시공간 데이타베이스에서 영역 합 질의를 위한 색인 기법

(An Indexing Technique for Range Sum Queries in Spatio-Temporal Databases)

조 형 주 † 최 용 진 †† 민 준 기 ††† 정 진 완 ††††
(Hyung-Ju Cho) (Yong-Jin Choi) (Jun-Ki Min) (Chin-Wan Chung)

요 약 시공간 데이타베이스는 최근에 많은 주목을 받았지만, 영역 합 질의에 대한 연구는 그 중요성에 비하여 많이 부족하다. 영역 합 질의를 처리하기 위하여, 많은 양의 데이타에 대한 직접적인 접근은 엄청난 계산 비용을 야기하기 때문에, 최근에 기존 색인 기법을 활용한 materialization 방법이 제안되었다. 간단하면서 효과적인 방법은 시공간 조건을 가지는 윈도우 질의를 효율적인 처리하는 MVR-tree에 materialization 방법을 적용하는 것이다. 그러나, MVR-tree는 노드들 사이의 존재하는 원형 경로 때문에, 중간 노드에 미리 계산된 합을 유지하는 것이 불가능하다. 다른 색인 구조들에 기초한 집합적 구조(aggregate structures)는 만족스러운 질의 성능을 제공하지 못 한다. 본 논문에서는 적응적 분할 기법을 사용하는 새로운 색인 기법(Adaptive Partitioned Aggregate R-Tree, APART)과 다양한 환경에서 영역 합 질의를 효율적으로 처리하는 질의 처리 알고리즘을 제안한다. 실험 결과는 APART의 성능이 다양한 상황에서 기존의 집합적 색인 기법들보다 2배 이상 우월하다는 것을 보여준다.

키워드 : 시공간 데이타베이스, 영역 합 질의, 적응적 분할 기법

Abstract *Although spatio-temporal databases have received considerable attention recently, there has been little work on processing range sum queries on the historical records of moving objects despite their importance. Since to answer range sum queries, the direct access to a huge amount of data incurs prohibitive computation cost, materialization techniques based on existing index structures are recently suggested. A simple but effective solution is to apply the materialization technique to the MVR-tree known as the most efficient structure for window queries with spatio-temporal conditions. However, the MVR-tree has a difficulty in maintaining pre-aggregated results inside its internal nodes due to cyclic paths between nodes. Aggregate structures based on other index structures such as the HR-tree and the 3DR-tree do not provide satisfactory query performance. In this paper, we propose a new indexing technique called the Adaptive Partitioned Aggregate R-Tree (APART) and query processing algorithms to efficiently process range sum queries in many situations. Experimental results show that the performance of the APART is typically above 2 times better than existing aggregate structures in a wide range of scenarios.*

Key words : spatio-temporal databases, range sum queries, adaptive partitioning method

1. 서 론

이동 객체에 대한 응용 분야가 빠르게 증가함에 따라서, 시공간 데이타베이스에서 aggregation은 데이타 분석을 위한 가장 중요한 연산자들 가운데 하나가 되었다. 시공간 데이타베이스는 이동 객체들의 과거 행적들에 정보를 저장한다. 비행기, 사람, 자동차와 같은 전형적인 이동 객체들 뿐만 아니라, 공간 상에서 발생하는 사건들도 일종의 이동 객체로 간주될 수 있다. 왜냐하면, 사건들도 공간 정보와 시간 정보의 조합으로 구분될 수 있

† 비 회 원 : LG전자 DM 연구소 연구원
hjcho@islab.kaist.ac.kr
†† 비 회 원 : 한국과학기술원 전산학과
omni@islab.kaist.ac.kr
††† 정 회 원 : 한국기술교육대학교 인터넷 미디어학부 교수
jkmin@kut.ac.kr
†††† 종신회원 : 한국과학기술원 전자전산학과 및 영상정보특화연구센터 교수
chungcw@cs.kaist.ac.kr
논문접수 : 2003년 12월 30일
심사완료 : 2004년 12월 21일

기 때문이다. 예를 들어, 핸드폰의 통화 기록이나 교통 사고 기록이 여기에 속한다.

영역 질의[1-4], 최근접 질의[5-7], 궤적 질의[8]를 위한 색인 기법, 질의 처리 기법 등에 대한 다양한 연구들이 있었지만, 이동 객체에 대한 영역 합 질의에 대한 연구는 거의 없었다. 온라인 분석처리 기법(OLAP: On-line Analytical Processing)과 같은 다차원 데이타베이스와 마찬가지로, 영역 합 질의는 많은 시공간 데이타베이스 응용 분야에서 아주 중요하다. 이해를 돕기 위해서 실생활에서 사용 가능한 2가지 질의 예를 들어 보겠다.

- Q1 : 지난 주에 강남 지역에서 발생한 총 통화량이 얼마인지 알려 달라.
- Q2 : 2003년 월 단위로 각 시도 별 강우량의 합을 보여달라.

시공간 데이타는 기본적으로 <위치정보, 시간정보, 값> 3개의 속성으로 구성된다. 예를 들어, Q1질의에서 다루는 데이타에서 공간 정보는 통화할 때의 위치 정보이고, 시간 정보는 통화를 시작한 시간과 끝나는 시간 정보이고, 값 정보는 통화 시간이다. 값 정보는 사용자 질의에 따라서, 달라질 수 있다. 예를 들어, 통화한 사람의 나이가 값 정보로 사용될 수 있다. Q2의 경우에서도 강우량 정보는 <위치정보, 시간정보, 값>으로 구성된다. 예를 들어, 위치정보는 '서울', 시간정보는 '2004년 3월 31일부터 오후 2시부터 4시'이고, 값 정보는 '40mm'가 될 수 있다. 그러한 정보들은 미래 상황을 예측하거나 계획을 세울 때 유용하다. Q1과 같은 질의는 이동 통신 회사가 네트워크 트래픽을 계산하거나 새로운 기지국에 대한 수요를 예측할 때 사용될 수 있다.

영역 합 질의는 현재 운영되고 있는 데이타베이스에 저장된 정보들을 사용하여 처리될 수 있지만, 많은 양의 데이타를 처리하므로 계산 비용이 크다. 이 문제를 해결하기 위해서, 질의 실행 전에 계산 결과를 저장하는 materialization 기법이 제안되었다. 그러나, 전통적인 온라인 분석처리 기법에서 사용되는 데이타 큐브[9,10]는 시공간 분야의 영역 합 질의를 처리하기 부적당하다. 데이타 큐브를 만들기 위해서는 미리 정의된 계층 구조(예: 일/월/년)를 미리 고정시켜야 한다[5,11,12]. 그래서, 공간과 시공간 분야에서 영역 합 질의를 처리하기 위해서, 색인 기법의 계층 구조를 이용한 materialization 기법이 사용된다[5,11,12]. 직관적인 방법은 과거 이동 객체들의 정보를 가장 효율적으로 처리하는 MVR-tree[3]에 materialization 기법을 적용하는 것이다. 그러나, MVR-tree는 노드들 사이에 사이클이 있는 경로가 존재하기 때문에, materialization 기법을 적용하는 것이 어렵다[12]. 과거 이동 객체들을 저장하기 위한 다른 색인 기법들 중에서 HR-tree와 3DR-tree는 materialization 기법을 적용할 수 있지만, 성능이 좋지 않다.

공간 데이타와 달리, 시공간 데이타는 공간 정보와 시간 정보를 동시에 가지고 있다. 그러나, 시간 정보와 공간 정보는 다른 특징을 가지고 있다. 이동 객체들의 현재 및 과거 정보는 정해진 공간 영역 안에서 시간 순서대로 기록되기 때문에, 시간이 경과할수록 시간 축은 길어지지만, 공간 영역은 크게 변하지 않는다. 결과적으로, 시간 영역은 주의 깊게 분할되어야 하고, 분할된 시공간 영역은 각각 materialization 기법이 적용되는 것이 바람직하다.

현재까지 영역 합 질의 비용을 줄이기 위해서, 질의 워크로드(query workload)를 사용하여 시간축을 분할하는 방법은 없었다. 질의 워크로드는 질의 작업 부하량으로 주어진 시간 안에 시스템이 처리해야 하는 작업의 양을 의미한다. 이 논문에서는 질의 워크로드를 사용하여 시간축을 분할하는 방법과, 이것을 3DR-tree[4]에 적용하여 영역 합 질의를 구하는 방법을 제안한다. 본 논문에서는 질의 워크로드를 사용하는 색인 기법을 APART(Adaptive Partitioned Aggregate R-tree)라고 부른다. 제안된 기법들은 다른 집합적 함수(예: count와 average)에도 그대로 적용될 수 있다. 다음은 논문의 중요한 결과들을 요약한 것이다.

- 질의 분석 결과를 활용한 동적 분할 기법을 개발하고, 이것을 3DR-tree 기반의 집합적(aggregate) 색인 기법에 적용하였다.
- 분할 기법에 의해서 발생하는 중복 계산의 문제를 해결하는 영역 합 질의 알고리즘을 개발하였다.
- 실험을 통하여 APART의 성능이 기존 집합적 색인 기법보다 2배 이상 개선된 것을 확인했다.

논문의 구조는 다음과 같다. 2장은 시공간 데이타에 대한 집합적 색인 기법에 대한 관련 연구를 제시한다. 3장은 질의 정보를 이용한 동적 분할 기법과 영역 합 질의 기법을 기술한다. 4장은 다양한 질의 워크로드와 데이타를 이용한 실험 결과를 보여준다. 5장은 논문의 결론을 제시한다.

2. 관련 연구

이동 객체들의 과거 정보를 처리하기 위해서, 다양한 색인 기법들이 제안되었다. 3DR-tree[4], HR-tree[1], MVR-tree[3]는 대표적이다. Interval 질의는 연속적인 타임스탬프(timestamp)들로 이루어진 윈도우 질의를 의미하고, 타임스탬프 질의는 하나의 타임스탬프에 대한 윈도우 질의를 의미한다. 3DR-tree는 공간 속성과 시간 속성을 동시에 처리하기 위해서 3차원 R-tree[13,14]를 사용하였다. HR-tree는 하나의 타임스탬프는 하나의 R-tree를 사용하여 저장하고, 인접한 타임스탬프에서 변

경되지 않은 노드들은 공유한다. HR-tree와 3DR-tree와 비교해서, MVR-tree는 타임스탬프 질의와 interval 질의 모두에 대해서 좋은 성능을 보여준다.

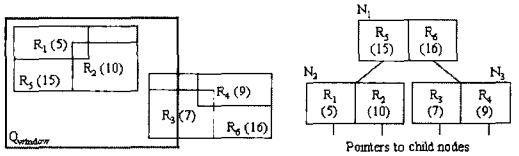


그림 1 aR-tree의 예

R-tree는 공간 데이타베이스 분야에서 가장 우수한 성능을 보여주는 색인 구조들 중의 하나라고 알려져 있다. 개념적으로, aR-tree(aggregate R-tree)[2,12]는 영역 합 질의를 위해서 R-tree를 확장하였다. aR-tree는 영역 합 질의를 효과적으로 처리하기 위해서, 중간 노드에 하위 노드들의 합을 미리 저장하고, 이것들을 사용하여 말단 노드에 대한 접근 회수를 크게 줄일 수 있다. 그림 1은 aR-tree의 예를 보여준다. 그림 1의 Q_{window}와 교차하는 레코드들의 합을 계산하는 질의가 주어지면, aR-tree는 N_2는 방문하지 않는다. N_2에 속하는 레코드들의 합은 N_1에 있는 R_5에 계산되어 있기 때문이다.

aR-tree와 유사하게, 기존의 시공간 색인 구조들을 집합적 색인 구조로 변경할 수 있다. HR-tree와 3DR-tree는 aR-tree가 사용했던 materialization 기법을 채택할 수 있다. 그러나, aHR-tree(aggregate HR-tree)와 a3DR-tree(aggregate 3DR-tree)는 HR-tree와 3DR-tree의 단순한 확장이기 때문에, 이러한 색인 기법들은 영역 합 질의에서도 좋은 성능을 제공하지 못한다. a3DR-tree는 하나의 R-tree를 사용하여 과거부터 현재까지 정보를 처리하기 때문에, 시간이 지날수록 특정한 시간을 포함하는 질의에 대해서도 항상 전체 시간 정보를 가지는 R-tree를 검색함으로 인해, 시간이 지날수록 질의 처리 성능이 떨어진다. aR-tree는 2차원상의 공간 정보를 처리하기 때문에, 시간축을 따라서 동적으로 들어오는 정보를 처리할 수 없다. aHR-tree는 매 시간마다 R-tree를 생성하기 때문에, 여러 타임스탬프에 걸치는 영역 합 질의를 처리할 경우, 여러 개의 R-tree를 동시에 검색함으로 인해, 성능이 나쁘다. 최악의 경우에 전체 시간에 대한 영역 합 질의를 처리하기 위해서는 모든 R-tree들을 검색해야 한다. MVR-tree는 기본적으로 트리 구조가 아닌 사이클(cycle)을 허용하는 그래프 구조이기 때문에, 색인 기법에서 사용할 수 있는 materialization 기법을 적용할 수 없다[3,12]. 주목할 사실은 MV3R-tree[3] 자체는 MVR-tree와 3DR-tree의 단순한 접합 구조로 3DR-tree를 이용하여 materialization 기법을 기법을 적용할 수 있지만, 영역합 질의를 처리하는 데는 성능상으로 좋지 못하다. aRB-tree (aggregate RB-tree)[12]는 미리 정해진 공간 영역에서 값이 변화하는 경우를 위한 집합적 색인 방법이다. 이 방법은 이동 객체 대신에 정해진 공간 영역(예: 도로, 행정지역)에서 변화하는 값들을 aR-tree와 aB-tree (aggregate B-tree)를 사용하여 영역 합 질의를 처리한다.

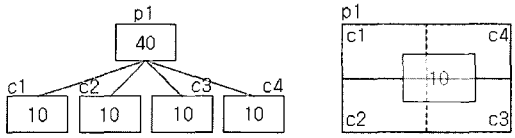


그림 2 격자 셀 기반의 색인 방법에서 materialization 기법의 적용 예

격자 셀 기반의 색인 방법(grid cell-based index) [15]을 영역합 질의에 적용할 경우 다음과 같은 제약 사항이 있다. R-tree는 데이타 분할 기법으로 같은 레벨에 있는 노드들이 공간적으로 교차하는 것을 허용하지만, 격자 셀 기반의 색인 방법은 영역 분할 기법으로 같은 레벨에 있는 노드들이 교차하지 않기 때문에, 점들로 이루어진 공간 데이타를 색인하는 데는 유용하지만, 영역을 가진 공간 데이타의 경우에는 적용하기가 어렵다. 그림 2에서 보는 것처럼 공간 데이타가 두 개 이상의 셀들에 교차할 때, materialization을 적용할 수 없다.

3. APART(Adaptive Partitioned Aggregate R-tree)

3.1절은 질의 워크로드에 기반한 동적 분할 기법을 설명한다. 다음으로, 3.2절은 APART의 영역 합, 삽입, 삭제 알고리즘을 기술한다. 3.3절은 APART의 동적 분할 기법의 이론적 배경을 제시한다.

3.1 질의 워크로드 기반의 동적 분할 기법

시공간 데이타는 공간 속성과 시간 속성을 동시에 가지고 있지만, 레코드들이 시간축을 따라서 삽입되기 때문에, 시간 영역은 증가하지만, 공간 영역은 크게 증가하지 않는다. 이런 이유로, 3DR-tree와 a3DR-tree는 증가되는 탐색 영역 때문에 성능이 좋지 않다. 본 논문에서는 a3DR-tree의 약점을 극복하기 위해서 질의 워크로드 기반의 동적 분할 기법을 a3DR-tree에 적용한다. 결과 색인 기법을 APART라고 부른다. APART는 여러 개의 분할된 a3DR-tree들로 구성되고, 각각의 a3DR-tree는 고정된 시간 영역을 책임진다.

HR-tree와 MVR-tree와는 달리, APART는 시간 영역을 분할하기 위해서 질의 워크로드 정보를 활용한다. 그림 3은 2개의 a3DR-tree들(a3DR-tree[0,5)와 a3DR-

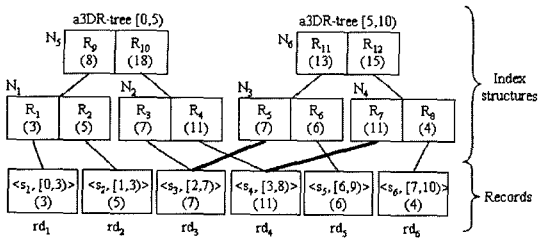


그림 3 a3DR-tree의 시간 영역 분할의 예

tree[5,10))로 구성된 APART의 예를 보여준다. 그림 3의 예제는 단지 전체 시간 영역을 2개로 나누어서 각각을 a3DR-tree가 처리하는 것을 보여줄 뿐, 최종적인 APART의 모습은 아니다. a3DR-tree [t_i, t_j)는 관할 시간 영역이 [t_i, t_j)라는 것을 의미한다. 레코드 rd=<s, [t_i, t_j), value>는 이동 객체의 하나의 스냅 샵 정보를 나타내고, s는 공간 정보를, [t_i, t_j)는 시간 정보를, value는 해당 공간 정보와 시간 정보와 관련된 측정값을 의미한다. 하나의 레코드의 시간 영역이 2개 이상의 분할된 a3DR-tree들의 시간 영역과 교차할 때는 이들 a3DR-tree들에 동시에 속한다. 예를 들어, rd_1 = <s_1, [0, 3), 3>는 a3DR-tree [0,5)에 속하고, rd_3 = <s_3, [2, 7), 7>은 a3DR-tree [0,5)과 a3DR-tree [5,10)에 동시에 속한다. 그림에서, a3DR-tree [5,10)는 2개의 레코드 rd_3와 rd_4에 대해서 중복 포인터(그림 3에서 굵은 선으로 표시된 포인터)를 가진다. 이것은 rd_3와 rd_4가 이전 a3DR-tree [0, 5)에도 속한다는 것을 의미한다.

L은 a3DR-tree [t_i, t_{i+1})의 관할 시간 영역의 길이를 의미한다. 즉, $L = t_{i+1} - t_i$로 표현된다. L을 동적으로 결정하기 위해서, APART는 질의 워크로드를 이용한다. $\overline{T_q}$는 질의들의 평균적인 시간 간격 길이를, $\overline{T_r}$는 레코드들의 평균적인 시간 간격 길이를 의미한다고 하자. $\overline{T_q}$와 $\overline{T_r}$는 질의나 레코드들이 매번 삽입될 때마다, 동적으로 계산된다. 그러면, $\overline{T_q}$와 $\overline{T_r}$를 사용하여, L을 다음과 같이 결정한다.

$$L = \max(\overline{T_q}, \overline{T_r}) \quad (1)$$

식 (1)을 사용하여, L은 질의 워크로드와 데이타로부터 얻어지는 $\overline{T_q}$와 $\overline{T_r}$에 의해서 자동적으로 결정된다. 3.3절에서 결정된 $L = \max(\overline{T_q}, \overline{T_r})$이 질의 비용이나 색인 크기 면에서 좋은 결과를 낼 수 있다는 것을 이론적으로 보여준다.

3.2 영역 합 질의 알고리즘

질의 적응적인 분할 방법은 영역 합 질의 조건이 여러 개의 a3DR-tree를 교차할 때, 정확한 질의 결과를 제공할 수 없는 경우가 발생한다. 구체적인 예로, 여러 개의 a3DR-tree를 접근하여 합을 단순하게 구할 때 문제가 발생하는 경우는 (예 1)과 같다.

예 1: 그림 3의 제시된 분할된 a3DR-tree들에 2개의 영역 합 질의 *Q1*과 *Q2*에 대한 결과를 계산해 보자. *Q1*=<*S*, [5,9), *sum*>과 *Q2*=<*S*, [3,10), *sum*>에서 공간 조건 *S*는 주어진 전체 영역을 의미한다. *Q1*은 a3DR-tree [5,10)과 교차하기 때문에, a3DR-tree [5,10)에 속하는 R_5, R_6, R_7, R_8들의 합이 질의 결과가 된다. 그래서, *Q1*의 결과는 28이 된다. *Q2*는 a3DR-tree [0,5)와 a3DR-tree [5,10)과 교차하기 때문에, a3DR-tree [0,5)에 대한 질의 결과와 a3DR-tree [5,10)에 대한 질의 결과를 더하면 된다. a3DR-tree [0,5)에서는, R_2, R_{10}에 미리 계산된 값들의 합이 질의 결과가 된다. 그래서, a3DR-tree [0,5)에서 *Q2*에 대한 영역 합은 23이 된다. 비슷한 방법으로, a3DR-tree [5,10)에서 *Q2*에 대한 영역 합은 28이 된다. 최종적인 영역 합은 51이 된다. 이것은 실제로 rd_1부터 rd_6까지 합한 값 36 보다 크다. 즉, *Q1*에 대해서는 아무런 문제가 없지만, *Q2*는 중복된 레코드들을 고려하지 않고 영역 합을 구하기 때문에 문제가 발생한다.

*Q2*에서 발생하는 문제를 해결하기 위해서는 레코드가 여러 a3DR-tree들에 속하는 경우에 주의 깊게 계산해야 한다. P_1는 a3DR-tree [0,5)에서 *Q2*를 만족하는 레코드들의 집합, P_2는 a3DR-tree [5,10)에서 *Q2*를 만족하는 레코드들의 집합이라고 하자. 그림 3으로부터, P_1={rd_2, rd_3, rd_4}와 P_2={rd_3, rd_4, rd_5, rd_6}는 쉽게 계산된다. *Q2*에 대한 최종적인 질의 결과는 ($P_1 \cup P_2$)에 속하는 레코드들의 합이 된다. Sum(P)는 집합 P에 속하는 레코드들의 합이라고 하면, $Sum(P_1 \cup P_2) = Sum(P_1) + Sum(P_2) - Sum(P_1 \cap P_2)$이다. $Sum(P_1)$은 a3DR-tree [0,5)에서 *Q2*를 만족하는 레코드들의 합이 되고, $Sum(P_2)$는 a3DR-tree [5,10)에서 *Q2*를 만족하는 레코드들의 합이 되고, $Sum(P_1 \cap P_2)$는 두 개의 a3DRt-ree에 속하면서 *Q2*를 만족하는 레코드들의 합이 된다. 즉, ($P_1 \cap P_2$)={rd_3, rd_4}이기 때문에, $Sum(P_1 \cap P_2)$=18이다.

두 개 이상의 a3DR-tree에 속하는 레코드들이 최종적인 질의 결과에 중복해서 포함되는 것을 막기 위해서, a3DR-tree의 노드 엔트리(node entry)의 구조를 다음과 같이 변경하였다.

<MBB, ptr, aggr_sum> → <MBB, ptr, aggr_sum, dup_aggr_sum>

ptr은 자식 노드나 레코드를 가리키는 포인터이고, MBB(minimum bounding box)는 ptr이 지시하는 노드 또는 레코드의 공간 영역과 시간 영역을 포함하는 3차원 사각형이고, aggr_sum은 자식 노드에 속하는 모든 레코들의 합을 의미한다. 마지막으로, 새로 추가된 dup_

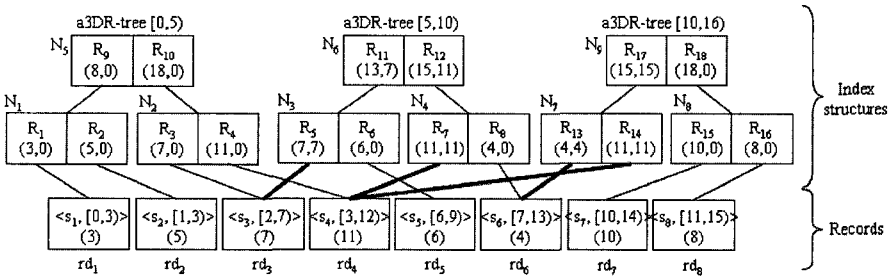


그림 4 영역 합 질의를 위한 APART의 예

```
function Range_Sum (query Q)
begin
1.    sum := 0
2.    for each a3DR-tree t_i which intersects Q do
3.      if t_i is the first a3DR-tree to be processed then      sum += Sum (t_i.root, Q, true)
4.      else                                                   sum += Sum (t_i.root, Q, false)
5.    return (sum)
end

function Sum (node N, query Q, flag First)
begin
1.    sum := 0
2.    if N is leaf-node then
3.      for each entry e_i which belongs to N do
4.        if e_i intersects Q then
5.          if First is true then sum += e_i.aggr_sum    /* in case that it is the first a3DR-tree to be processed */
6.          else                 sum += (e_i.aggr_sum - e_i.dup_aggr_sum)
7.    else
8.      for each entry e_i which belongs to N do
9.        if e_i is contained in Q then
10.         if First is true then sum += e_i.aggr_sum
11.         else                  sum += (e_i.aggr_sum - e_i.dup_aggr_sum)
12.       else if e_i partially intersects Q then     sum += Sum (e_i.ptr, Q, First)
13.   return (sum)
end
```

그림 5 APART의 영역 합 질의 알고리즘

aggr_sum은 자식 노드에 속하는 레코드들 중에서, 중복 포인터를 가지고 있는 레코드들의 합을 의미한다. 말단 노드 엔트리의 경우에, dup_aggr_sum은 0 또는 aggr_sum이 되기 때문에, 색인 구조의 크기를 줄이기 위해서, dup_aggr_sum은 플래그(flag)로 대치될 수 있다. 플래그의 값의 범위는 {0,1}이 된다. 그림 4는 APART의 예를 보여준다. rd_4=<s_4, [3,12), 11>의 경우에, 레코드의 시간 영역이 [3,12)이기 때문에, a3DR-tree [0,5), a3DR-tree [5,10), a3DR-tree [10,16)에 모두 속한다. 그래서, rd_4를 저장하기 위해서, R_4= (11,0), R_7=(11,11), R_{14}=(11,11)가 존재한다. 본 논문에서는 편의상 R_k=(x,y)를 사용하였다. x와 y는 R_k의 aggr_sum과 dup_aggr_sum을 의미한다. R_4, R_7, R_{14}의 dup_aggr_sum의 차이는 R_4는 rd_4를 저장한 처음 레코드이고, R_7과 R_{14}는 R_4가 가리키는 레코드 rd_4를 중복해서 저장하기 때문이다.

그림 5는 APART에서 영역 합을 계산하는 알고리즘을 보여준다. 그림 4의 예제를 사용하여, 알고리즘의 동

```
function Insert_Record (record rd=<s, [t_i,t_j), value>)
begin
1.    for each a3DR-tree t_i which intersects rd do
2.      if t_i is the first a3DR-tree to be processed then        Insert (t_i, rd, true)
3.      else                                                      Insert (t_i, rd, false)
end

function Insert (a3DR-tree t, record rd, flag First)
begin
1.    call ChooseLeaf to select a leaf node L in which to place rd
2.    /* a new entry e is created and installed in L */
3.    if First is true then /* i.e., it is the first a3DR-tree to be processed */
4.      e.aggr_sum = rd.value      and      e.dup_aggr_sum = 0
5.    else
6.      e.aggr_sum = rd.value      and      e.dup_aggr_sum = rd.value
7.
8.    call AdjustTree to propagate changes upward
9.    /* let e_p be an entry which indicates a node N_c on the path from L to the root */
10.       e_p.aggr_sum = 0      and      e_p.dup_aggr_sum = 0 /* 0              */
11.       for each entry e_i which belongs to N_c do
12.           e_p.aggr_sum += e_i.aggr_sum      and      e_p.dup_aggr_sum += e_i.dup_aggr_sum
End
```

그림 6 APART의 레코드 삽입 알고리즘

작 방법을 설명하겠다. *Q3*=<*S*, [7,15), *sum*>가 주어졌다고 가정하자. 이전과 마찬가지로 *S*는 전체 공간 영역을 나타낸다. *Q3*를 처리하기 위해서, a3DR-tree [5,10)과 a3DR-tree [10,16)이 검색된다. 이 경우에 a3DR-tree [5,10)은 첫번째 검색되는 a3DR-tree 이므로, 단순하게 aggr_sum을 사용하여 영역 합을 구한다. rd_4, rd_5, rd_6가 *Q3*를 만족하므로, a3DR-tree [5,10)에서의 영역 합은 R_6=(6,0)과 R_{12}=(15,11)을 합하여 21이 된다. 비록 a3DR-tree 에 중복되는 포인터를 가지고 있는 레코드들 R_5=(7,7)과 R_7=(11,11)이 존재하지만, *Q3*를 처리하는 경우에는 고려할 필요가 없다. 비슷한 방법으로, a3DR-tree [10,16)에서의 영역 합은 18이 된다. 왜냐 하면, rd_4, rd_6, rd_7, rd_8이 *Q3*를 만족하기 때문에, 영역 합은 R_7=(15,15)과 R_{12}=(18,0)으로부터 18 (= (15-15)+(18-0)) 이 된다. 이것은 이전에 rd_4나 rd_6와 같이 여러 개의 a3DR-tree들에 속하는 레코드들의 중복해서 영역 합에 포함되는 것을 막을 수 있다. 최종적으로, *Q3*에 대한 영역 합은 39 (= 21+18)가 된다.

레코드의 삽입과 삭제는 R-tree의 삽입과 삭제 알고리즘을 그대로 사용한다. 차이점은 삽입과 삭제 연산을 수행할 때, 각 엔트리들이 aggr_sum과 dup_aggr_sum을 유지하는 것이다. 알고리즘을 간단하게 기술하기 위해서, 오버플로우(overflow)와 언더플로우(underflow)의 경우는 생략하였다. 그림 6은 APART의 레코드 삽입 알고리즘을 보여준다. *ChooseLeaf*와 *AdjustTree* 알고리즘은 R-tree에서 동작하는 것과 같은 방법으로 동작한다. 만약 하나의 레코드 rd=<s, [t_i, t_j), *value*>가 여러 개의 a3DR-tree들과 교차하는 경우에, 교차하는 첫번째 a3DR-tree의 말단 노드 엔트리의 aggr_sum = *value*가 되고, dup_aggr_sum = 0이 된다. 나머지 교차하는 a3DR-tree들의 말단 노드 엔트리는 aggr_sum과 dup_aggr_sum 이 모두 *value*가 된다.

```
function Delete_Record (record rd=<s, [ti,tj), value>)
begin
1.    for each a3DR-tree ti which intersects rd do
2.        Delete (ti, rd, true)
end

function Delete (a3DR-tree t, record rd)
begin
1.    call FindLeaf to locate a leaf node L containing rd
2.    /* let e be a leaf node entry indicating rd which is to be removed from L */
3.
4.    call CondenseTree to propagate changes upward
5.    /* let ep be an entry which indicates a node Nc on the path from L to the root */
6.        ep.aggr_sum = 0    and    ep.dup_aggr_sum = 0 /* 0          */
7.        for each entry ei which belongs to Nc do
8.            ep.aggr_sum += ei.aggr_sum    and    ep.dup_aggr_sum += ei.dup_aggr_sum
End
```

그림 7 APART의 레코드 삭제 알고리즘

그림 7에 나타난 것처럼, 삭제 알고리즘도 삽입 알고리즘과 유사하게 동작한다. 삭제될 레코드가 여러 개의 a3DR-tree에 속하는 경우에, 삭제 알고리즘은 해당 레코드와 관련된 a3DR-tree들의 모든 말단 노드 엔트리들을 제거해야 한다. *FindLeaf*와 *CondenseTree* 알고리즘은 R-tree에 있는 것을 그대로 사용하였다.

2.3 APART의 분할되는 최적의 시간 영역 구하는 방법

일반화를 위하여, 공간 영역을 $[0,1)^d$ ($d \geq 2$) 이라고 가정한다. 그림 8은 L과 $\overline{T_q}$의 관계를 보여준다. 이 관계는 탐색해야 하는 a3DR-tree들의 수에 많은 영향을 준다. 편의상 2차원 이상의 공간 영역을 단순하게 1차원으로 표시하였다. 그림 8의 경우에, 주어진 질의를 처리하기 위해서, 2개의 a3DR-tree들이 검색한다.

$\overline{N_{visited}}$는 주어진 질의를 처리하기 위하여 접근하는 a3DR-tree들의 평균 개수를 의미하고, $\overline{S_{visited}}$는 방문해야 하는 a3DR-tree들의 평균적인 면적이라고 하자. 직관적으로, $\overline{N_{visited}}$와 $\overline{S_{visited}}$는 질의의 시간 영역이 하나의 분할된 a3DR-tree의 시간 영역과 일치할 때, 가장 좋은 결과를 제공한다. 즉, $\overline{N_{visited}} = 1$이고, $\overline{S_{visited}} = \overline{T_q}$ 이다.

그림 9(a)와 그림 9(b)는 시간 영역의 길이가 $\overline{T_q}$인

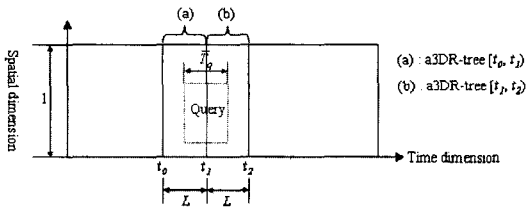
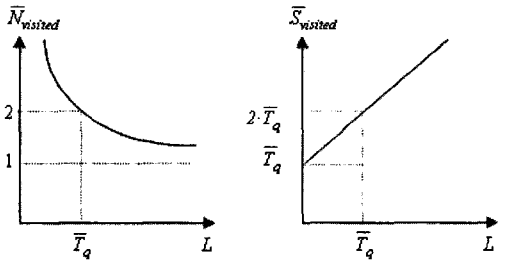


그림 8 L과 $\overline{T_q}$의 관계



(a) $\overline{N_{visited}}$ *vs* L (b) $\overline{S_{visited}}$ *vs* L

그림 9 L에 의해서 결정되는 $\overline{N_{visited}}$과 $\overline{S_{visited}}$

영역 합 질의를 처리하기 위해서 요구되는 $\overline{N_{visited}}$와 $\overline{S_{visited}}$를 L의 함수를 사용하여 보여준다. $\overline{N_{visited}}$는 다

음과 같이 표현된다.

$$\overline{N}_{visited} = \frac{\overline{T}_q}{L} + 1 \quad (2)$$

유사하게, $\overline{S_{visited}}$는 다음과 같이 나타난다.

$$\overline{S}_{visited} = L \cdot \overline{N}_{visited} = \overline{T}_q + L \quad (3)$$

[보조정리1] 식 (2)의 증명

증명: 시간 영역의 길이가 $\overline{T_q}$인 질의 Q가 주어지면, $\overline{N_{visited}}$는 $\left\lfloor \frac{\overline{T_q}}{L} \right\rfloor$ 또는 $\left\lfloor \frac{\overline{T_q}}{L} \right\rfloor + 1$이 된다. $\left\lfloor \frac{\overline{T_q}}{L} \right\rfloor$을 N이라고 가정하고, P_N을 Q가 N개의 a3DR-tree들과 교차할 때의 확률이라고 하자. 그림 10에서 보는 것처럼, P_N은 T가 시간축을 따라서 t_0와 t_1사이를 움직일 때, T가 (b)구간에 속할 확률이다. 그래서,

$$P_N = \frac{N \cdot L - \overline{T}_q}{L} = N - \frac{\overline{T}_q}{L}$$

P_{N+1}은 질의 Q가 (N+1)개의 a3DR-tree들과 교차할 때의 확률이라고 하자. 그러면, P_{N+1} =1- P_N이 된다. 따라서, P_{N+1}은 다음과 같다.

$$P_{N+1} = 1 - P_N = 1 - N + \frac{\overline{T}_q}{L}$$

위에 제시된 두 개의 식들로부터, $\overline{N_{visited}}$는 다음과 같다.

$$\overline{N}_{visited} = N \cdot P_N + (N+1) \cdot P_{N+1} = \frac{\overline{T}_q}{L} + 1$$

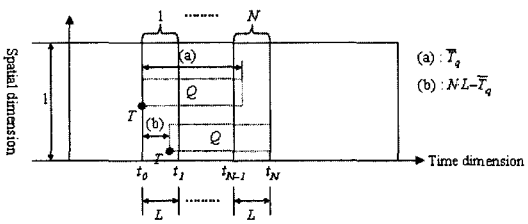


그림 10 Q가 N개의 a3DR-tree와 교차할 확률 P_N

식 (3)은 [보조정리1]로부터 자연스럽게 증명된다. 식 (2)와 식 (3)으로부터 알 수 있는 사실은 $\overline{N_{visited}}$와 $\overline{S_{visited}}$를 모두 최소화 시키는 L을 찾는 것은 불가능하다는 것이다. 그래서, $L = \overline{T_q}$로 사용한다. 이유는 $L = \overline{T_q}$일 때, $\overline{N_{visited}}$ =2가 되고, $\overline{S_{visited}}$ =2 · $\overline{T_q}$가 된다. 이것은 $\overline{N_{visited}}$ =1과 $\overline{S_{visited}} = \overline{T_q}$이 최적화된 결과라는 것을 고려한다면, $L = \overline{T_q}$은 좋은 선택이 될 수 있다.

그림 11은 APART의 색인 크기를 L의 함수를 사용하여 보여준다. $SIZE_{APART}$와 $SIZE_{a3DR\text{-}tree}$는 APART와 a3DR-tree의 색인 크기를 각각 의미한다고 하자. 그러면, $SIZE_{APART}$와 $SIZE_{a3DR\text{-}tree}$의 관계는 다음과 같다.

$$SIZE_{APART} = SIZE_{a3DR\text{-}tree} \cdot \left(\frac{\overline{T}_r}{L} + 1 \right) \quad (4)$$

$SIZE_{APART}$와 $SIZE_{a3DR\text{-}tree}$의 차이는 시간 영역을 분할한 경우에, 레코드가 여러 개의 분할된 a3DR-tree에 속하기 때문에, 말단 노드 엔트리의 수가 여러 개가 필요하다.

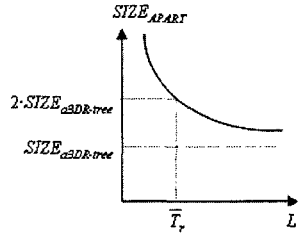


그림 11 $SIZE_{APART}$ vs L

[보조정리2] 식 (4)의 증명

증명: $\overline{N_{pointers}}$는 시간 영역의 길이가 $\overline{T_r}$인 레코드를 APART가 처리하기 위해서 요구되는 평균적인 말단 노드 엔트리의 개수라고 하자. 그러면, $\overline{N_{pointers}}$는 시간 영역의 길이가 $\overline{T_r}$인 질의와 교차하는 a3DR-tree의 개수와 같다. 즉, $\overline{N_{pointers}}$는 $\overline{N_{visiters}}$와 같은데, $\overline{T_r}$ 대신에 $\overline{T_r}$이 사용되었다고 가정하면 $\overline{N_{pointers}}$는 다음과 같다.

$$\overline{N}_{pointers} = \frac{\overline{T}_r}{L} + 1$$

위 식이 의미하는 것은 APART에서는 평균적으로 $\overline{N_{pointers}}$개의 말단 노드 엔트리가 사용된다는 것을의미한다. 대조적으로 a3DR-tree는 1개의 말단 노드 엔트리가 사용된다. 결과적으로, APART의 색인 크기는 분할하지 않은 a3DR-tree보다 $\overline{N_{pointers}}$만큼 크다. 따라서,

$$SIZE_{APART} = SIZE_{a3DR\text{-}tree} \cdot \left(\frac{\overline{T}_r}{L} + 1 \right)$$

그림 11에서 보여지는 것처럼, L이 너무 작으면, $SIZE_{APART}$가 매우 크게 된다. 이를 막기 위해서 L에 최소한의 경계값(threshold value)을 주는 것이 필요하다. 본 논문에서는 L의 최소한의 경계값으로 $\overline{T_r}$을 사용한다. $L \geq \overline{T_r}$이면, $SIZE_{APART} \leq 2 \cdot SIZE_{a3DR\text{-}tree}$이다. 결과적으로, $L = \max(\overline{T_q}, \overline{T_r})$가 된다.

4. 실 험

4.1 실험 환경

시공간 데이타베이스에서는 실험에서 사용할 수 있는 실제 데이타(real data)를 구하기 어렵기 때문에, GSTD 방법[16]과 GNBM[17] 방법을 통해 생성된 데이타를 사용하여 실험하였다. 4.2 실험 결과는 GSTD 방법을 사용하여 얻은 데이타를 사용한 결과이고, 4.3 실험 결과는 GNBM 방법을 사용하여 얻은 결과이다. GSTD 방법은 이동 객체의 움직임을 모방하는데 널리 사용되고[2,3,18], GNBM 방법은 도로와 같은 주어진 경로 위를 움직이는 이동 객체를 표현하는데 사용된다. 이동 객체의 움직임을 묘사하는 데 사용되는 레코드는 3개의 속성(<s, [t_i, t_j), value>)으로 구성된다. GNBM 방법에서는 그림 12에서 보여지는 Tiger/Line 지도[19]를 위에서 움직이는 객체들의 정보를 사용하였다.

그림 12 Tiger/Line 데이타 - Wisconsin 주의 도로 정보

이동 객체들은 모양이나 위치를 시간축을 따라서 임의대로 변경한다. 실험을 위해서 1,000 타임스탬프 동안 10,000 개 이동 객체들의 변화 정보를 수집하였다. $DS_{agility}$는 전체 이동 객체들 중에서, 매 타임스탬프에 모양, 위치, 값(value)를 변경하는 이동 객체의 비율을 가리킨다. 실험에서는 $DS_{agility}$의 값을 1%에서 20%까지 변화시킨다. 예를 들어, $DS_{agility}$ = 5%라면, 매 타임스탬프에 500(=10,000 × 5%)개의 객체가 상태를 변경하고, 500 개의 레코드가 생성된다.

논문에서 구현된 시공간 색인 구조는 모두 R*-tree [13]를 기초로 구현되었다. R*-tree는 제안된 R-tree 계열[13,14,20]중에서 현재까지 가장 성능이 좋다고 알려져 있어서, aHR-tree, a3DR-tree, APART의 구현할 때, R*-tree의 알고리즘을 바탕으로 구현하였다. 색인 구조의 1개의 노드는 1개의 디스크 페이지로 매핑되고, 크기는 1KB이다. 이 때, aHR-tree, a3DR-tree, APART의 팬아웃(fanout)은 각각 36, 31, 27이 된다. aHR-tree에서 루트 노드를 제외한 다른 노드들은 시간 정보를 포함하지 않기 때문에, aHR-tree의 팬아웃의 크기는 a3DR-tree나 APART보다 크다. APART는 엔트리에 dup_aggr_sum과 같은 추가적인 정보를 포함하기 때문에, APART의 팬아웃이 가장 작다.

표 1 실험에 사용된 5개의 질의 워크로드

	Q_T	Q_S
WKLD1	[0, 40)	1% - 20%
WKLD2	[40, 80)	1% - 20%
WKLD3	[80, 120)	1% - 20%
WKLD4	[120, 160)	1% - 20%
WKLD5	[160, 200)	1% - 20%

표 1에 보여지는 것처럼, 5개의 질의 워크로드가 사용되었다. 여기서 Q_T와 Q_S는 각각 질의 워크로드에 있는 질의들의 시간 영역의 길이와 공간 영역의 면적을 가리킨다. 각 워크로드는 표 1에 나타난 분포 정보를 따르는 200개의 질의들로 구성되었다.

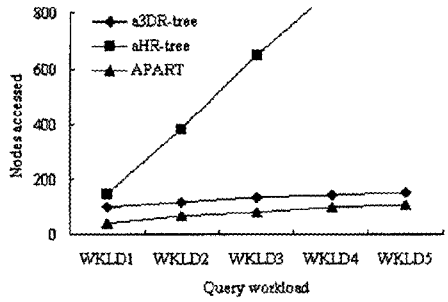
$\overline{T_q}$를 계산하기 위해서, 질의 워크로드에 있는 10% 질의들을 무작위로 선택하여, 이들의 평균을 계산하였다. $\overline{T_r}$은 색인을 만드는 동안에 레코드들을 사용하여 계산되었다. 성능 비교를 위하여, 질의들을 처리하는 동안에 발생하는 노드 접근 회수를 조사하였다.

4.2 GSTD 데이타를 이용한 실험 결과

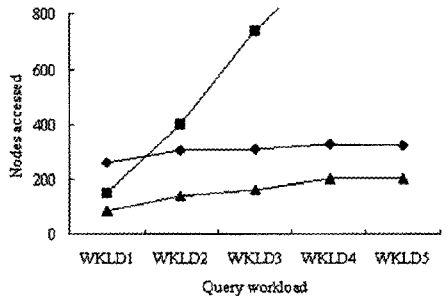
그림 13은 $DS_{agility}$=10%일 때, 다양한 질의 워크로드에 대한 실험 결과를 보여준다. 다른 $DS_{agility}$에 대한 실험 결과는 $DS_{agility}$=10%일 때의 결과와 유사하기 때문에, 생략되었다. 평균적으로, a3DR-tree와 aHR-tree의 질의 비용은 APART의 질의 비용보다 각각 2.07배와 4.26배 크다. aHR-tree의 질의 비용은 질의 워크로드의 시간 영역이 길수록 커진다. 이것은 접근해야 하는 R-tree의 수가 증가하기 때문이다. 반대로, a3DR-tree는 질의 워크로드의 시간 영역의 길이가 짧은 경우에 질의 결과가 aHR-tree와 APART보다 나쁘다. WKLD1일 때, 이것을 확인할 수 있다. APART는 질의 워크로드를 기반으로 시간 영역을 분할했기 때문에, aHR-tree와 a3DR-tree보다 좋은 성능을 나타내었다.

그림 14는 WKLD3에 대하여 $DS_{agility}$를 1%에서 20%까지 변화시키면서 실험을 하였다. 평균적으로, a3DR-tree와 aHR-tree의 질의 비용은 각각 APART의 질의 비용보다 4.53배, 6.03배나 크다. 특히, aHR-tree의 경우는 $DS_{agility}$에 관계없이 가장 좋지 않은 성능을 보여주었다. APART는 다양한 $DS_{agility}$에 대해서도 가장 좋은 성능을 보여주었다.

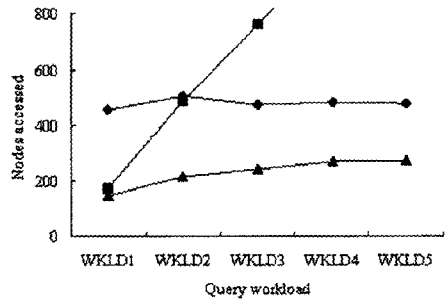
다음의 2가지 조건(WKLD3와 $DS_{agility}$=10%)에 대해서 각 색인 구조의 크기를 조사하였다. 그림 15(a)는 WKLD3에 대하여, $DS_{agility}$를 1%에서부터 20%까지 변경하면서, 색인 구조의 크기를 조사하였다. 특정한 질의 워크로드를 기준으로, $DS_{agility}$를 변경하면서 색인 구조



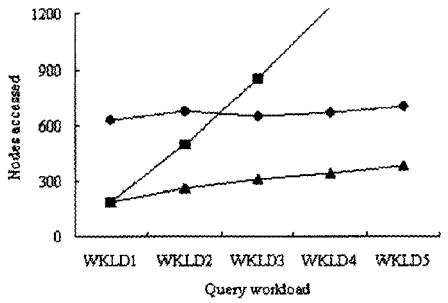
(a) Qs = 1%



(b) Qs = 5%

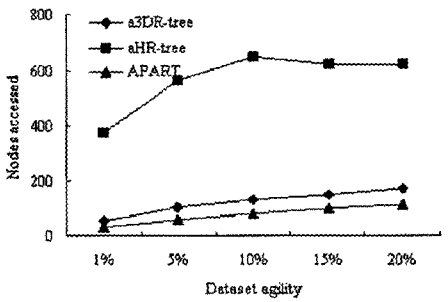


(c) Qs = 10%

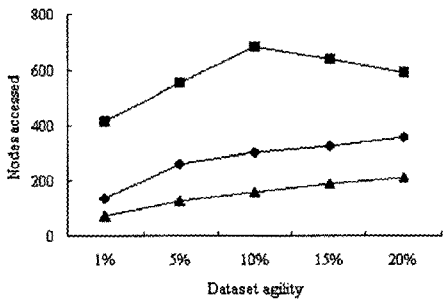


(d) Qs = 20%

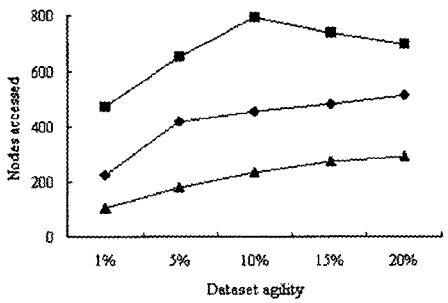
그림 13 다양한 질의 워크로드에 대한 실험 결과



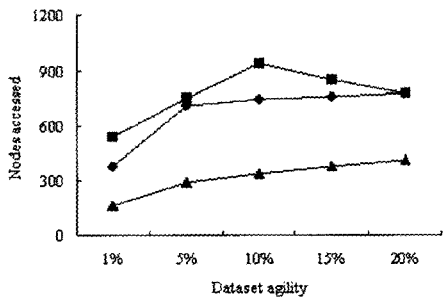
(a) Qs = 1%



(b) Qs = 5%

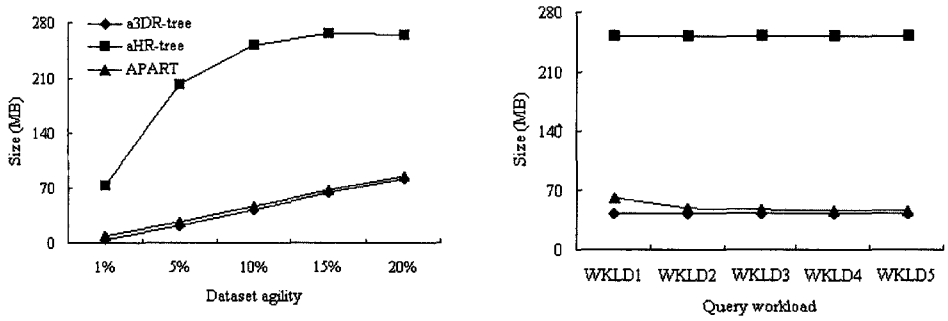


(c) Qs = 10%



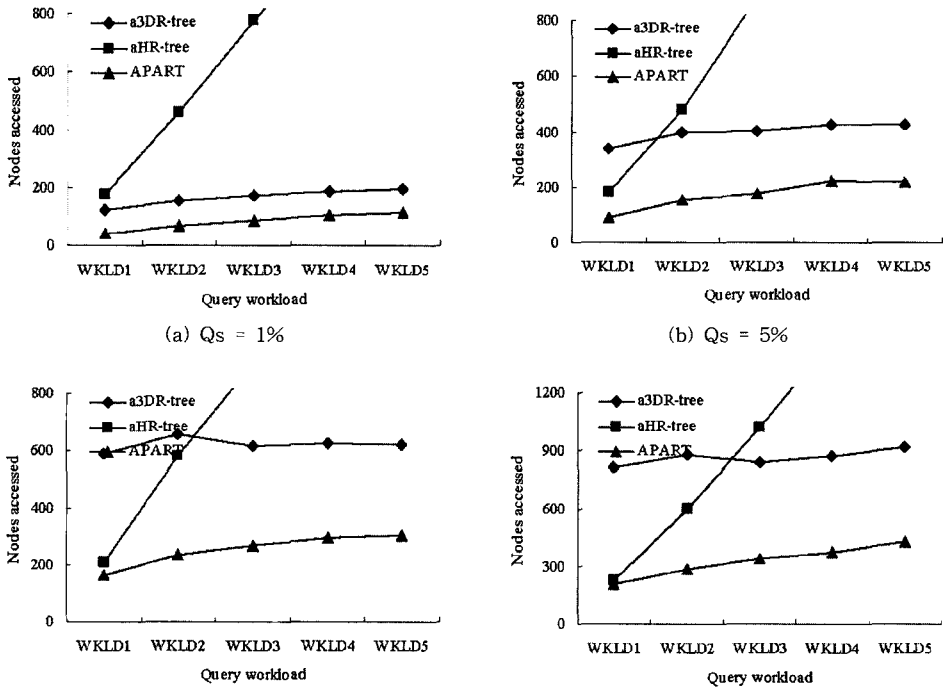
(d) Qs = 20%

그림 14 다양한 $DS_{agility}$에 대한 실험 결과



(a) WKLD3 (b) $DS_{agility}$ = 10%

그림 15 색인 크기 비교



(a) Qs = 1% (b) Qs = 5%

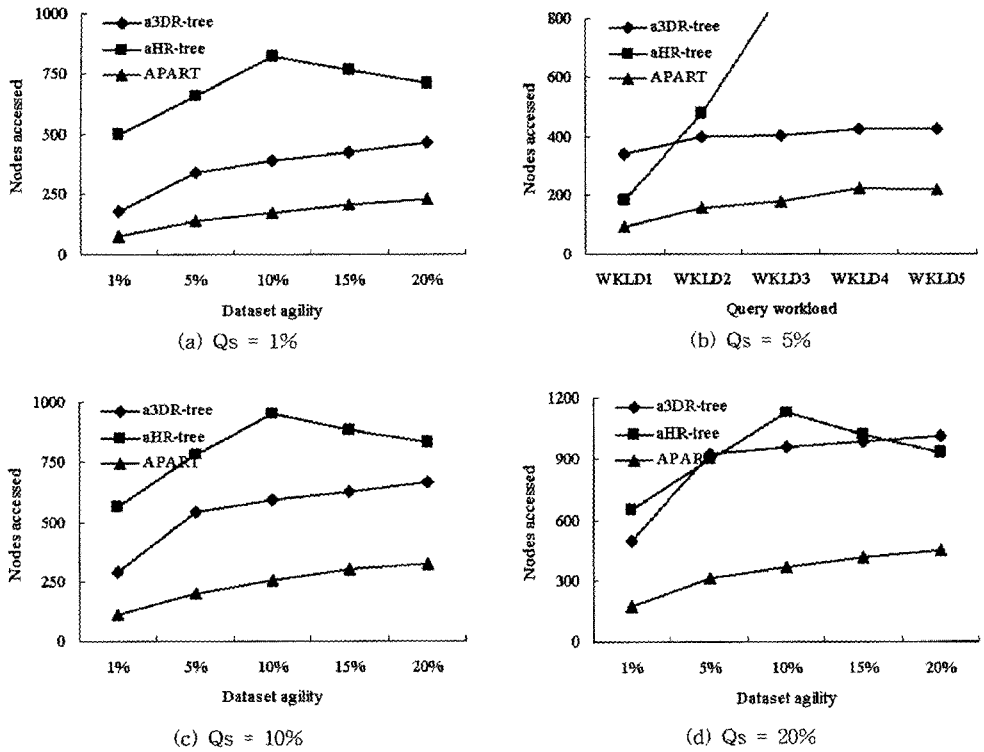
(c) Qs = 10% (d) Qs = 20%

그림 16 다양한 질의 워크로드에 대한 실험 결과

의 크기를 조사하는 이유는 질의 워크로드가 APART의 색인 크기에 영향을 주기 때문이다. 예상대로, a3DR-tree가 가장 작고, APART의 크기는 a3DR-tree와 차이가 크지 않았다. 그러나, aHR-tree는 인접한 R-tree 사이에 중복되는 엔트리들 때문에, 크기가 a3DR-tree보다 최소 3배부터 15배까지 크게 나타났다. 그림 15(b)는 $DS_{agility}$ = 10%일 때, 질의 워크로드를 변경하면서, 색인 크기를 조사하였다. APART를 제외한 aHR-tree와 a3DR-tree의 크기는 변하지 않는다. 이것은 aHR-tree와 a3DR-tree는 질의 워크로드 정보를 사용하지 않기 때문이다. 질의 워크로드의 시간 영역의 길이가 커질수록, APART의 크기는 감소하였다. 이것은 식 (4)를 통하여 알 수 있다. 즉, L이 길어지면, 하나의 레코드가 여러 개의 a3DR-tree에 속할 확률이 줄어든다. 모든 경우에서, aHR-tree의 색인 크기가 가장 크고, a3DR-tree의 색인 크기가 가장 작다.

4.3 GNBM 데이타를 이용한 실험 결과

그림 16은 $DS_{agility}$=10%일 때, 다양한 질의 워크로드에 대한 실험 결과를 보여준다. 다른 $DS_{agility}$에 대한 실험 결과는 $DS_{agility}$=10%일 때의 결과와 유사하기 때문



(a) Qs = 1% (b) Qs = 5%

(c) Qs = 10% (d) Qs = 20%

그림 17 다양한 $DS_{agility}$에 대한 실험 결과

에, 생략되었다. 평균적으로, a3DR-tree와 aHR-tree의 질의 비용은 APART의 질의 비용보다 각각 3.04배와 4.64배 크다. aHR-tree의 질의 비용과 a3DR-tree는 질의 비용이 APART보다 큰 이유는 시간 분할 영역이 질의 영역보다 너무 작거나 크기 때문이다.

그림 17은 WKLD3에 대하여 $DS_{agility}$를 1%에서 20%까지 변화시키면서 실험한 결과를 보여준다. 평균적으로, a3DR-tree와 aHR-tree의 질의 비용은 각각 APART의 질의 비용보다 5.35배, 6.57배나 크다. APART는 여전히 다양한 $DS_{agility}$에 대해서도 가장 좋은 성능을 보여주었다.

이동 객체가 도로와 같이 제한된 영역만을 이동하기 때문에, 같은 질의 워크로드에 대하여 GSTD보다는 GNBM에서 전체적으로 질의 비용이 증가하는 경향을 보였다. 색인의 크기는 색인에 사용된 이동 객체의 수와 $DS_{agility}$에 따라 결정되기 때문에, GSTD의 결과와 크게 다르지 않았다.

4. 결 론

모바일, 인공위성, 무선통신 기술 발달과 함께, 영역 합 질의들을 사용하는 많은 실생활 응용 프로그램들이 등장하고 있다. 다른 집합적 색인 구조와 달리, APART는 다양한 질의 워크로드와 데이타들에 대해서 좋은 성능을 제시할 수 있는 질의 기반의 동적 분할 기법을 사용한다. 분할에 의해서 생겨나는 레코드들의 중복 계산을 피하기 위해서, APART는 자식 노드에 대한 영역 합과 중복되는 레코드들의 영역 합도 함께 저장한다. 본 논문에서는 다양한 실험을 통하여, APART가 기존 색인 기법들보다 우수하다는 것을 보여주었다.

참 조 문 헌

[1] M. Nascimento and J. Silva: Towards historical R-trees. In *Proceedings of the 1998 ACM symposium on Applied Computing*, pages 226-234, 1998.

[2] S. Saltenis, C. Jensen, S. Leutenegger, and M. Lopez: Indexing the Positions of Continuously Moving Objects. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 331-342, 2000.

[3] Y. Tao and D. Papadias: The MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *Proceedings of the International Conference on Very Large Data Bases*, pages 431-440, 2001.

[4] M. Vazirgiannis, Y. Theodoridis, and T. Sellis:

Spatio-Temporal Composition and Indexing for Large Multimedia Applications. Multimedia Systems 6(4), pages 284-298, 1998.
[5] Z. Song and N. Roussopoulos: K-Nearest Neighbor Search for Moving Query Point. In *Proceedings of the Symposium on Large Spatial Databases*, pages 79-96, 2001.
[6] Y. Tao and D. Papadias: Time-Parameterized Queries in Spatio-Temporal Databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 334-345, 2002.
[7] Y. Tao, D. Papadias, and Q. Shen: Continuous Nearest Neighbor Search. In *Proceedings of the International Conference on Very Large Data Bases*, pages 287-298, 2002.
[8] D. Pfoser, C. S. Jensen, and Y. Theodoridis: Novel Approaches in Query Processing for Moving Object Trajectories. In *Proceedings of the International Conference on Very Large Data Bases*, pages 395-406, 2000.
[9] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. In *Proceedings of IEEE International Conference on Data Engineering*, pages 152-159, 1996.
[10] V. Harinarayan, A. Rajaraman, and J. Ullman: Implementing Data Cubes Efficiently. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 205-216, 1996.
[11] I. Lazaridis and S. Mehrotra: Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 401-412, 2001.
[12] D. Papadias, Y. Tao, P. Kalnis, and J. Zhang: Indexing Spatio-Temporal Data Warehouses. In *Proceedings of IEEE International Conference on Data Engineering*, pages 166-175, 2002.
[13] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger: The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 322-331, 1990.
[14] A. Guttman: R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 47-57, 1984.
[15] W. Choi, B. Moon, and S. Lee: Adaptive cell-based index for moving objects. *Data & Knowledge Engineering*, 48(1): 75-101, 2004.
[16] Y. Theodoridis, J. Silva, and M. Nascimento: On the Generation of Spatio-temporal Datasets. In *Proceedings of the Symposium on Large Spatial Databases*, pages 147-164, 1999.
[17] T. Brinkhoff: A Framework for Generating Network-Based Moving Objects. *GeoInformatica* 6(2): 153-180, 2002.
[18] M. Nascimento, J. Silva, and Y. Theodoridis: Evaluation of Access Structures for Discretely Moving Points. In *Proceedings of the Workshop on Spatio-Temporal Database Management*, pages 171-188, 1999.
[19] http://www.esri.com/data/download/census2000_tigerline/index.html.
[20] T. Sellis, N. Roussopoulos, and C. Faloutsos: The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the International Conference on Very Large Data Bases*, pages 507-518, 1987.

조 형 주
1997년 서울대학교 컴퓨터공학과 졸업. 1999년 서울대학교 컴퓨터공학과 석사졸업. 1999년~현재 한국과학기술원 전자전산학과 전산학과 박사과정 재학중. 2004년~현재 LG전자 DM 연구소 연구원. 관심분야는 시간/공간/시공간 데이타베이스 질의처리, 인덱싱, 질의최적화

최 용 진
1997년 성균관대학교 정보공학과(학사) 1999년 한국과학기술원 전산학과(석사) 2003년 8월 한국과학기술원 전자전산학과(박사). 현재 한국과학기술원 박사후과정 관심분야 공간 데이타베이스, GIS, 시공간 데이타베이스, LBS

민 준 기
1995년 숭실대학교 전자계산학과(학사) 1997년 한국과학기술원 전산학과(석사) 2002년 8월 한국과학기술원 전산학전공(박사). 2003년 8월 한국과학기술원 박사후과정. 현재 한국기술교육대학교 인터넷미디어학부 교수. 관심분야는 데이타베이스, XML, 시공간 DB, OLAP

정 진 완
1973년 서울대학교 공과대학 전기공학과(학사). 1983년 University of Michigan 컴퓨터공학과(박사). 1983년~1993년 미국 GM 연구소 전산과학과 선임연구원 및 책임연구원. 1993년~현재 한국과학기술원 전산학과 부교수 및 교수. 관심분야는 XML, 멀티미디어 데이타베이스, GIS, 웹 정보검색, 객체지향 데이타베이스