is the only leaf page
     get_result(D);
  }
  else {
     for(i=0; i<NZ; i++) wait(zone_locks[i]); //lock all zones
     first_zone = NZ;
     for each index entry I_i {
          if check_predicate(I_i) {
             if(zone(I_i.child_pointer) < first_zone)
               first_zone = zone(I_i.child_pointer);
             add_queue(I_i);
          }
     }
     for (i=first_zone; i<NZ; i++) { ---------- (B)
        signal (zone_locks[i]); //unlock zone i
        zone_search(i);
        wait (zone_locks[i]); //lock zone i
     }
     // The query has been completed;
     for(i=0; i<NZ; i++)
        signal(i); //release all zone
  }
}
```

```
zone_search (zid)
{
  int pid, level;
  while(IPA[zid] queue is not empty) repeat {
     <pid, level> = get_fifo (IPA[zid]);
     read_page(pid);
     if(level == tree_height) { // pid is a leaf
          get_result(D);
     }
     else {
          for each index entry I_i {
             if check_predicate(I_i)
                  add_queue(I_i);
          }
     }
  } //while loop
}
get_result (D) {
  for each data entry D_i {
          if D_i satisfies the query predicate
             add D_i to the result set.
  }
}
add_queue (I_i) {
   add <I_i.child_pointer, current_depth+1>
          to IPA[zone(I_i.child_pointer)] queue;
}
```

그림 3 지역화된 질의 처리 알고리즘

표 2 실험에 사용된 디스크 모델

DCap (Disk capacity): 200 GB; Average seek time: 7.5 ms; Average rotational latency: 4.17 ms; Average data transfer rate of the disk: 50.47 MB/sec

zid	physical zone number	*ZCap* (GB)	*ZR*	average 8KB page read/write time in ms (the zoned R*-trees)
0	8	6	0.03	4.41270302
1	11	6	0.03	4.454687191
2	4	9	0.045	4.491726772
3	14	6	0.03	4.510334263
4	7	9	0.045	4.526061249
5	12	8	0.04	4.573418958
6	10	9	0.045	4.578541932
7	13	8	0.04	4.595950336
8	5	12	0.06	4.613985193
9	2	14	0.07	4.625634668
10	1	17	0.085	4.703298305
11	6	14	0.07	4.704762063
12	9	13	0.065	4.72599499
13	3	21	0.105	4.892135707
14	0	48	0.24	5.64322693

본 실험에서는 디스크의 I/O 시간에 중점을 두기 위해 하나의 페이지 버퍼(buffer)만을 가정했으며 프리 패칭(pre-fetching) 역시 가정하지 않았다. 페이지 크기는 8KB로 고정하였고, 페이지에 저장되는 모든 값은 4바이트로 제한하였다.

4.1 SMD-Zoning

첫 번째 실험은 균일하게 분포된 데이타에 대해서, 차원의 수를 2부터 64까지 변화시키며 실시하였다. [0,1]의 범위를 갖는 각 차원공간에 대해서 131,072(2^{17})개의 임의의 포인트 데이타를 갖는 화일을 생성하였으며 이 데이타 화일을 이용하여 R*-트리와 Margin-Zoned R*-트리, Area-Zoned R*-트리를 각각 구축하였다. 각 트리에 대한 검색 성능은 2000개의 범위 질의로 구성된 네 개의 질의 집합에 대해서 측정하였다. 처음 세 집합은 임의의 중심 포인트로 부터 각 차원의 양방향으로 각각 0.02, 0.1, 0.25 만큼 크기의 범위를 갖는 질의들을 생성하였다. 그리고 나머지 하나의 집합은 임의의 중심 포인트로 부터 부피가 0.0001이 되도록 각 차원 방향으로 범위를 설정하였다. 즉, 마지막 집합의 질의들은 범위가 전체 데이타 공간의 0.01%가 되도록 설정한 것이다.

이러한 환경에서 실험한 결과는 그림 4에서 보여주고 있다. 우선 그림 4(a)에서 보는 바와 같이 다중 존에 저

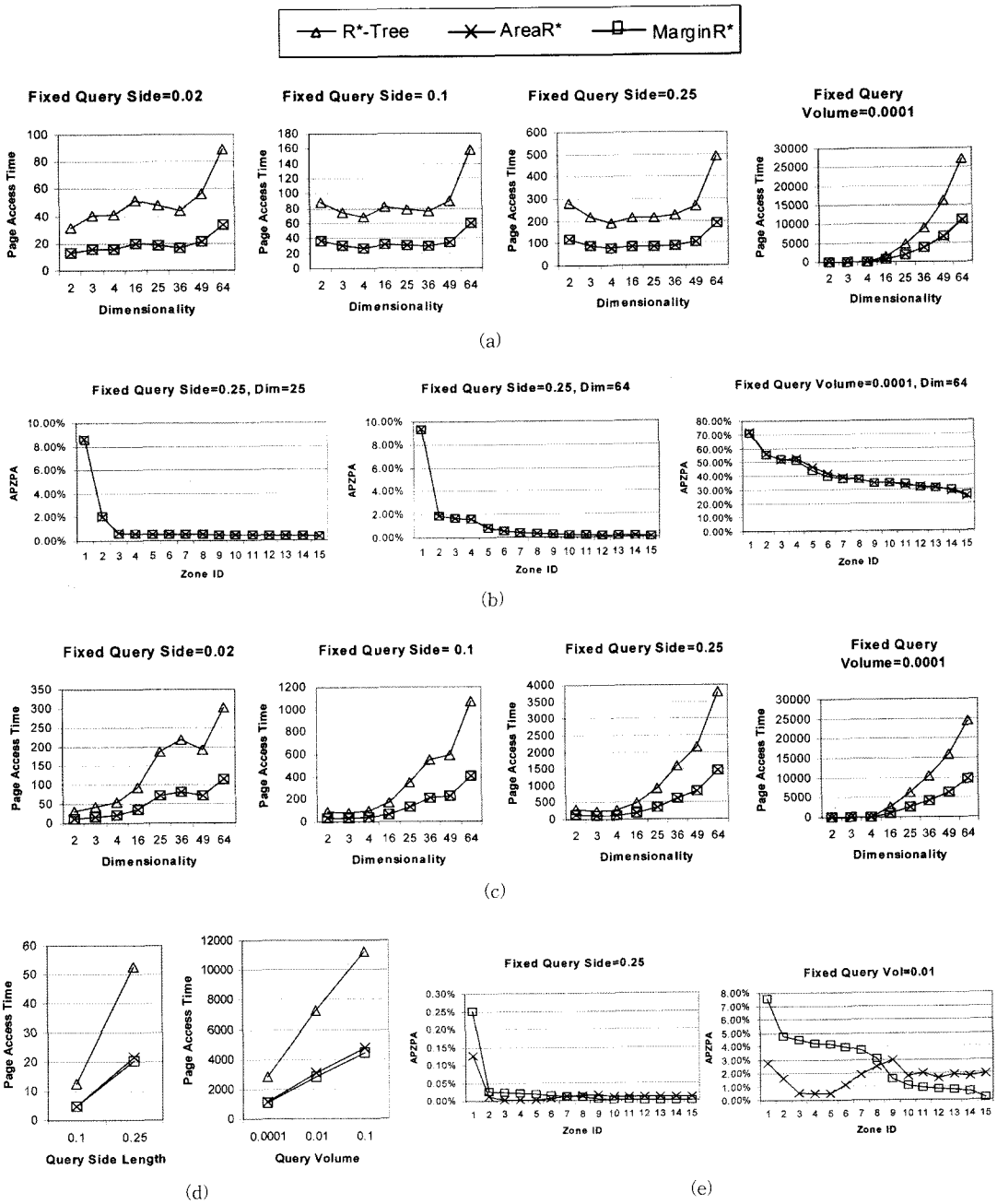


그림 4 SMD-Zoning에서의 실험결과: (a) 평균 페이지 접근시간(millisecond), (b) *APZPA*, (c) 클러스터 데이타의 페이지 접근시간(millisecond), (d) 실제 데이타의 페이지 접근시간(millisecond), (e) 실제 데이타의 *APZPA*

장된 R*-트리는 기존의 R*-트리 비해서 평균 페이지 접근시간(하나의 질의를 처리하기 위해 필요한 페이지들을 디스크로 부터 읽는 평균시간: 단위는 millisecond)이 매우 적은 것을 알 수 있다. 또한 질의의 종류나 차원의 수에 관계없이 대부분의 경우 R*-트리에 비해 2~3배 정도의 꾸준한 성능 향상을 기대할 수 있다는 사실을 보여준다.

그림 4(b)는 하나의 질의에 대해서 각 존 내에서 접근되는 페이지들의 평균 비율(*APZPA: Average Percentage of Zoned Pages Accessed*)을 보여준다. 이

그림에서 보는 바와 같이 빠른 존에서 더 많은 비율의 페이지들을 접근하는 것을 확인할 수 있다. 이는 빠른 존에 저장된 페이지일수록 더 높은 접근 확률을 가지며, 질의 당 접근되는 페이지 수가 그렇지 못한 존에 비해 많다는 것을 의미한다.

두 번째 실험에서는 데이타가 일부 영역에 집중되어 있는 경우에 대해 실험하였다. 10개의 데이타 클러스터(cluster)를 생성하였는데 각 클러스터는 차원별로 0.01에서 0.3 정도의 크기를 갖도록 하였으며 101,072개의 포인트 데이타가 각 클러스터에 분산되어 위치하도록 하였다. 이와는 별도로 클러스터에 관계없이 30,000개의 데이타 객체를 임의의 위치에 생성하였다. 이러한 환경에서의 실험 결과는 그림 4(c)에 있다. 이 그림에서 보는 바와 같이 실험 결과는 그림 4(a)와 거의 유사하다는 것을 알 수 있다.

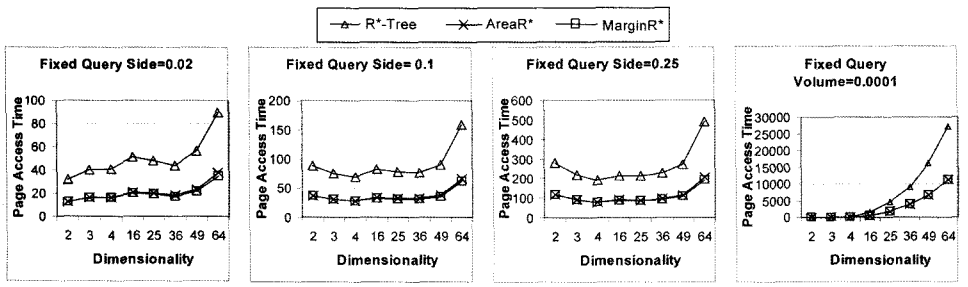
마지막 실험은 대용량의 실제 데이타를 이용하여 수행하였다. 이 데이타는 통신 회사의 데이타베이스에서 실제 운영되던 것으로 1,028,872개의 레코드로 구성되어 있다. 이 데이타는 실제로 모두 19개의 속성들로 구성되어 있으나 데이타 분석을 통해 모두 25개의 차원을 갖는 데이타로 변환하였으며 각 차원도 0과 1 사이의 값을 갖도록 정규화하였다. 이 데이타로 구성된 R*-트리의 성능을 측정하기 위해 각각 3000개로 구성된 5개의 질의 집합을 생성하였다. 처음 두 집합에서는 각 차원의 질의 범위가 각각 0.1과 0.25가 되도록 생성하였다.

다음 세 집합에서는 각 질의의 범위 크기(부피)가 각각 0.0001, 0.01, 0.1이 되도록 생성하였다.

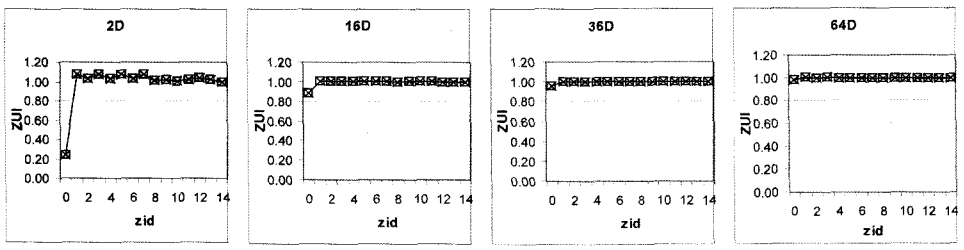
실험 결과는 그림 4(d)에 나타나있다. 이 그림에서 보는 바와 같이 실험 결과는 이전 실험들과 유사한 추세를 보여준다. 하지만 Margin-Zoned R*-트리의 경우 Area-Zoned R*-트리에 비해서 페이지 접근 시간이 5%~10% 정도 더 줄어든 결과를 보여주고 있다. 그 이유는 그림 4(e)의 그래프로 부터 설명이 가능하다. 이 그림에서 보는 바와 같이 Margin-Zoned R*-트리에서의 페이지 접근은 Area-Zoned R*-트리에 비해 빠른 존에 치우쳐 있으며 이로 인해 더 빠른 페이지 접근 시간을 보여주고 있다. 이러한 차이는 실제 데이타를 이용한 실험에서만 관찰할 수 있었는데, 실제 데이타는 임의로 생성된 데이타에 비해 어느 정도 치우친(skewed) 데이타 분포를 갖는다. 따라서 페이지의 영역은 각 차원간에 길이에 많은 차이가 생기게 되며, 결과적으로 *margin*을 이용할 경우 페이지 간의 *RI*의 변화가 *area*을 이용한 경우보다 극명하게 차이 나기 때문이다.

4.2 DMD-Zoning

DMD-Zoning 알고리즘에 대한 실험 환경은 SMD-Zoning의 경우와 동일하다. 다만 인덱스 트리의 페이지들은 데이타가 하나씩 삽입될 때 필요에 따라 동적으로 생성되며 저장될 존의 결정도 동적으로 이루어진다. 그림 5, 6, 7은 각각 DMD-Zoning에서 데이타의 균일한 분포, 집중된 분포, 마지막으로 실제 데이타에서의 실험 결과를 보여준다. 그림 5(a), 6(a), 7(a)에서 보는 바와 같이 DMD-Zoning은 SMD-Zoning과 같이 평균 페이



(a)



(b)

그림 5 균일분포에서의 DMD-Zoning 실험결과: (a) 페이지 접근시간(millisecond), (b) ZUI 측정 결과

지 접근시간을 최소화함으로써 질의 성능을 향상시킨다는 것을 알 수 있다.

SMD-Zoning과는 달리 DMD-Zoning에서는 모든 존의 활용도가 항상 동일한 상태로 유지되는 것이 보장되지 않는다. 따라서 DMD-Zoning에서는 각 존에 대한 활용도 변화의 분석이 매우 중요하다. 그림 5(b), 6(b), 7(b)는 각 존의 활용도에 대한 실험 결과를 보여준다. 우선 그림 5(b)에서 보는 바와 같이 동일한 데이타 분포 환경에서는 차원이 클 경우 모든 존이 거의 동일한 활용도를 갖는다는 것을 알 수 있다. 그러나 차원이 작을 경우 존 0은 활용도가 1보다 작은 값을 갖는다. 이 결과는 다음 두 가지 이유에 기인한다. 우선 작은 차원을 갖는 경우는 위치 값을 저장하기 위해 많은 차원을 갖는 경우보다 작은 용량만이 필요하다. 따라서 동일한 페이지 크기에 보다 많은 엔트리를 저장할 수 있으므로 결과적으로 동일한 수의 데이타를 저장하는데 있어서 보다 작은 수의 페이지만이 필요하다. 예를 들어 본 실험에서 2, 16, 64차원의 R*-트리에서는 각각 280, 1616, 6550개의 페이지가 필요하였다. 따라서 작은 차원에서는 저장된 페이지의 수에 약간의 변화가 있어서 활용도 값은 큰 차원의 경우 보다 민감하게 변화한다.

두 번째 이유는 작은 차원의 경우 각 페이지의 fanout이 크므로 큰 차원보다 트리의 하단으로 내려갈수록 area나 margin이 급격한 비율로 작아진다. 따라서 상단 부분에서는 페이지 간에 *RI* 값의 편차가 더욱 크게 된다. 예를 들어 본 실험에서 2차원 데이타의 경우 존 0과 1은 존 용량이 같음에도 불구하고 저장된 페이지의 수가 각각 2와 9개로 큰 차이가 발생하였다. 즉, 존 0에 저장될 정도의 *RI* 값을 갖는 페이지는 루트를 포함해서 하나 혹은 둘 정도밖에 되지 못한다.

그림 6(b)는 집중된 분포의 데이타에서 ZUI의 변화를 보여준다. 이 그림에서 큰 차원의 데이타에 대해서 Margin-Zoned R*-트리는 존 0에서 큰 ZUI 값을 갖는다. 그 이유는 다음과 같이 해석할 수 있다. 큰 차원의 경우 인덱스 페이지의 margin은 area보다 트리의 아래로 내려갈수록 훨씬 느린 비율로 작아진다. 따라서 빠른 존의 활용도가 느린 존과 유사한 확률로 1보다 크거나 같게 되므로, 결과적으로 추후에 저장될 페이지들이 존 0으로 비교적 쉽게 보내는 결과를 낳게 된다. 같은 이유로 그림 6(b)의 존 12, 13, 14와 같이 느린 존에서의 활용도가 1보다 작은 경우도 발생한다. 마지막으로 그림 7(b)는 실제 데이타의 ZUI를 보여주는데 가장 이상적인 결과를 보여주고 있다.

전반적으로 본 논문에서 실시한 실험에서는 다중 존 디스크를 이용한 인덱스 저장 기법이 기존의 방법에 비해서 평균 60%정도의 꾸준한 성능 향상을 보였다. 그 이유는 본 논문이 목표한 대로 빠른 존에 더 높은 접근 빈도를 갖는 페이지를 저장하고, 존 간의 지역화를 이용한 질의 처리 알고리즘을 활용했기 때문이다. 그림 4(d)와 7(a)에서 보는 바와 같이 실제 데이타에서 존의 활용도는 거의 동일하지만 Margin-Zoned R*-트리에서의 성능 향상 정도가 Area-Zoned R*-트리에 비해 약간

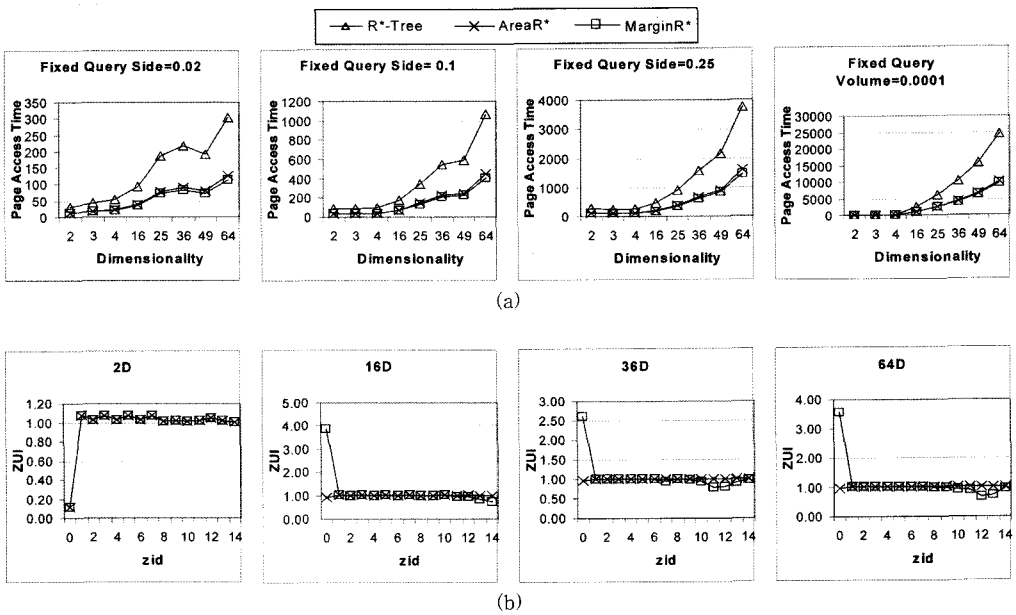


그림 6 치우친 분포에서의 DMD-Zoning 실험결과: (a) 페이지 접근시간(millisecond), (b) ZUI 측정 결과

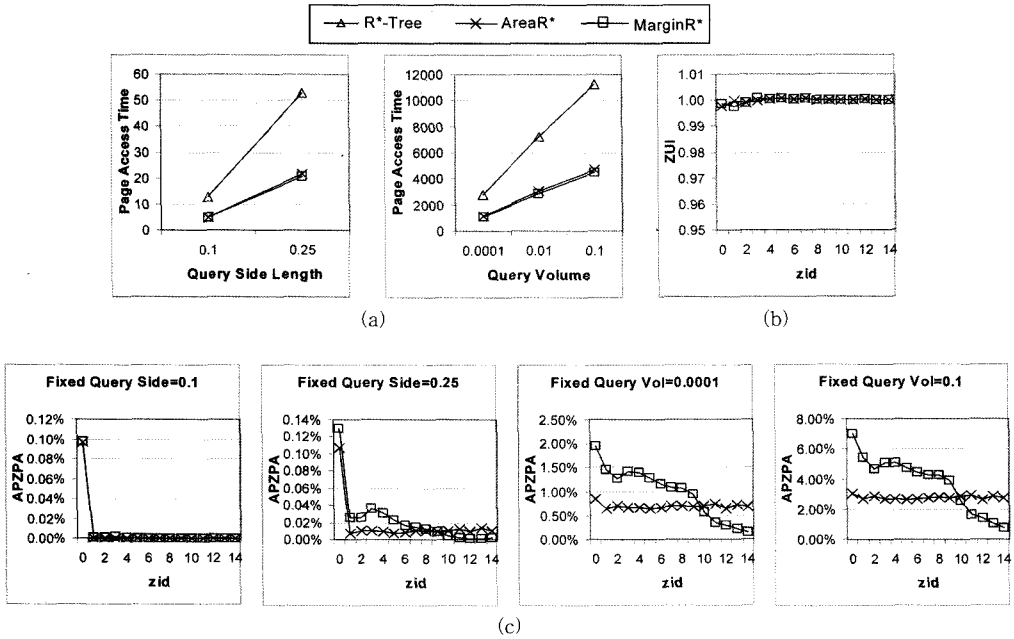


그림 7 실제 데이타에서의 DMD-Zoning 실험결과: (a) 페이지 접근시간(millisecond), (b) ZUI 측정 결과, (c) APZPA 측정 결과

높은 것을 알 수 있다. 그 이유는 그림 4(e)와 그림 7(c)에서 찾을 수 있는데, Margin-Zoned R*-트리의 APZPA 값은 빠른 존에서 비교적 높게 나타난 반면에 Area-Zoned R*-트리는 존 간의 변화가 비교적 작다는 것을 알 수 있다. 반면에 Margin-Zoned R*-트리나 Area-Zoned R*-트리 모두 존 활용도에서 대부분의 경우 좋은 결과를 보이나 Area-Zoned R*-트리가 집중된 분포의 데이타에서 더 좋은 활용도를 보였다.

5. 결 론

본 논문에서는 질의 처리 성능을 향상시키기 위해서 다차원 인덱스를 다중 존 디스크에 효율적으로 저장하는 기법을 제안하였다. 제안된 기법은 데이타베이스의 응용에 따라 정적 환경과 동적 환경을 모두 고려하여 설계되었다. 제안된 방법은 디스크 존들이 균일한 활용도를 가지며 자주 접근되는 페이지를 보다 빠른 존에 할당하는 방법을 사용하였다. 또한 이 기법들은 질의 처리 시간에 존 간의 이동을 최소화하는 지역화 개념을 적용함으로써 질의 성능을 극대화할 수 있었다. 제안된 기법들은 실제 데이타를 비롯한 다양한 종류의 실험을 통해 기존의 방법에 비해 획기적으로 질의 성능을 높인다는 사실을 확인 할 수 있었다.

본 논문에서 제안한 알고리즘은 실험에 이용한 R*-트리 이외에 다양한 인덱스 환경에서도 적용이 가능하다. 예를 들어 KDB-tree[13,14]나 R-tree[4,12,15,16], QSF-tree [11], B-tree[17]등의 인덱스 기법은 R*-tree와 동일한 방법으로 영역을 설정하므로 수정 없이 바로 적용될 수 있다. 반면에 SS-tree[18]나 TV-tree[19]등은 영역 설정 방법이 위의 기법들과 다르므로 직접적으로 이용할 수는 없으나 본 논문에서 제안한 정의 1, 2를 수정함으로써 바로 적용할 수 있으며, 이외의 어떠한 계층적 인덱스 기법에도 이와 같은 방법으로 적용이 가능하다.

본 논문에서는 하나의 버퍼만을 가정한 실험 결과를 제시하였다. 하지만 존을 고려한 페이지 저장 기술은 버퍼링 기법과 연관되어 고려될 수 있다. 예를 들어 인덱스의 상위 페이지와 같이 버퍼에 상주되는 페이지들은 처음 한번만 접근되므로 가장 느린 존에 저장하는 것이 오히려 가장 좋은 성능을 보여 줄 수도 있다. 따라서 본 논문에서 제시한 알고리즘은 인덱스 버퍼링 알고리즘이나 예상 버퍼링, 버퍼 사이즈 등의 요소에 따라 여러 가지 방법으로 변화될 수 있으며 이에 대한 후속 연구가 요구된다. 향후에는 이에 대한 연구뿐만 아니라 다중 디스크 모델 환경에서 다중 존을 이용한 인덱스 접근 기법과 동시성을 고려한 존 배치 기법 그리고 k-nearest neighbor와 같은 다른 종류의 질의 모델에서의 존 배치 기법에 대해 연구할 계획이다. 또한 동적 환경의 다차원 접근 방법에서 보다 정확하고 현실적인 페이지 접근 확률(상대 중요도:RI)을 도출하는 기법에 대한 연구도 병행할 계획이다.

참 고 문 헌

[1] Ng, S.W., Advances in Disk Technology: Performance Issues. IEEE Computer Magazine, pages 75-81, 1998.

[2] Ruemmler, C. and Wilkes, J. An Introduction to Disk Drive Modeling, IEEE Computer, March 1994.

[3] Leutenegger, S.T. and Lopez, M.A., The Effect of Buffering on the Performance of R-trees. IEEE Transactions on Knowledge and Data Engineering, 12(1):33-44, 2000.

[4] Beckman, N., et. Al., The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In ACM SIGMOD International Conference on Management of Data, pages 322-331, 1990.

[5] Yu, B. and Kim, S.-H. Zoning Multidimensional Access Methods for Analytical Database Applications, Proc. the 3rd International Conference on Computer Science and its Applications, pp. 191-196, 2005.

[6] Yu, B. and Kim, S.-H. An Efficient Zoning Technique for Multidimensional Access Methods, Proc. the VLDB Workshop on Trends in Enterprise Application Architecture, LNCS 3888, pp. 129-143, 2005.

[7] Faloutsos, C. and Kamel, I. On Packing R-tree. In Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), pp. 490-499, 1993.

[8] Leutenegger, S.T. and Lopez, M.A. and Edingnton, J.M., STR: A Simple and Efficient Algorithm for R-tree Packing. IEEE International Conference on Data Engineering, pages 497-506, 1997.

[9] Rosenberg, A.L. and Snyder, L., Time- and Space- Optimality in B-trees, ACM Transactions on Database Systems, 6(1):174-193, 1981.

[10] Roussopoulos, N. and Leifker, D., Direct Spatial Search on Pictorial Database Using Packed R-trees, ACM International Conference on Management of Data, pages 17-31, 1985.

[11] Orlandic, R. and Yu, B., Scalable QSF-Trees: Retrieving Regional Objects in High-Dimensional Spaces, JDM Journal of Database Management, Idea Group Publishing, 15(3):45-59, 2004.

[12] Papadias, D. Theodoridis, Y., Sellis, T., & Egenhofer, M.J. Topological relations in the world of minimum bounding rectangles: A study with R-trees. Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 92-103, 1995.

[13] Orlandic, R., & Yu, B. A Retrieval Technique for High-Dimensional Data and Partially Specified Queries. DKE Data & Knowledge Engineering, Elsevier 42(2), pp. 1-21, 2002.

[14] Robinson, J.T. The K-D-B Tree: A Search Structure for Large Multidimensional Dynamic Indexes. Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 10-18, 1981.

[15] Berchtold, S. Keim, D., & Kriegel, H.-P. The X-tree: An index structure for high-dimensional data. Proc. VLDB Int. Conf. on Very Large Data Bases, pp. 28-39, 1996.

[16] Guttman, A. R-trees: A dynamic index structure for spatial searching. Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 47-54, 1984.

[17] Comer, D., The Ubiquitous B-tree. ACM Computing Surveys 11, pp. 121-137, 1979.

[18] White, D.A. and Jain, R. Similarity Indexing with the SS-tree. Proc. 12th IEEE Conf. on Data Engineering, pp. 516-523, 1996.

[19] Lin, K., Jagadish, H., and Faloutsos, C. The TV-tree: An Index Structure for High-Dimensional Data. VLDB Journal. Vol. 3, pp. 517-542, 1995.

[20] Berchtold, S., Bohm, C., & Kriegel, H.-P. The Pyramid-technique: Towards breaking the curse of dimensionality. Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 142-153, 1998.

[21] Cheng, R., Kalashnikov, D., and Prabhakar, S. Evaluating Probabilistic Queries over Imprecise Data. Proc. ACM SIGMOD International Conference on Management of Data, pp. 551-562, 2003.

[22] Jun, B., Hong, B., and Yu, B. Dynamic Splitting Policies of the Adaptive 3DR-tree for Indexing Continuously Moving Objects. Proc. DEXA International Conference on Database and Expert Systems Applications, LNCS Lecture Notes in Computer Science, Vol. 2736, pp. 308-317, Springer-Verlag, Berlin Hidelberg, 2003.

[23] Papadias, D., Tao, Y., and Sun, J. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. Proc. the VLDB International Conference on Very Large Databases, pp. 790-801, 2003.

[24] Pfoser, D., Jensen, C.S., and Theodoridis, Y. Novel Approaches to the Indexing of Moving Object Trajectories. Proc. VLDB Very Large Data Base Conference, pp. 395-406, 2000.

[25] Saltenis, S., Jensen, C.S., Leutnegger, S.T., and Lopez, M.A. Indexing the positions of continuously moving objects. Proc. the ACM SIGMOD International Conference on Management of Data, pp. 331-342, 2000.

유 병 구

1994년 항공대학교 전산과학과(학사). 2000년 Illinois Institute of Technology, Computer Science(박사). 2000년~2006년 Assistant Professor of Computer Science, University of Wyoming. 2006년~현재 Associate Professor of Computer Science and Information Systems, National University. 관심분야는 Spatial database, multimedia systems, storage systems

김 선 호

1986년 연세대학교 전자공학과(학사). 1986년~1992년 삼성전자 정보통신연구소 연구원. 1994년 University of Southern California, MS in Electrical Engineering(석사). 1999년 University of Southern California, Ph.D. in Computer Science(박사). 1999년~현재 University of Denver, Assistant Professor in Computer Science Department. 관심분야는 Spatial database, multimedia systems, storage systems.

장 재 영

1992년 서울대학교 계산통계학과(학사) 1994년 서울대학교 계산통계학과 전산과학전공 대학원(석사). 1999년 서울대학교 계산통계학과 전산과학전공 대학원(박사) 2000년 3월~현재 한성대학교 컴퓨터공학과 부교수. 관심분야는 데이타베이스, 데이타마이닝, 정보 검색