공학박사 학위논문

RFID 태그 객체를 위한 효율적인 구간 색인 구조

지도교수 홍 봉 희

2006년 2월

부 산 대 학 교 대 학 원

컴 퓨 터 공 학 과

반 재 훈

반재훈의 공학박사 학위 논문을 인준함

2005년 12월 15일

주 심 류 광 렬 ㊞

부 심 홍 봉 희 ㊞

위 원 채 흥 석 ㊞

위 원 탁 성 우 ㊞

위 원 김 일 곤 ㊞

목 차

그림 목차

알고리즘 목차

표 목차

1. 서론

RFID(Radio Frequency IDentification)는 각종 물품에 소형 칩인 태그(Tag)를 부착하고 사물의 정보와 주변 환경 정보를 판독·해독기능이 있는 판독기(Reader)를 통해 인식하여 무선주파수로 전송·처리하는 비접촉식 인식시스템이다. RFID는 높은 인식률, 비접촉형 인식매체, 도달거리, 다른 통신망과의 연계 및 통신 가능성 등의 확장성으로 인해 항만/물류/유통, 군사, 식품/안전 등 비즈니스 영역에 킬러 애플리케이션으로서 막대한 파급 효과를 끼칠 전망이다[1][2][3].

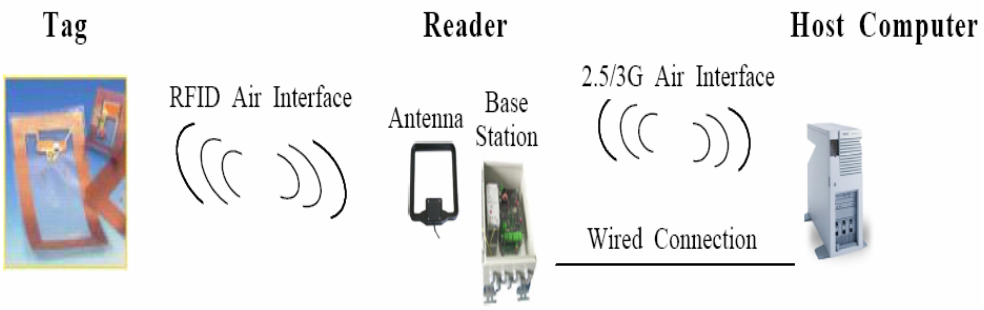


그림 1 RFID(Radio Frequency IDentification) 시스템

예를 들어, 항만 물류 시스템에서 RFID 시스템을 도입할 경우, 항만에서 처리되는 컨테이너의 위치를 빠르게 검색하고 접근하여 효율적인 수송 및 환적 업무의 수행이 가능하게 된다. RFID 시스템 도입 전 전통적인 항만 물류 시스템에서는 외부 차량이 항만 입구를 통과하면 사람이 일일이 컨테이너의 물품을 검사한 뒤 지정된 위치로 수송하여 선적을 기다리게 된다. 또한 운송 중인 컨테이너와 제품의 위치를 추적하기 위해서는 사람의 손을 거쳐

처리하거나 자동으로 처리하더라도 위치를 등록하는 등의 추가 업무가 필요하게 된다. 그러나 RFID 시스템을 도입하면 자동으로 수송해야 할 컨테이너의 위치를 제공받고, 차량의 시동을 멈추게 하는 지연 없이 컨테이너가 위치한 곳으로 이동할 수 있다. 또한 운송 중인 컨테이너와 제품의 위치 추적이 가능하며, 중앙 관제 시스템에서 실시간으로 제품의 이동을 감시할 수 있다.

그러나 컨테이너와 제품에 RFID 태그를 장착할 경우 관리하고자 하는 객체 수가 방대하게 증가하기 때문에, RFID 시스템뿐만 아니라 대용량의 위치 데이터를 효율적으로 관리하는 데이터베이스가 필요하게 된다. 이러한 RFID 태그를 위한 데이터베이스에서는 태그를 장착한 객체의 위치를 추적하고 이동 경로를 감시하기 위해 위치 기반 질의를 지원해야 하므로, 태그의 궤적을 위한 데이터 모델과 색인 구축이 필수적이다.

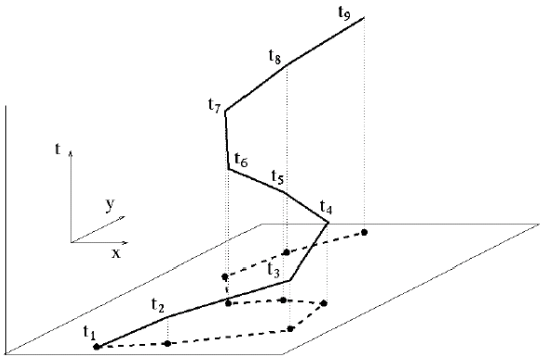


그림 2 이동 객체의 궤적 모델링

GPS 장치를 탑재한 차량과 같은 이동 객체(moving object)는 속도나 방향이 바뀔 때마다 3차원 시공간 상의 점으로 위치를 보고한다. 그리고 보고된 두 점을 선분으로 연결하여 이동 경로를 나타내며, 이러한 경로들을 모아 다중

선(polyline)인 궤적(trajectory)을 표현한다[36][38][46][60]. 즉, 그림 2와 같이 객체가 시간 $[t_i,t_j]$ 동안 위치 (x_i,y_i)에 있고, 시간 $[t_j,t_k]$ 동안 위치 (x_j,y_j)에 있다면, 3차원 시공간에서 선분 $\overline{[(x_i, y_i, t_i),(x_i, y_i, t_j))}$ 와 선분 $\overline{[(x_j, y_j, t_j),(x_j, y_j, t_k))}$으로 궤적을 표현할 수 있으며, 3차원 R-Tree를 사용하여 색인을 구축한다. 태그는 이동 객체와 유사하게 시간에 연속적으로 이동하므로 태그의 궤적을 추적하기 위해서 이동 객체를 위한 시공간 색인을 적용할 수 있다.

태그는 전략적으로 중요한 지점에 설치된 판독기의 인식 영역 내에서 위치를 보고하기 때문에, 판독기의 인식 영역의 크기와 수에 따라 위치의 정확성이 결정된다. 판독기의 위치를 기반으로 태그가 보고된 두 시점을 선분으로 연결하여 궤적을 표현할 경우에 위치 오차가 크며 판독기에 태그가 위치한 시간을 표현하지 못하는 문제가 발생한다. 예를 들어서, 50m의 인식 영역을 가지는 900MHz 판독기를 그림 3와 같이 설치하고 두 판독기에 보고된 태그의 시공간 위치를 연결하여 궤적을 표현하면 실제 차량이 이동한 궤적과는 상당한 오차가 발생한다. 또한 RFID 환경에서 제일 빈번한 질의인 “어느 판독기에 어떤 태그가 얼마나 오랫동안 위치했는가?”를 처리할 수 없다. 따라서 RFID 태그를 장착한 객체의 위치 추적 서비스를 제공하기 위해서는 새로운 과거 이력 표현 방법으로서 시간과 순서를 기반으로 하는 데이터 모델과 색인 구조가 필요하다.

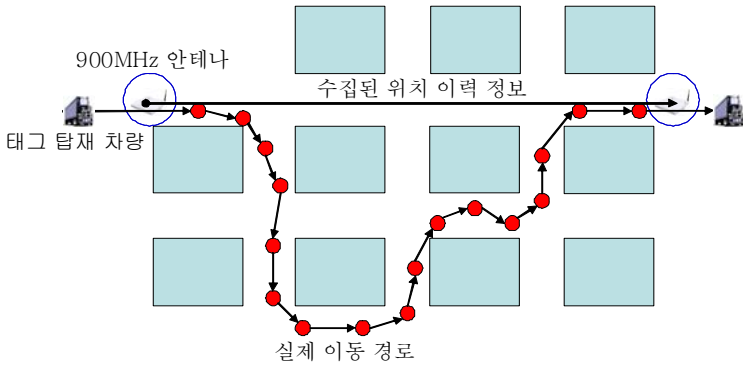


그림 3 태그와 이동 객체의 이동 경로의 비교

태그의 위치를 추적하기 위해서는 이벤트 기반으로 태그의 궤적을 표현해야 한다. 태그가 판독기의 인식 영역에 들어갈 때 Enter 이벤트가 발생하고 인식 영역을 빠져 나올 때 Leave 이벤트가 발생한다[1]. 이러한 이벤트 발생 시 태그가 보고하는 두 개의 시공간 위치 즉, Enter 이벤트 시 보고하는 시공간 위치와 Leave 이벤트 시 보고하는 시공간 위치를 연결한 선분으로 궤적을 표현하면 그림 4와 같이 태그의 위치를 추적할 수 있다.

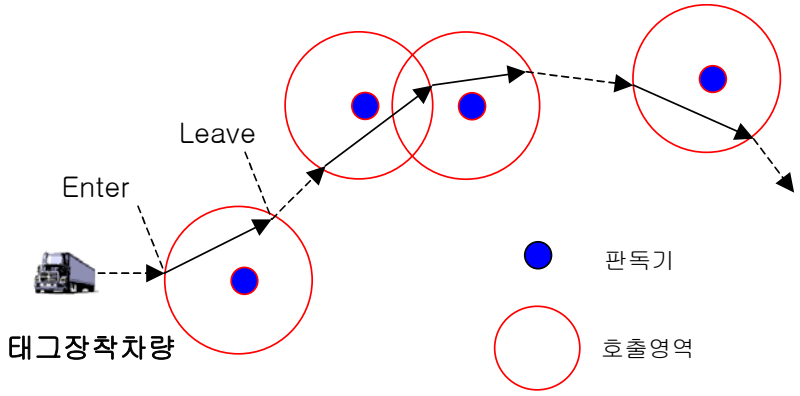


그림 4 태그의 이동에 따른 이벤트 발생

그러나 이러한 궤적의 모델링 기법을 사용하는 경우에 판독기의 인식영역에 들어와 머무는 태그를 찾을 수 없는 문제가 발생한다. 만약 태그가 판독기에 머무는 경우에 아직 Leave 이벤트가 발생하지 않았으므로 태그가 판독기에 들어갈 때인 Enter 이벤트 발생시에 보고하는 시공간 점으로만 궤적이 구성된다. 이러한 시공간 점은 태그가 판독기에 머문다는 정보를 표현할 수 없으므로 판독기에 머무는 태그를 찾을 수 없는 문제가 발생한다. 따라서 판독기에 머무는 태그를 표현할 수 있는 새로운 데이터 모델과 색인 구조가 필요하다.

RFID 시스템의 응용에서 사용되는 질의는 객체 식별자와 위치, 그리고 시간을 인수로 사용한다. 물류 시스템을 예로 들면, “외부 A 차량이 수송할 화물이 어디에 있는가?”라는 질의는 객체 식별자와 현재 시간이 주어질 때, 객체의 위치를 찾는 질의이다. “외부 A 차량이 2시에서 3시 사이에 수송한 화물이 어디에 있었는가”와 같이 객체 식별자와 과거 시간이 주어진 질의도 사용된다. 위치와 시간이 주어지는 질의는 “3번 레인에서 2시에서 3시 사이에 존재했던 컨테이너는?”과 같이 공간 영역에 포함되는 객체 식별자를 찾는 질의이다. 다른 중요한 질의로는 “오염 지역을 경유한 선박의 화물의 위치는?”과 같이 특정 객체의 과거 이력을 찾는 질의가 있다. 위의 질의에서 처리되는 데이터는 공간과 시간을 포함하는 다차원 대용량 데이터이며, 과거 이력 및 현재 상태의 검색을 지원해야 한다. 그러나 기존 이동 객체를 위한 시공간 색인은 현재 위치 질의와 태그 식별자가 주어지는 질의를 처리할 수 없는 문제점이 있다. 따라서, 효율적인 질의 처리를 위해서는 RFID 태그를 위한 새로운 데이터 모델과 색인 구조가 필요하다.

이 논문에서는 위와 같은 문제를 해결하기 위하여 RFID 태그의 궤적을 위한 구간 데이터 모델을 정의한다. 이 모델에서는 태그의 궤적을 판독기에 들어올 때 생성되는 시간에 종속적인 선분인 동적 구간(dynamic interval)과 판독기에 나올 때 생성되는 시간에 고정적인 정적 구간(static interval)으로 표현한다. 따라서 판독기에 머무는 태그의 궤적은 시간에 종속적인 동적 구간으로 표현되므로 이러한 태그를 찾을 수 있게 한다.

또한 태그의 추적을 위한 질의를 분류한다. 분류된 질의는 태그의 위치를 찾는 Find 질의, 판독기에 위치한 태그를 찾는 Look 질의, 특정 태그와 같이 위치한 태그를 찾는 With 질의, 태그의 이동 경로를 찾는 History 질의이다. 이 논문에서는 이러한 질의를 효율적으로 처리하기 위해 질의의 매개 변수로 태그의 식별자가 들어간다는 특성을 고려하여 객체 식별자를 색인의 도메인으로 추가한다.

또한 이 논문에서 제시한 태그의 구간 데이터 모델에 적합한 R-tree 기반 색인 구조인 IR-tree(Interval R-tree)를 제시하며 효율적인 질의처리를 위해 시간에 종속적인 동적 구간의 특성을 고려한 새로운 삽입 및 분할 알고리즘을 제안한다. 마지막으로 다양한 데이터 집합에서 제안된 색인과 기존 알고리즘을 사용하는 색인과의 성능비교를 통하여 색인의 우수성을 입증한다.

이 논문에서 제시하는 태그의 궤적을 구간 데이터로 모델링하여 IR-tree를 구성하는 경우에 다음과 같은 장점이 있다. 첫째, 판독기에 머무는 태그를 검색하는 질의 처리를 수행할 수 있다. 둘째, 동적 구간은 시간에 종속적이므로 이러한 특징을 고려한 삽입, 분할 알고리즘을 사용하면 색인을 구성하는 노드의 면적과 겹침이 줄어들어 질의의 수행 속도를 향상 시킬 수 있다.

마지막으로 태그를 장착한 객체의 위치 추적이나 객체의 현재의 상태를 감시하는 자동화 생산(automated manufacturing), 재고 관리(inventory tracking), 공급망 관리(supply chain management) 등과 같은 유비쿼터스 응용분야에서 태그의 과거 이력정보인 궤적과 현재 위치를 탐색할 수 있다.

이 논문의 구성은 다음과 같다. 2장에서는 관련연구를 기술하며 3장에서는 대상환경 및 태그의 궤적 표현으로 인해 발생되는 문제를 정의한다. 4장에서는 RFID 시스템에서 태그의 위치 추적을 위해 사용되는 질의를 분류하며 태그의 궤적을 표현하기 위한 구간 데이터 모델을 정의한다. 5장에서는 정의된 구간을 이용한 IR-tree의 데이터 구조를 제시하며 질의 수행 방법과 질의의 효율적인 처리를 위한 새로운 삽입, 분할 알고리즘을 제시한다. 6장에서는 제안된 IR-tree를 구현하고 기존 삽입, 분할 알고리즘을 사용하는 색인과의 성능 비교를 통하여 그 우수성을 입증한다. 마지막으로 7장에서는 결론 및 향후 연구를 기술한다.

2. 관련 연구

이 장에서는 태그의 이동을 기존의 이동 객체의 데이터 모델에 적용했을 시에 문제점을 기술하기 위해서, 이동 객체의 모델링에 관한 연구를 먼저 기술한다. 다음으로 이동 객체와 관련된 질의 종류와 색인 구조에 관한 연구를 기술한다. 색인 구조는 구간 데이터를 위한 색인 구조와 이동 객체를 위한 색인 구조로 구분하여 기술한다.

2.1 이동 객체 모델링에 관한 연구

이동 객체의 데이터 모델은 이동 객체의 현재 및 미래 위치와 관련된 MOST(Moving Object Spatio-Temporal)[55][62]와 이동 궤적을 표현하는 시공간 데이터 모델(Spatio-Temporal Data Model)[20][21]로 나눌 수 있다.

MOST 모델은 시간 함수를 이용하여 이동 객체의 현재 및 미래 위치를 표현한다. 그래서 매번 반복되는 데이터베이스 갱신으로 인한 성능 저하의 문제점을 해결한다. 즉, 명시적 변경 없이 질의 시간에 따라 변화하는 동적 속성(dynamic attributes)을 이용하여 미래 위치를 계산하고 질의를 처리한다. 그러나 이 모델은 이동을 기술하는 정확한 시간 함수를 찾기가 어렵다는 문제점이 있다. 태그의 경우 속도나 방향 정보를 획득할 수 없기 때문에 MOST 모델은 적합하지 않다.

시공간 데이터 모델은 2차원 공간과 시간 차원을 동시에 고려하며 이동 객체를 시간이 지남에 따라 위치가 변화하는 3차원 시공간 객체로 정의한다.

이 모델은 연속적 모델과 이산적 모델로 나뉘어진다. 연속적 모델은 이동 객체를 무한개의 점 집합을 가진 곡선으로 표현하며, 이산적 모델은 이동 객체를 유한개의 점 집합을 가진 다중선(polyline)으로 표현한다. 이동 객체 데이터베이스에 위치 정보를 저장하기 위해서는 그림 5와 같이 객체의 궤적을 3차원 선분(line segment)들의 집합으로 정의하는 이산적 모델을 이용한다. 시공간 데이터 모델을 사용하는 색인 구조로서 R-Tree 계열의 색인은 3차원 선분을 MBB(Minimum Bounding Box)형태로 저장한다.

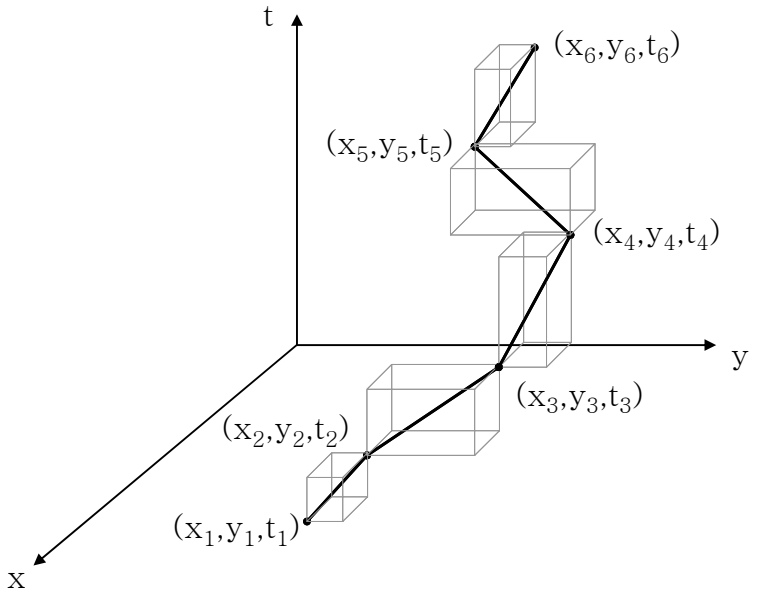


그림 5 이산적 모델

2.2 이동 객체 질의에 관한 연구

이동 객체에 관한 질의는 크게 좌표 기반 질의와 궤적 기반 질의로 나뉘

어진다[46]. 좌표 기반 질의는 3차원 시공간에서 점 질의, 영역 질의, 타임 슬라이스(time slice) 질의 등으로 분류된다. 점 질의는 주어진 시공간 상의 점과 동일한 점 객체를 찾는 질의이며, 영역 질의는 주어진 공간 상의 영역과 시간 구간에 포함되는 모든 객체를 찾는 질의이다. 타임 슬라이스 질의는 영역 질의의 특별한 경우로서, 특정한 시간에 주어진 공간 윈도우를 지나가는 이동 객체를 찾는 질의이다.

궤적 기반 질의는 enter, leave, cross, bypass와 같은 위상 질의(topological query)와 움직인 거리, 속도, 방향과 같은 항해 질의(navigational query)로 분류된다. 또한 궤적을 선택하는 과정과 선택된 궤적을 추출하는 두 단계로 이루어진 복합 질의가 있다. 복합 질의는 “오늘 오전 7시부터 8시까지 신선대 앞을 지나간 컨테이너 차량들이 오전 10시까지 이동한 궤적을 찾아라” 와 같이 시공간 도메인의 영역 질의와 궤적 질의가 복합된 질의이다. 복합 질의에서 궤적을 선택하는 방법은 궤적 식별자(trajectory identifier)를 사용하는 방법, 시공간 영역 질의에서 결과로 나온 궤적 선분을 지정하는 방법, 위상 질의를 사용하는 방법, 유도된 정보를 사용하는 방법 등이 있다.

2.3 구간 데이터를 위한 색인 구조에 관한 연구

구간(interval)은 [l, r](l≤r)인 형태를 가지며 어떤 속성의 기간(duration)을 나타내는 순서쌍이다. 지금까지 연구된 구간 데이터를 위한 색인 구조는 크게 메인 메모리 거주 색인 구조와 디스크 기반 색인 구조로 구분된다.

메인 메모리 거주 색인 구조는 모두 이진 검색 트리 계열과 유사한 구조

를 가지며 Segment Tree[5], Interval Tree[6], Priority Search Tree[7]등이 있다. 이러한 색인의 데이터 구조는 이진 검색 트리 구조이기 때문에 모든 데이터는 메모리에 거주하게 된다. 따라서 디스크 페이지로 저장할 수 없는 문제점이 있다. 그리고 1차원 구간 데이터를 위한 색인 구조이고 backbone tree를 먼저 생성한 후 데이터를 삽입한다. 그래서 다차원 구간 데이터를 저장할 수 없으며, 동적인 환경에서는 사용할 수 없는 문제점이 있다.

디스크 기반 색인 구조로서 대표적인 색인은 R-tree 계열이 있다[13][26]. R-tree[26]는 데이터 분할 방법의 대표적인 공간 색인으로서 공간 객체를 최소 경계 사각형(Minimum Bounding Rectangle)으로 표현하며 단말 노드에 모든 데이터를 저장한다. 색인에 데이터를 삽입하기 위한 ChooseSubtree알고리즘은 최소 영역 확장 정책(Least Area Enlargement Policy)을 사용하여 삽입할 노드를 선택한다. 또한 노드 오버플로우 시에 분할되는 2개의 노드가 최소 영역을 갖도록 분할한다.

R*-tree[13]는 R-tree가 삽입 시에 영역만을 고려하는 단점을 보완하기 위하여 겹침(overlap)과 가장자리(margin)를 추가적으로 고려한 색인 구조이다. R*-tree는 삽입을 위해서 단말 노드를 선택할 시에, 최소 겹침 확장 정책(Least Overlap Enlargement Policy)을 사용한다. 그리고 오버플로우가 발생한 노드의 분할은 노드의 가장자리가 최소가 되는 분할 축을 선택하며, 형제 노드와의 겹침 값이 최소가 되도록 엔트리를 분배한다. 또한 재삽입 정책을 사용하여 R-tree의 공간 활용도가 낮은 문제점을 보완하였다.

SR-tree[14]는 메인 메모리 거주 색인 구조인 Segment Tree[5]를 디스크 기반으로 확장한 색인이다. 색인 구조는 R-tree[26]와 동일한 구조를 가지며 기존

의 삽입 정책과 분할 정책을 그대로 사용한다. 차이점은 자식 노드를 완전히 걸치는 긴 구간 데이터가 삽입될 경우, 데이터를 단말 노드에 저장하지 않고 걸쳐지는 노드의 부모 노드에 저장한다는 점이다. SR-tree는 비단말 노드에도 데이터를 저장하므로, 노드의 크기가 루트 노드로 갈수록 커지는 단점이 있다. 그리고 아주 긴 구간 데이터가 삽입될 경우, 선분을 잘라서 다시 삽입해야 하고, 삽입과 분할로 인하여 노드의 크기가 변할 경우 비단말 노드에 저장되어 있는 구간 데이터를 다른 노드로 이동해야 한다. 태그의 경우 현재 위치를 저장하기 위한 방법이 제공되어야 하며 시간에 종속적인 선분으로 현재를 표현하는 경우에 길이가 고정인 선분을 절단하여 저장하는 SR-tree는 적합하지 않다.

2.4 이동 객체의 색인 구조에 관한 연구

이동 객체의 색인에 관련된 연구는 크게 3가지 부분으로 나누어 볼 수가 있다. 첫째, 현재와 미래 위치를 검색하기 위한 색인에 관련된 연구들로서 R-tree, 해쉉, Quad-tree를 기반으로 하는 색인이 있다. 둘째, 시공간 색인에 대한 연구로서 3DR-tree, HR-tree, 2+3 R-tree가 있다. 셋째, 이동 객체의 궤적과 같은 과거 이력 색인에 대한 연구로서 STR-tree, TB-tree가 있다.

2.4.1 현재 또는 미래 위치 검색을 위한 색인 연구

이동 객체 색인에 관한 초기 연구는 대부분 이동 객체의 이동성을 표현하거나 이동 객체의 위치 변경에 따른 색인 변경의 비용을 줄이는 것을 목표

로 하였다. 이동 객체의 현재 위치를 저장하는 색인에 관한 연구에는 갱신 횟수에 관한 연구[33][57]와 현재 및 미래 위치를 예측하는 연구들[53][59]로 나누어 진다.

해쉉 기법을 이용한 이동 객체 색인은 이동 객체의 위치를 2차원 공간 상의 점으로 모델링한다. 관련 연구 [57]은 해쉬 기반 색인 [39]에서 이동 객체가 셀을 벗어나지 않으면 색인 갱신이 없다는 아이디어를 사용한다. 해쉉 함수를 사용하여 빠른 검색이 가능하나, 이동 객체의 분포에 따른 노드의 오버플로우 문제를 해결해야 한다. LUR-tree[33]는 이동 객체의 위치 변경으로 발생하는 갱신 연산을 R-tree의 노드 내에서의 갱신으로 제한함으로써 노드의 구조 변경에 따른 비용을 감소시켰다. 하지만 검색 비용이 증가하는 문제점이 있다.

Tayeb[59]은 이동 객체의 이동 방향을 이용하여 보고 위치에서 미래 위치까지의 미래 궤적을 PMR-quadtree[54]에 선분으로 저장한다. 하지만 색인을 시간 구간 마다 재생성해야 하는 문제가 있으며, PMR-quadtree의 특성상 중복 저장의 문제를 가진다. 이동 객체는 위치 보고 후에도 계속적으로 이동을 하고 있으며, 보고 시각, 방향, 속도 등의 파라미터를 이용하여 현재 및 미래의 위치를 계산할 수 있다. 이러한 위치 예측을 위해서는 색인이 시간에 대한 함수를 표현 또는 저장하여야 한다.

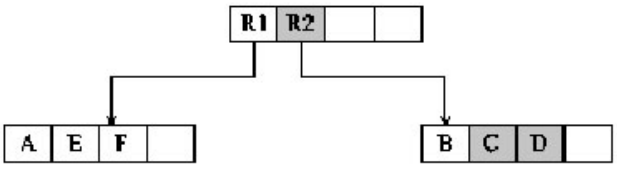
TPR-tree[53]는 이동 객체가 위치를 보고한 시점 이후의 대략적인 위치(특정 임계 값 범위를 초과하지 않는)를 검색할 수 있도록 하였다. 이 경우 이동 객체의 위치보고는 현재시간과 공간좌표, 그리고 이동 객체의 이동속성을 포함하게 되고 이동속성을 사용하여 예측된 위치와 실제 이동 객체의 위치

의 오차가 미리 정해진 임계 값을 초과하는 순간에 위치보고가 이루어진다. 따라서 충분히 넓은 임계 값을 설정할 경우 색인상에 발생하는 삽입연산의 횟수가 크게 감소한다는 특징이 있다.

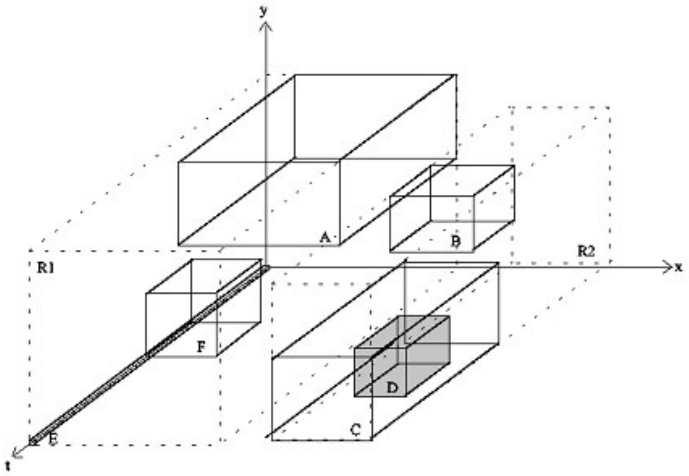
2.4.2 과거 위치 검색을 위한 시공간 색인 연구

이동 객체는 시간에 따라 위치가 변하기 때문에 시공간 객체로 볼 수 있다. 시공간 색인 중에서 이동 객체를 실험한 색인에는 3DR-tree[60], HR-tree[36]가 있으며 현재와 과거 위치를 검색하기 위한 2+3 R-tree[38]가 있다.

기존 2차원 공간 데이터를 저장하는 R-tree[26]에 기반한 3DR-tree[60]는 시간을 또 다른 하나의 축으로 추가하여 이동 객체를 색인한다. 즉, 객체가 시간 $[t_i,t_j]$ 동안 위치 (x_i,y_i)에 있고, 시간 $[t_j,t_k]$ 동안 위치 (x_j,y_j)에 있다면, 3차원 시공간에서 선분 $\overline{[(x_i,y_i,t_i),(x_i,y_i,t_j))}$와 선분 $\overline{[(x_j,y_j,t_j),(x_j,y_j,t_k))}$으로 궤적을 표현할 수 있으며, 3차원 R-Tree를 사용하여 색인한다. 그러나 3DR-tree는 두 끝점이 알려져 있는 선분만 저장할 수 있기 때문에 현재 위치를 저장할 수 없다. 또한 특정 시간과 공간 영역이 주어지는 영역 질의에 높은 성능을 보이지만, 노드 분할 시에 시간 도메인에 대한 고려가 없기 때문에 공간 활용도 크게 떨어지는 단점이 있다. 그림 6은 3DR-tree의 구조와 노드 C와 D가 심하게 겹치는 모습의 예이다. 하지만, 현재까지 3DR-tree는 영역 질의 성능이 우수한 색인으로 분류된다.



(a) 3DR-tree의 구조



(b) 노드의 MBB와 겹침

그림 6 3DR-tree의 노드 구조와 겹침 문제

HR-tree[36]는 R-tree에 거래시간 개념을 추가하여 이력 정보를 표현할 수 있으며 연속적인 상태를 표현하기 위하여 R-tree에 중첩(overlapping) 개념을 추가한 것이다. 그림 7과 같이 HR-tree는 타임스탬프마다 다른 색인 인스턴스로 구성된다. 기본적인 기술은 원래의 트리는 유지하고 상태가 변한 루트와 가지를 대체함으로써 현재와 과거를 유지하는 것이다. 상태가 변하지 않는 가지들은 복제되지 않는다. HR-tree는 이동 객체들의 이동이 빈번하지 않은 경우에는 효율적이지만, 이동이 많이 발생하는 경우 단말 노드 및 비단말

노드를 새로 생성해야 하고, 특히 영역 질의의 성능이 저하되는 문제를 가진다.

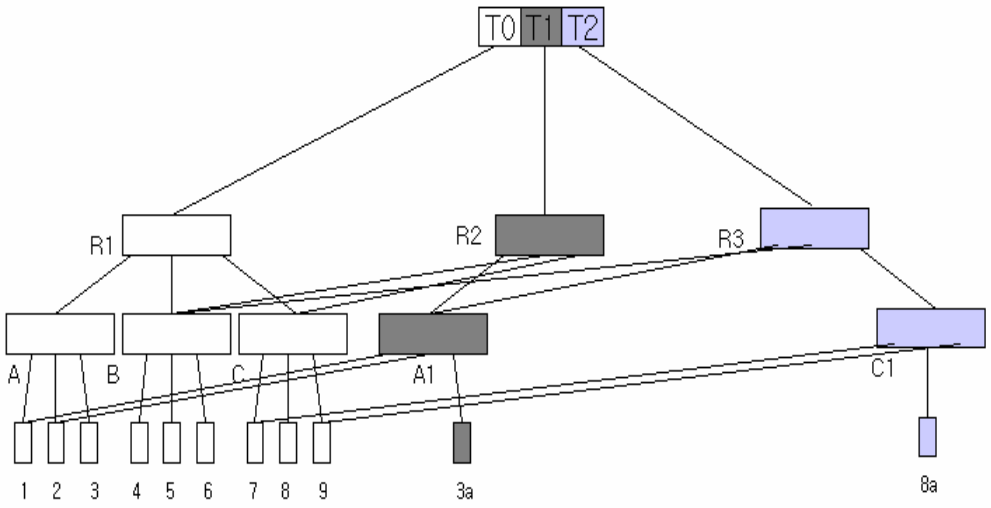


그림 7 HR-tree의 구조.

현재 및 과거 위치를 검색하기 위한 2+3 R-tree[38]는 3DR-tree의 문제점을 2차원 점과 3차원 선분에 대해 각각 따로 저장하는 두 개의 다른 R-tree를 사용함으로써 해결한다. 2차원 점은 현재 시점에서 객체의 공간 상의 점을 표현하며, 3차원 선분은 객체의 과거 이력을 표현한다. 2+3 R-tree에서 객체의 위치에 대한 끝 시간을 아직 모른다면, 그 데이터는 2차원 R-tree에서 객체의 위치에 대한 시작 시간과 oid를 유지하면서 저장된다. 열린(opened) 객체의 현재 상태에서 끝 시간을 알게 될 경우, 2차원 R-tree에서 해당 엔트리를 삭제하고, 3차원 시공간 상의 선분을 구성하여 3차원 R-tree에 삽입한다. 이와 같이 두 개의 색인을 유지하기 때문에 새로 보고되는 이동 객체의 위치에 따라 모두 삽입 되어야 하며 두 개의 R-tree에서 질의를 수행해야 하는 문제

점이 있다.

2.4.3 궤적 검색을 위한 궤적 색인 연구

이 논문에서는 궤적 질의와 복합 질의와 같은 궤적 검색 질의를 위해 제안된 이동 객체 색인을 궤적 색인으로 분류한다. 궤적 색인은 이동 객체의 과거 궤적을 효과적으로 추출하기 위하여 궤적의 요소인 선분들을 삽입 시에 궤적 보전 정책을 사용하는 색인이다. 궤적 보전 정책은 선분의 삽입 시에 기존의 공간 근접성을 이용하는 공간 색인의 삽입 정책과는 달리, 같은 객체의 궤적을 최대한 같은 단말 노드에 저장하도록 하는 정책이다.

STR-tree[46]는 이동 객체의 궤적을 선분의 집합으로 나타내고, 보존 파라미터를 사용하여 같은 객체의 궤적을 근접한 페이지에 저장하도록 유도한다. STR-tree의 부분적인 궤적 보존 정책은 3DR-tree에 비해 궤적 질의의 성능 향상 효과가 낮을 뿐만 아니라, 삽입 정책에서 공간 근접성과 같은 공간 구별이 낮아지기 때문에 영역 질의의 성능이 좋지 않다.

TB-tree[46]는 궤적 질의 및 복합 질의 성능 향상을 위하여 극단적인 궤적 보존 정책을 사용하였다. 하나의 단말 노드는 오직 동일한 궤적에 속하는 선분들만 저장할 수 있다. 공간적으로 근접한 선분들이 같은 궤적이 아니면, 서로 다른 노드에 저장된다. TB-tree는 이동 객체의 궤적을 3차원 최소경계박스에 저장하고, 그림 8과 같이 최소경계박스의 연결로 이동 객체의 모든 궤적을 표현한다. 단말 노드에서 이동 객체의 궤적은 양방향 연결 리스트(linked list)로 구성되어 있으므로 궤적에 대한 질의를 처리할 때는 뛰어난 성능을 보인다. 그러나 TB-Tree는 궤적에 대한 질의 성능은 좋은 반면 시간 구

간에 대한 질의나 공간 구간에 대한 질의에는 좋지 않은 성능을 보인다. 또한 특정 객체의 이동 궤적이 단절 되어 있는 경우 새로운 단말 노드를 생성하여 저장하기 때문에 RFID 태그를 위한 색인 구조로는 적합하지 않다.

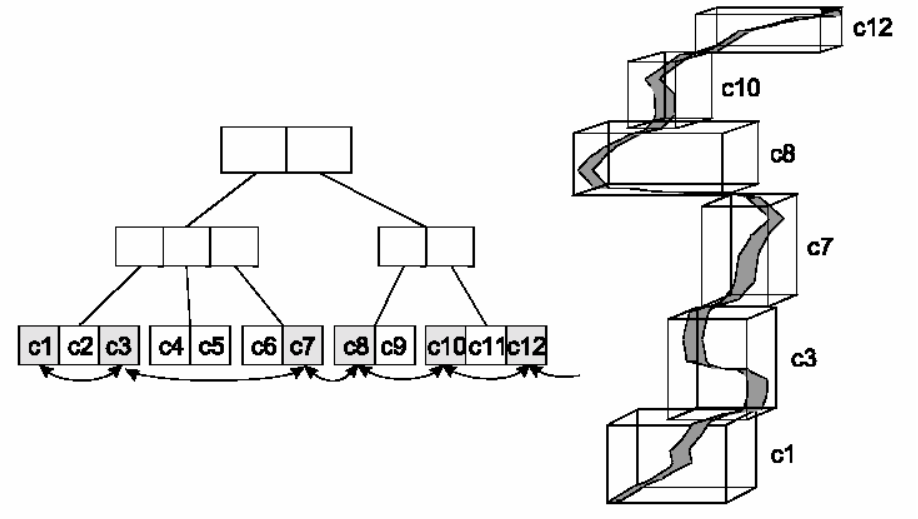


그림 8 TB-tree의 구조

3. 대상 환경 및 문제정의

이 장에서는 논문의 연구 대상인 RFID 환경에 대하여 먼저 설명한다. 그리고 이러한 환경에서 태그의 이동으로 인해 발생되는 이벤트를 소개한다. 또한 태그의 추적을 위해 궤적 모델 및 색인이 필요한 이유를 설명하며 궤적 모델을 기존의 방법을 사용하는 경우에 발생되는 문제점을 제시한다.

3.1 RFID 시스템 환경 정의

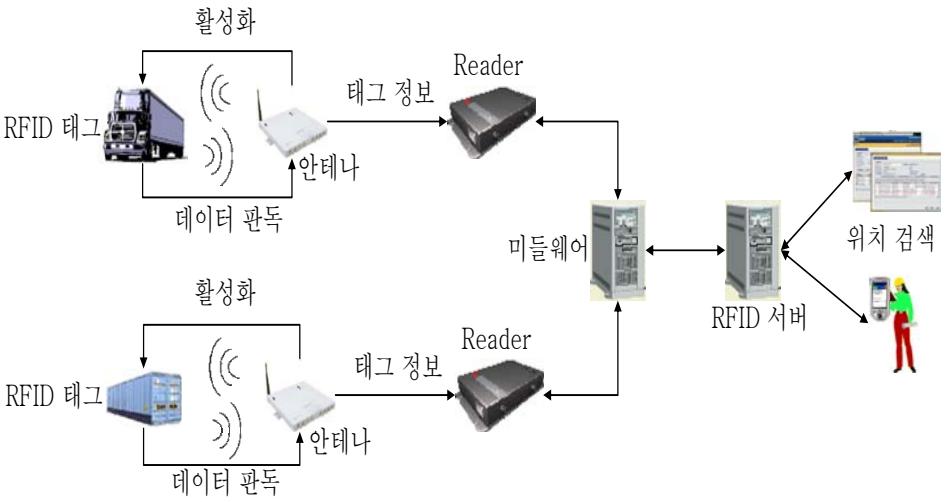


그림 9 RFID 시스템의 구성

RFID 시스템은 그림 9와 같이 태그와 판독기, 그리고 서버로 구성된다[1]. 태그는 전원 공급의 유무에 따라 전원을 필요로 하는 능동형(Active)형과 내부나 외부로부터 직접적인 전원의 공급 없이 판독기의 전자기장에 의해 작

동되는 수동형(Passive)형으로 구분된다. 능동형 태그는 판독기의 필요전력을 줄이고 판독기와의 인식거리를 멀리 할 수 있는 장점이 있으나, 전원 공급 장치를 필요로 하기 때문에 작동 시간의 제한을 받으며 수동형 태그에 비해 고가인 단점이 있다. 반면에 수동형 태그는 능동형 태그에 비해 매우 가볍고 가격도 저렴하면서 반영구적으로 사용이 가능하지만, 인식거리가 짧고 판독기에서 더 많은 전력을 소모한다는 단점이 있다. 이러한 태그는 물류 관리나 위치 추적과 같은 응용에서 제품이나 컨테이너와 같이 이동성이 있는 객체에 부착된다.

판독기는 응용 환경에 따라 전략적으로 중요한 지점에 설치되고, 고정되어 있으며, 미리 계산된 위치 좌표를 가질 수 있다. 또한, 판독기는 태그와 통신할 수 있는 공간 상의 영역을 가지며 이것을 판독기의 호출 영역(interrogation zone)이라고 한다.

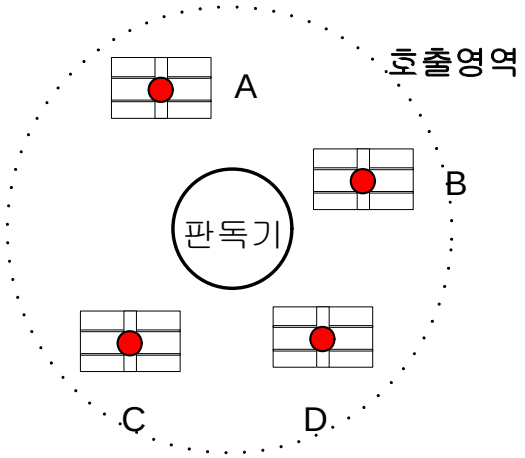


그림 10 태그 위치와 판독기 위치의 관계

태그의 실제 위치는 측정할 수 없기 때문에, 호출 영역 안에 존재하는 모든 태그의 위치는 판독기의 위치로 수집된다. 예를 들어 그림 10과 같이 판독기가 위치하는 경우에 호출영역에 들어온 태그 객체 A, B, C, D는 실제의 위치가 다르지만 판독기의 위치로 인식되므로 인식되는 위치는 모두 동일하게 된다.

응용 환경에 따라 판독기의 호출 영역의 크기도 다양하며, 판독기의 수에 따라 호출 영역이 서로 겹치는 상황이 발생한다. 호출 영역이 겹치는 경우, 주파수 간섭으로 인해 태그를 인식하지 못하는 충돌 문제가 발생할 수 있으며, 이 문제를 해결하기 위해서 공간적으로 가까운 판독기는 서로 다른 시간에 주파수를 사용해야 한다. 따라서 이 논문에서는 판독기가 주기적으로 주파수를 사용하여 태그를 인식하며, 공간적으로 가까운 판독기는 서로 다른 시간 프레임을 사용하여 충돌이 발생하지 않음을 가정한다.

이와 같이 호출 영역을 가지는 판독기는 영역에 들어오거나 나가는 태그의 정보를 수집하여 서버로 전송하고 이러한 정보를 사용하여 자동화 생산(automated manufacturing), 재고 관리(inventory tracking), 공급망 관리(supply chain management) 등과 같은 다양한 응용분야에 적용된다. 예를 들어 그림 11은 물류 관리를 위해 제품을 배송하는 회사에 RFID 시스템을 적용시키는 경우이다. 태그를 배송하고자 하는 제품에 부착하고 각 창고의 입구에 판독기를 설치하면 창고를 출입하는 제품을 통제할 수 있다.

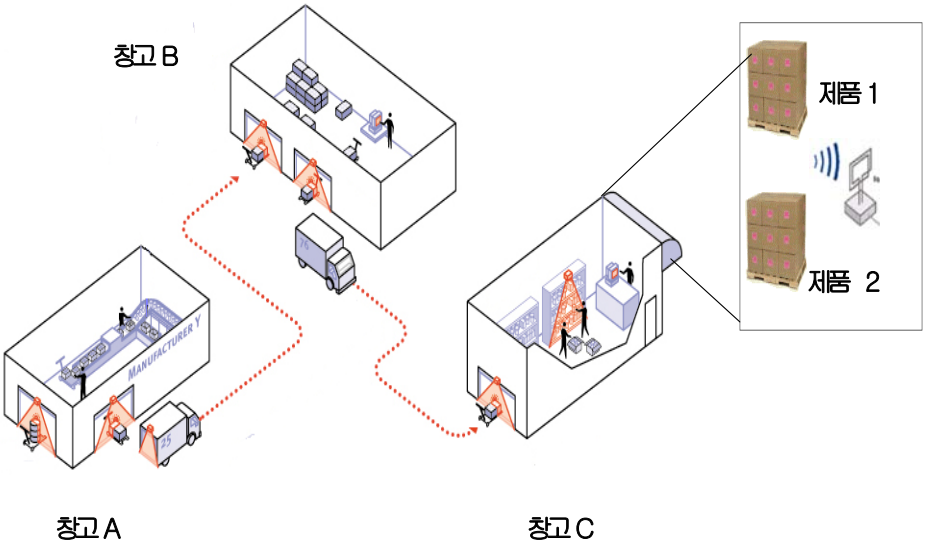


그림 11 RFID 시스템의 응용 분야 예

이러한 응용분야에서는 태그를 장착한 객체의 위치 추적이나 객체의 현재의 상태를 감시하는 서비스를 제공해야 한다. 즉, “제품 1이 현재 어느 창고에 있는가?”라는 태그의 현재 위치 추적이나, “두 시간 전에 창고 B를 나간 제품은 무엇인가?”라는 태그의 과거 위치 추적을 통해 제품의 위치 및 현재 상태를 감시하는 서비스를 제공해야 한다. 따라서 태그의 위치 추적을 위한 데이터 모델과 과거 위치 및 현재 위치 추적을 위한 질의의 효율적인 처리를 위한 색인 기법이 필요하다.

3.2 RFID 태그의 궤적

객체 식별자 *tid*를 가지는 태그가 판독기의 호출 영역 안으로 들어가서 인

식되면 Enter 이벤트가 발생하고, 호출 영역 밖으로 나가서 인식되지 않으면 Leave 이벤트가 발생한다. 이러한 이벤트가 발생할 때마다 판독기는 자신의 위치와 태그의 식별자를 서버로 전송한다. 이러한 정보를 이용하여 이 논문에서는 색인을 구성하는 방법을 제시한다.

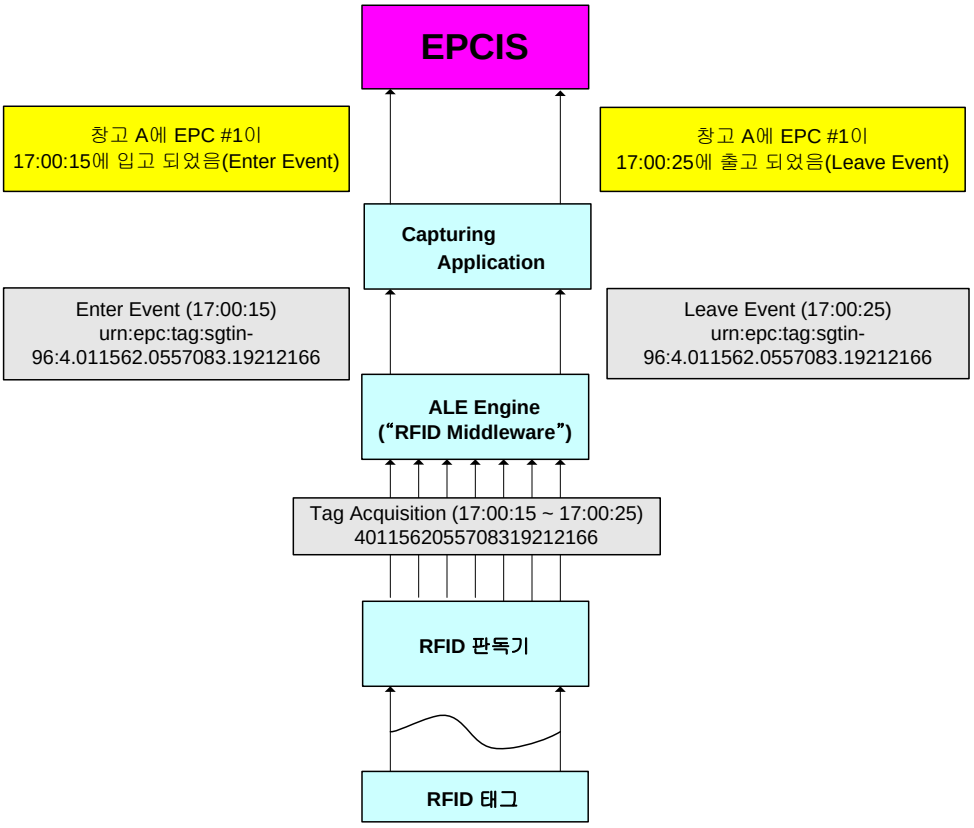


그림 12 RFID 시스템의 전체 구조

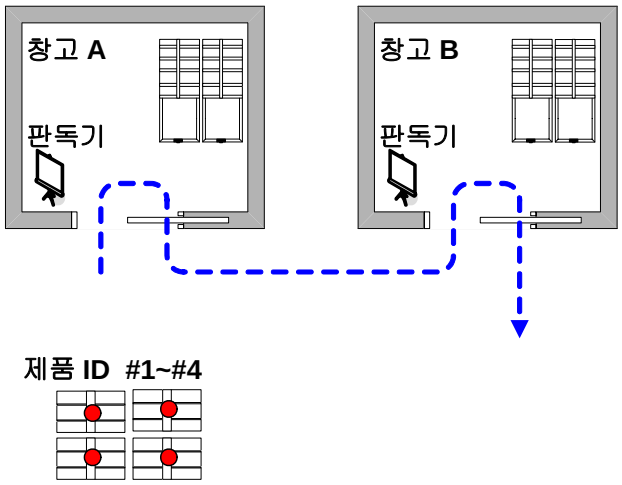
그림 12는 RFID 시스템의 전체 구조를 나타낸 것이다. 그림과 같이 RFID 시스템은 태그의 움직임을 읽어 정보를 전송하는 판독기와 전송된 데이터를 필터링하여 응용프로그램에 전송하는 ALE 미들웨어 및 최종적으로 다양한

서비스를 제공하는 EPC-IS 등으로 구성되어 있다. 그림과 같이 판독기로부터 수집된 정보는 상위 레이어에 전송될수록 추상화된다.

이 논문에서는 가장 상위 레이어인 EPC-IS에 들어오는 정보를 가지고 태그 객체를 추적하기 위한 색인에 대하여 연구한다. 따라서, 식별자 *tid*를 가지는 태그가 판독기와 연결된 안테나의 호출 영역(*loc*)으로 이동할 때, 시간 t_{enter} 에서 인식 영역 안으로 들어가면 Enter 이벤트가 발생하고 (*tid, loc,* t_{enter}) 가 EPC-IS로 전송된다. 반대로 태그가 호출 영역을 벗어나면 Leave 이벤트가 발생하며 (*tid, loc,* t_{leave})가 EPC-IS로 전송된다. 이때 두 이벤트의 *tid*와 *loc*는 동일하며 t_{leave}는 t_{enter}보다 항상 크다.

그림 13은 RFID 응용에서 태그를 부착한 물품이 각 창고를 이동하는 모습과 생성된 데이터를 보여준다. 그림과 같이 태그를 부착한 물품이 입고되면 고정 데이터 테이블에 물품에 대한 정보가 등록된다. 이 정보는 등록 후에 변하지 않는 정보로 물품 자체에 대한 정보이다. 그 후에 물품이 시간에 따라 판독기가 설치된 창고에 입/출고되어 이동하면 이벤트 데이터 테이블에 물품 이동 정보가 저장되게 된다.

이러한 응용에서는 다양한 질의를 처리할 수 있다. 예를 들어 “16:00 창고 B에 있었던 물품의 제품명은?” 이라는 질의를 처리한다고 가정하자. 먼저 이벤트 데이터 테이블에 저장된 데이터에서 “16:00 창고 B”에 해당되는 데이터를 검색한다. 그 결과로 제품 ID #1~4가 추출되며 이 정보를 가지고 고정 데이터 테이블을 검색하여 제품명을 추출하여 질의를 처리한다. 이 논문에서는 이벤트 데이터 테이블에 저장된 태그의 이동 정보를 이용하여 색인을 구축하는 방법을 제시한다.



고정 데이터

제품 ID	제품명	제조사	단가
#1	쵸코파이	오리온	1000
#2	카스타드	오리온	800
#3	빅파이	롯데	2000
#4	새우깡	농심	500

이벤트 데이터

제품 ID	위치	입고시간	출고시간
#1	창고 A	11:00	15:00
#2	창고 A	11:00	15:00
#3	창고 A	11:00	15:00
#4	창고 A	11:00	15:00
#1	창고 B	15:30	18:00
#2	창고 B	15:30	18:00
#3	창고 B	15:30	18:00
#4	창고 B	15:30	18:00

그림 13 RFID 응용의 예와 생성되는 데이터

태그의 이동으로 인하여 생성되는 궤적은 다음과 같이 모델링될 수 있다. 첫째, Enter 이벤트 발생시 생성되는 3차원 공간의 점으로 표현할 수 있다. 둘째, Enter 이벤트 발생시 생성되는 3차원 공간의 점을 이어서 선으로 표현할 수 있다. 마지막으로 Enter 이벤트 발생시 생성되는 시공간 점과 Leave 이벤트 발생시 생성되는 3차원 공간의 점을 이어서 선으로 표현할 수 있다.

먼저 그림 14와 같이 태그 궤적을 Enter 이벤트 발생시 생성되는 3차원 공간의 점으로 표현할 수 있다. 이 경우에는 처리할 수 있는 질의는 Timestamp의 시간과 일치하는 시간구간을 가져야 한다. 만약 Timestamp 사이의 시간 구간을 가지는 질의를 처리하는 경우에는 질의의 시간 구간을 늘여서 처리해야 하므로 최악의 경우 인덱스 전체를 검색해야 하는 문제가 발생한다.

Tid	Rid	Timestamp
e5	r7	t1
e5	r8	t3
e5	r9	t5

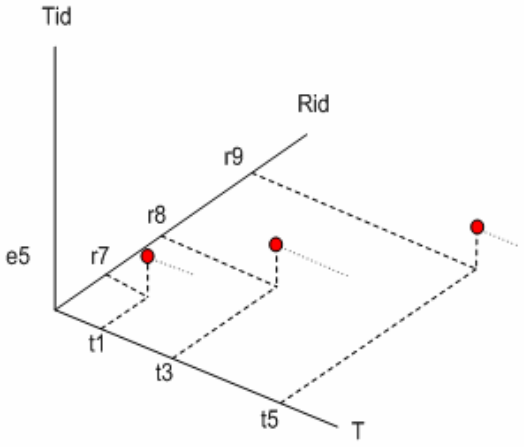


그림 14 태그 궤적을 Enter 점으로 표현

그림 15와 같이 태그 궤적을 Enter 이벤트 발생시 생성되는 3차원 공간의 점을 잇는 선으로 표현할 수 있다. 점으로 표현시 Timestamp의 사이의 시간 구간을 가지는 질의를 처리하기 위해서는 질의의 시간 구간을 늘여야만 하지만 이 경우에는 선으로 표현되므로 모든 질의를 그대로 처리할 수 있다. 그러나 태그가 판독기에 머물러 있는지 아니면 판독기에서 다른 판독기로 이동하고 있는지를 알 수 없다. 특히 물류 관련 응용에서는 창고 비용 또는 운송 비용을 계산하는 경우가 비일비재하므로 Enter 이벤트 점을 잇는 선으로 태그의 궤적을 표현하는 방법은 부적당하다.

Tid	Rid	Timestamp
e5	r7	t1
e5	r8	t3
e5	r9	t5

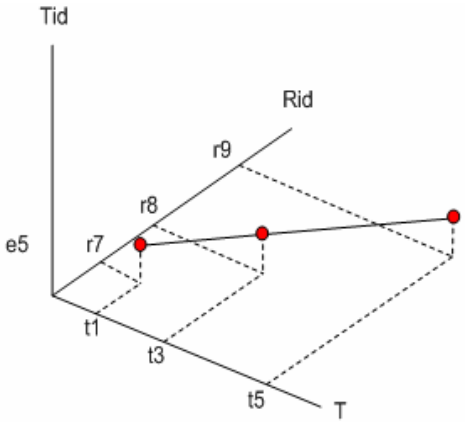


그림 15 태그 궤적을 Enter 점을 잇는 선으로 표현

그림 16은 태그 궤적을 Enter 이벤트와 Leave 이벤트 발생시 생성되는 3차원 공간의 점을 잇는 선으로 표현한 것이다. 앞의 두 방법과는 다르게 질의의 시간구간을 확장하지 않고 질의를 처리할 수 있으며 태그가 판독기에 머물러 있는지 아니면 판독기에서 다른 판독기로 이동하고 있는지를 알 수 있다. 따라서 이 논문에서는 태그의 궤적을 Enter 이벤트와 Leave 이벤트 발생시 생성되는 3차원 공간의 점을 잇는 선으로 표현한다.

Tid	Rid	Enter	Leave
e5	r7	t1	t2
e5	r8	t3	t4
e5	r9	t5	t6

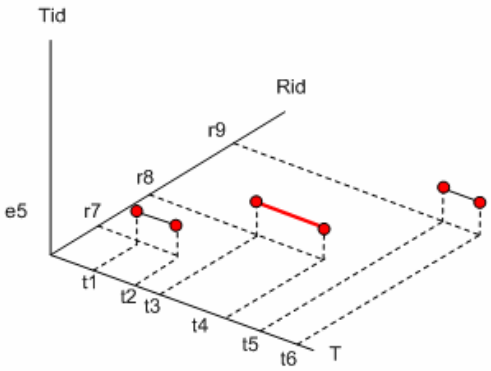


그림 16 태그 궤적을 Enter와 Leave 점을 잇는 선으로 표현

이 논문에서는 판독기 r에서의 태그의 단일궤적을 다음과 같이 정의한다. 정의에서 *tid, rid, t*는 3차원 공간의 각 축을 의미하며 *tid*는 태그 객체의 식별자, *rid*는 판독기의 식별자, t_{enter}, t_{leave}는 각각 *Enter*와 Leave 이벤트의 발생 시간을 의미한다.

정의 1: 판독기 rid_j에서의 태그객체 tid_i의 단일궤적 *tr*

$$tr = \{(tid,rid,t) \in R^3 \mid \ tid = tid_i \ , \ rid = rid_j \ , \ t_{enter} \le t \le t_{leave}\}$$

예를 들어 그림 17과 같이 태그 객체 tid_i가 판독기에 들어와 Enter 이벤트가 t_{enter}시간에 발생하고, 판독기에서 빠져나가 Leave 이벤트가 t_{leave} 시간에 발생했다고 가정하자. 이 경우에 판독기의 위치를 rid_j라 가정하면 Enter 이벤트에는 (tid_i, rid_i, t_{enter})의 시공간 위치가 보고되며 Leave 이벤트에는 (tid_i, rid_i, t_{leave})의 시공간 위치가 보고된다. 따라서 태그의 단일궤적은 이 두 개의 시공간 위치를 연결한 선분이 된다.

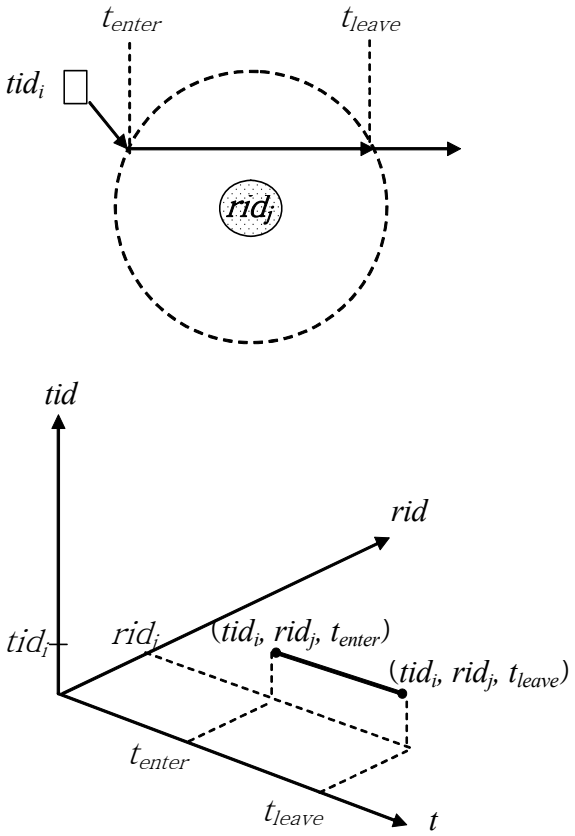


그림 17 태그의 단일 궤적

태그의 궤적은 다음과 같은 특징을 가지고 있다. 첫째, 판독기의 논리적인 위치로서 태그의 위치를 표현한다. 둘째, 판독기의 간섭으로 인해 궤적의 중복이 발생한다. 셋째, 판독기의 성김으로 인해 궤적의 분절이 발생한다.

먼저 태그의 위치는 그림 18과 같이 판독기의 논리적인 위치로서 표현된다. 즉, 두 개의 태그가 동일하지 않는 공간 좌표를 가지더라도 판독기의 위치로서 태그의 위치를 인식하므로 동일한 궤적으로 표현된다. RFID의 응용에서는 태그의 실제 위치보다는 어느 판독기에 위치하는가가 중요하므로 태그

의 위치를 논리적인 위치로 표현해도 문제가 발생하지 않는다.

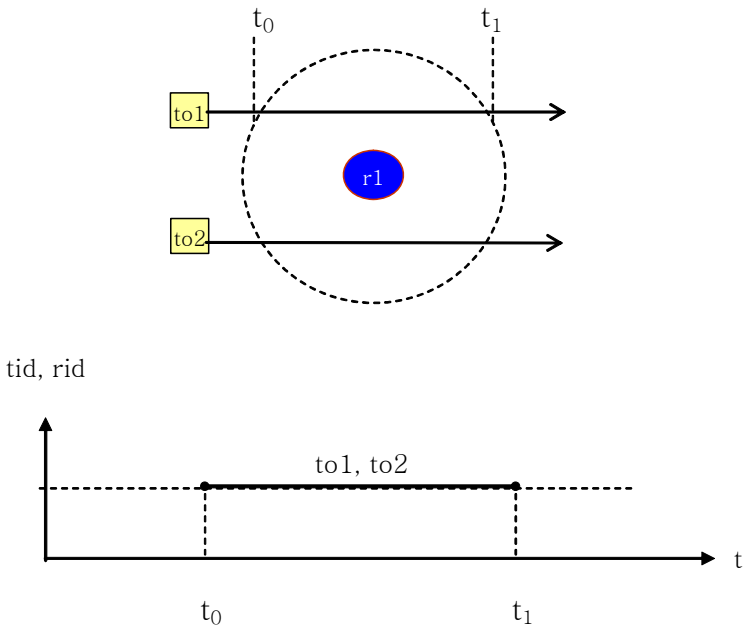


그림 18 판독기의 논리적인 좌표로 표현

판독기의 인식 영역에 간섭이 생기면 태그의 궤적은 중복이 발생한다. 이것은 그림 19와 같이 판독기의 인식 영역 구간이 중복되는 경우에 태그의 궤적도 t_1~t_2 시간 사이에 중복이 된다. 그러나 실제 응용에서는 판독기의 가격이 매우 고가이므로 인식 영역 구간이 중복되도록 배치하지 않는다. 또한 태그의 궤적이 중복되더라도 질의를 처리하는데 문제가 발생하지 않는다.

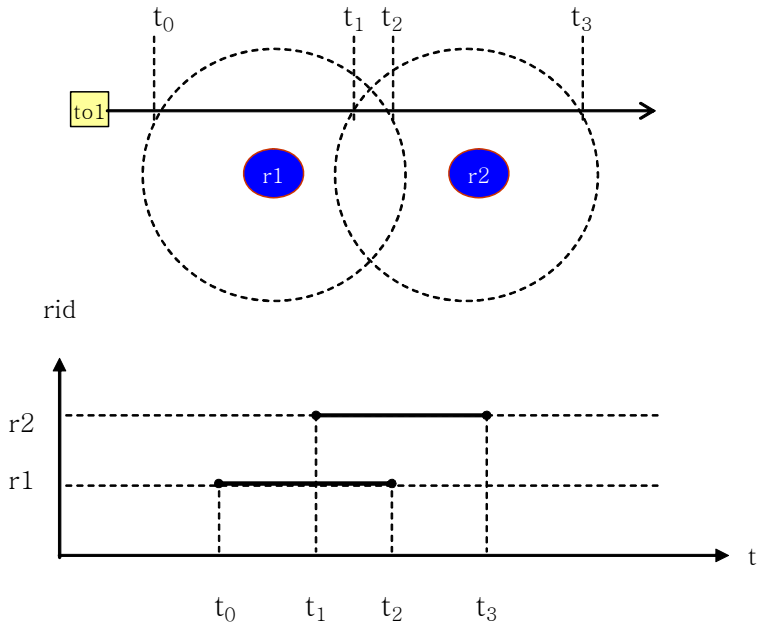


그림 19 태그 궤적의 중복

마지막으로 판독기의 인식 영역이 성기면 태그의 궤적은 단절이 발생한다. 즉, 태그가 인식 영역에 존재하는 거주 시간 구간과 인식 영역 밖에 존재하는 이주 시간 구간으로 구분할 수 있다. 예를 들어, 그림 20와 같이 판독기의 인식 영역 구간이 성기는 경우에 거주 시간 구간은 $[t_0,t_1],[t_2,t_3]$이고, 이주 시간 구간은 $(t_1,t_2),(t_3,\infty)$이다(∞는 unknown을 의미한다). 대부분의 응용에서는 판독기에 머물러 있는 시공간 구간 정보만을 필요로 하므로 역시 태그의 궤적을 정의 1과 같이 모델링하더라도 문제가 발생하지 않는다.

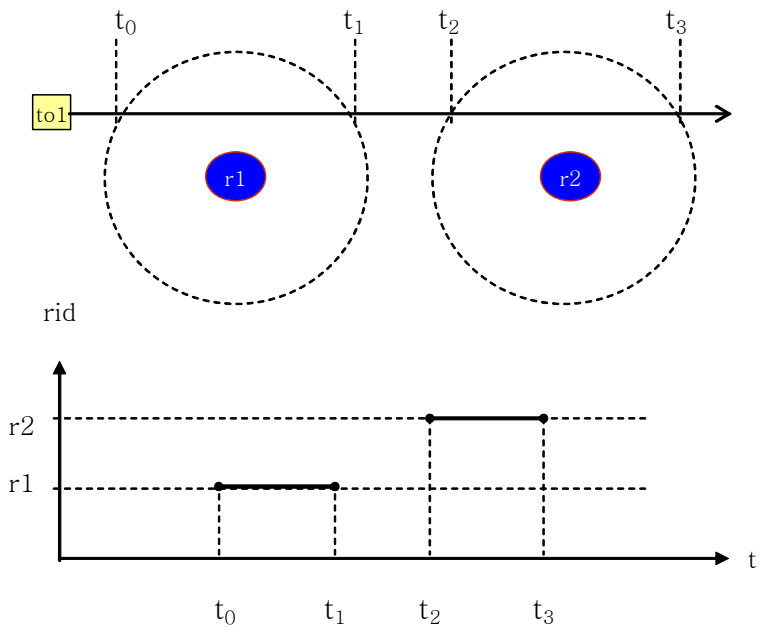


그림 20 태그 궤적의 단절

3.3 문제 정의

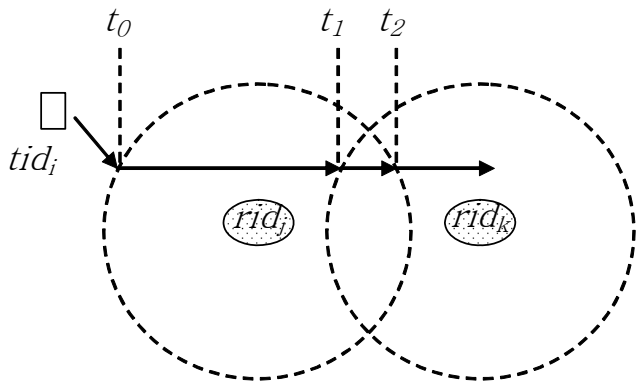
기존 이동 객체의 연구는 GPS 장치를 탑재한 차량의 위치를 수집하여 색인에 저장한다. 이동 객체는 속도나 방향이 바뀔 때마다 위치를 보고하며, 이동 궤적을 다중선으로 표현한다. 그리고 질의는 선형 보간법을 사용하여 질의 시간에 해당하는 위치를 찾는다.

그러나 RFID 태그의 경우, 태그는 자신의 위치를 보고할 수 있는 기능이 없기 때문에, 속도나 방향이 바뀔 때마다 위치를 보고할 수 없다. 단지 판독기와 연결되어 있는 안테나 주위에서만 위치 보고가 가능하다. 따라서 수집

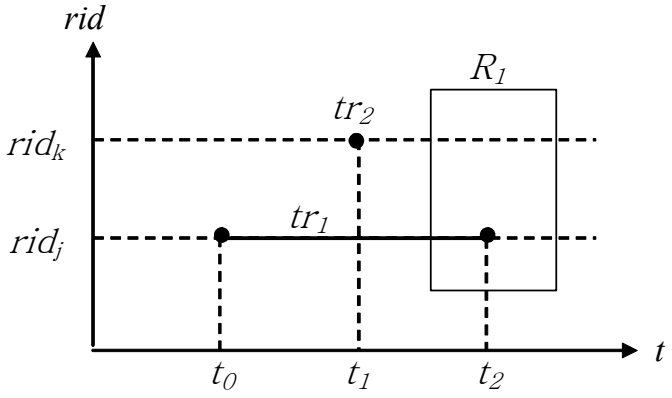
되는 객체의 위치는 아주 제한적이며, 정확한 이동 궤적을 보여주지 못한다. 호출 영역이 서로 겹치지 않고 공간적으로 아주 멀리 떨어져 있는 경우, 이동 궤적을 다중선으로 나타낼 수 없는 문제점이 있다. 따라서 태그의 궤적을 추적하기 위해서는 궤적을 모델링하고 이를 색인으로 구축하는 방법이 필요하다.

기존 이동 객체 데이터베이스에서 3DR-tree나 TB-tree는 과거 위치만을 저장한다. 물류와 같은 응용에서는 과거 위치 질의뿐만 아니라 현재 위치에 대한 질의도 빈번하게 일어나므로 과거와 현재를 모두 색인에 저장할 수 있어야 한다. 예를 들어서, “현재 12번 레인에 위치한 화물을 찾아라”와 같은 현재 위치 질의를 처리하기 위해서는 12번 레인의 현재 존재하는 화물을 검색해야 한다. 다른 예로서, “지금 한진 차량이 수송해야 하는 화물의 위치를 찾아라”와 같은 질의가 들어올 경우 각 화물에 대한 현재 위치를 검색해야 하며, 객체의 현재 상태에 대한 정보가 색인에 저장되어 있어야 한다.

태그를 위해 기존 연구에서 사용한 궤적의 표현 방법을 사용하면 판독기에 머무는 객체의 현재 위치를 찾을 수 없는 문제가 발생한다. 만약 태그가 판독기의 인식범위에 들어와 나가지 않고 머무는 경우에는 현재 질의 수행 시에 질의의 결과임에도 불구하고 질의의 후보가 될 수 없다. t_{now}를 현재시간이라고 가정하고 t_{enter}와 t_{leave}를 Enter 이벤트와 Leave 이벤트의 발생시간이라 가정하자. 만약 $t_{enter} \le t_{now} < t_{leave}$이면 Enter 이벤트만 발생하고 Leave 이벤트는 발생하지 않게 된다. 따라서 궤적은 Enter 이벤트 시에 보고된 시공간 위치인 점으로만 표현되므로 판독기에 머물러 있다는 정보를 표현할 수 없다. 따라서 질의 수행 시 이러한 태그를 찾을 수 없는 문제가 발생한다.



(a)



(b)

그림 21 판독기에 머무는 태그의 문제발생

예를 들어 그림 21-(a)와 같이 판독기 r_1에 태그가 시간 t_0에 들어와 t_2에 빠져나간다고 가정하자. 정의1에 따라 궤적은 그림 21-(b)의 tr_1과 같이 두 개의 시공간 위치를 연결한 선분으로 표현된다. 만약 태그가 판독기 r_2에 시간 t_1에 들어와 빠져나가지 않으면 태그의 Leave 이벤트는 발생하지 않게 된다. 따라서 궤적은 tr_2와 같이 Enter 이벤트 발생시에 보고된 시공간 위치인 점으로 표현되며 이것이 색인에 삽입되게 된다. 그림 21-(b)의 R_1과 같이 특정 시

간에 판독기 r_1과 r_2에 위치하는 태그를 찾는 질의를 수행한다고 가정하자. 질의가 수행되면 R_1에 포함되는 궤적을 찾는데, tr_1은 포함되나 tr_2는 포함되지 않으므로 태그가 판독기 r_2에 머물러 있음에도 불구하고 그 정보가 표현되지 않으므로 tr_2는 질의의 결과에서 제외된다. 따라서 판독기에 들어와 Enter 이벤트만 발생되고 머무는 태그의 현재 위치를 표현할 수 있는 방법이 필요하다.

기존 이동 객체를 위한 색인들은 객체 식별자를 색인 도메인으로 사용하지 않는다. 대신에 3DR-tree 같은 경우, 단말 노드의 엔트리에 속성으로서 저장된다. 비단말 노드는 자식 노드에 대한 포인터와 MBB만 가지고 있기 때문에 객체 식별자에 대한 정보를 유지하지 않는다. 따라서 객체 식별자를 인수로 가지는 질의를 처리하기 위해서는 주어진 시간에 포함되는 모든 객체를 탐색하고 객체 식별자로 여과해야 한다.

이 논문에서는 이러한 문제를 해결하기 위하여 태그의 궤적을 위한 구간 데이터 모델을 제시한다. 먼저 태그의 궤적을 정적 구간(static interval), 동적 구간(dynamic interval)으로 정의한다. 정적 구간은 시간에 독립적인 일정한 시간 구간을 가지는 선분이며 동적 구간은 시간에 따라 시간 축 길이가 변하는 선분이다. 따라서, Enter 이벤트만 발생한 태그의 궤적은 시간에 종속적인 선분으로 표현되므로 현재 질의를 처리할 수 있다. 또한 태그의 궤적 추적을 위한 질의를 분류하고 이러한 질의의 효율적인 처리를 위해 태그의 식별자를 색인의 차원으로 추가하는 방법을 제시한다.

또한 제시된 태그의 구간 데이터 모델에 따라 색인의 구축 방법을 제시한다. 이러한 색인은 R-tree를 기반으로서, 구간의 특징을 고려한 삽입, 분할 알

고리즘을 제시하여 효율적으로 질의를 처리한다. 마지막으로 제시된 색인을 구현하고 기존 삽입, 분할 알고리즘을 사용하는 색인들과 다양한 실험데이터를 가지고 평가하여 제시한 색인의 우수성을 입증한다.

4. 태그의 궤적 표현을 위한 구간 데이터 모델

이 장에서는 RFID 시스템의 태그를 위한 질의를 정의하고 판독기에 머무는 태그를 표현하기 위한 정적, 동적 구간을 정의한다. 또한 질의의 효율적인 수행을 위해 태그 식별자를 색인의 차원으로 추가할 때 발생되는 문제점 및 해결책을 제시한다.

4.1 태그를 위한 질의 정의

RFID 시스템에서 태그와 관련된 질의를 분류하기 위해 태그는 아래와 같이 3가지 속성을 가진다고 가정한다.

1) 태그 식별자(*tid*)

2) 위치 좌표(*rid*)

3) 시간(*time*)

위의 3가지 인수는 단일 값을 가지거나 범위를 가질 수 있다. 태그 식별자는 태그를 인식할 수 있는 유일한 식별자이다. 위치 좌표는 2차원 또는 3차원 공간의 좌표를 가지며 이 논문에서 2차원 공간만을 다룬다. 시간은 태그가 판독기를 들어올 때와 나갈 때 보고되는 시간을 의미한다.

RFID 시스템에서 태그의 위치를 추적하기 위해서는 다양한 질의가 필요하다. 질의는 특정 시공간 지역에 포함되는 궤적을 추출하는데 다음과 같이 네 종류로 분류된다. 다음에서 질의의 분류를 위하여 $[a^{\vdash}, a^{\dashv}]$의 표현을 사용하

는데 이 의미는 좌표 축에 투영한 값의 범위를 나타낸다.

Type 1

Find 질의: Q = ([*tid*$^{\vdash}$, *tid*$^{\dashv}$], [*t*$^{\vdash}$, *t*$^{\dashv}$]) - 시간 [*t*$^{\vdash}$, *t*$^{\dashv}$]에 태그 식별자 [*tid*$^{\vdash}$, *tid*$^{\dashv}$]가 이동한 판독기의 위치 반환.

예: *"18:00~23:00까지 #13~#18 태그가 위치한 판독기는?"*

Type 2

Look 질의: Q = ([*rid*$^{\vdash}$, *rid*$^{\dashv}$], [*t*$^{\vdash}$, *t*$^{\dashv}$]) – 시간 [*t*$^{\vdash}$, *t*$^{\dashv}$]에 특정 판독기 [*rid*$^{\vdash}$, *rid*$^{\dashv}$]에 위치한 태그의 식별자 반환.

예: *"18:00~23:00까지 판독기 #1-3~ #1-4에 위치한 태그는?"*

Type 3

History 질의: Q = (*tid*) – 태그 식별자 *tid*가 지나간 모든 판독기의 위치 반환.

예: *"태그 #13이 거쳐간 모든 판독기의 위치를 찾아라."*

Type 4

With 질의: Q = (*tid*, [*t*$^{\vdash}$, *t*$^{\dashv}$]) – 시간 [*t*$^{\vdash}$, *t*$^{\dashv}$]에 태그 식별자 *tid*와 같이 이동한 모든 태그들의 식별자 반환.

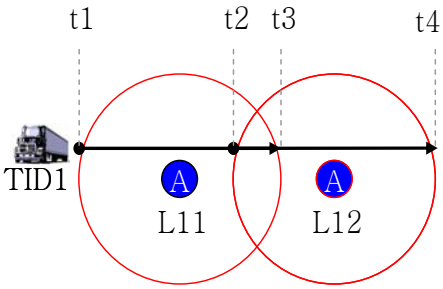
예: *“18:00~23:00까지 #13 태그와 같이 있었던 모든 태그를 찾아라.”*

Find 질의는 태그 식별자와 시간이 주어질 때, 태그의 위치를 검색하는 질의이다. Find 질의의 예로는 “한진 차량이 2시부터 3시까지 적재한 화물의 위치는?”가 있다. “적재한 화물”이 태그 식별자이고 “2시부터 3시 사이에”가 시간 범위가 된다. 질의 결과는 2시부터 3시까지 적재된 화물의 위치인 판독기의 위치가 된다. Look 질의는 공간과 시간이 주어질 때, 해당 영역에 존재하는 태그를 검색하는 질의이다. Look 질의의 예로는 “3번 레인에서 2시부터 3시까지 존재했던 컨테이너를 검색하라”가 있다. “3번 레인”은 질의에서 사용되는 판독기의 위치이며, 2시부터 3시까지 주어진 영역에 존재하는 태그를 찾는다.

질의에서 주목할 만한 것은 History 질의는 시간 범위가 시간 축 전체인 Find 질의이며 WITH질의는 Find 질의를 수행하고 난 뒤에 Look 질의를 수행하여 처리할 수 있다. 따라서 RFID 시스템에서는 Find 질의와 Look 질의가 기본 질의이다.

4.2 구간 데이터 모델

태그의 위치 추적을 위한 질의를 처리하기 위해서는 각 이벤트가 발생할 때마다 데이터를 생성해야 한다. 그림 22는 태그 식별자 TID1을 가지는 태그의 이동을 이벤트가 발생할 때마다 생성한 예를 보여준다. 태그가 Enter와 Leave 이벤트를 발생하는 시간 동안에는 호출 영역에 존재했음을 의미하기 때문에 태그의 궤적을 $(tid, rid, t_{enter}, t_{leave})$와 같이 표현할 수 있다.



tid	loc	t_{enter}	t_{leave}	state
TID1	L1	t1	-	Enter
TID1	L2	t2	-	Enter
TID1	L1	-	t3	Leave
TID1	L2	-	t4	Leave

그림 22 TID1의 이벤트 발생시마다 데이터 생성 예

태그가 호출 영역 안으로 들어가면, Enter 이벤트가 발생하고, 이때 t_{enter} 값을 알 수 있다. 그러나 언제 호출 영역 밖으로 나올지 모르기 때문에 t_{leave} 값은 아직 알 수 없다. 현재 위치 질의, 예를 들어 태그가 현재 어느 판독기의 호출 영역에 들어 있는지를 알기 위해서는 태그의 현재 상태를 데이터베이스에 유지하고 있어야 하므로, Enter 이벤트가 발생할 시에 t_{leave} 값을 표현할 방법이 필요하게 된다.

아직 값이 결정되지 않은 t_{leave}를 표현하기 위한 방법으로 t_{leave}를 시간 도메인의 최대값(MAX TIME)으로 표현하고 Leave이벤트가 발생할 시에 t_{leave} 값을 갱신할 수 있다. 그러나 시간 도메인의 최대값은 응용에 따라 틀리므로 그 값이 모호하며 현재 시간 이후의 미래 질의에 대하여 질의 처리 시 항상 특정 판독기에 머문다는 잘못된 질의 결과를 반환하게 된다.

따라서 이 논문에서는 태그가 판독기의 호출 영역에 머물러 t_{leave}의 값이 결정되지 않은 경우에 t_{leave}를 시간에 종속적인 선분으로 표현하고 데이터 삽입이나 질의 시에 확장한다.

따라서 이 논문에서는 시간에 따른 태그의 궤적은 다음과 같이 두 종류의 구간인 정적 구간과 동적 구간으로 분류하여 정의한다. 다음의 정적, 동적 구간의 정의를 위하여 이 논문에서는 태그 식별자 *tid*, 판독기의 식별자 *rid* 및 시간 *t*를 3차원의 각 축으로, tid_i는 태그의 식별자, rid_j는 판독기의 식별자 t_{enter} 를 Enter 이벤트의 발생시간, t_{leave} 를 Leave 이벤트의 발생시간, t_{now} 를 현재 시간으로 가정한다.

정의 2: 정적 구간(static interval) =

$$\{(tid,rid,t)\in R^3 \mid \quad tid = tid_i,\ rid = rid_j\ ,\ t_{enter} \le t \le t_{leave}\}.$$

정의 3: 동적 구간(dynamic interval) =

$$\{(tid,rid,t)\in R^3 \mid \quad tid = tid_i,\ rid = rid_j\ ,\ t_{enter} \le t \le t_{now}\}.$$

태그가 판독기에 들어오면 동적 구간은 생성되며 시간 구간의 끝 값은 현재시간을 나타내는 t_{now}로 변경된다. 따라서 동적 구간의 시간 축 길이는 $t_{enter} \le t \le t_{now}$가 되며 t_{now}에 의해 시간 축 길이가 계속해서 변하게 된다. 판독기에 들어와 아직 빠져나가지 않은 태그의 궤적이 시간에 종속적인 선분으로 표현되므로 질의발생시간을 t_{query} 라 하면 $t_{enter} \le t_{query} < t_{leave}$에 발생되더라도 질의를 처리할 수 있다. 만약 태그가 판독기를 빠져 나가게 되면 시간 구간의 끝 값은 t_{leave}로 변경되어 시간 축 길이가 $t_{enter} \le t \le t_{leave}$인 고정 구간으로 변경되어 $t_{query} \ge t_{leave}$에 발생하는 질의를 처리할 수 있다.

예를 들어, 그림 23-(a)와 같이 태그가 판독기에 t_{enter}시간에 들어와 t_{leave}시간에 빠져 나간다고 가정하자. $t_{enter} \le t_{query} < t_{leave}$인 경우에 태그의 궤적은 그림 23-(b)와 같이 태그가 아직 판독기를 빠져 나오지 않았으므로 시간 축 범위가 $[t_{enter}, t_{now}]$인 동적 구간이 되어 질의 처리가 가능하다. $t_{query} \ge t_{leave}$인 경우에는 태그가 그림 23-(c)와 같이 판독기를 빠져 나왔으므로 궤적은 시간 축 범위가 $[t_{enter}, t_{leave}]$인 정적 구간이 되어 질의 처리가 가능하다. 따라서 위와 같이 태그의 궤적을 정적 구간, 동적 구간으로 정의하면 언제나 질의처리가 가능하다.

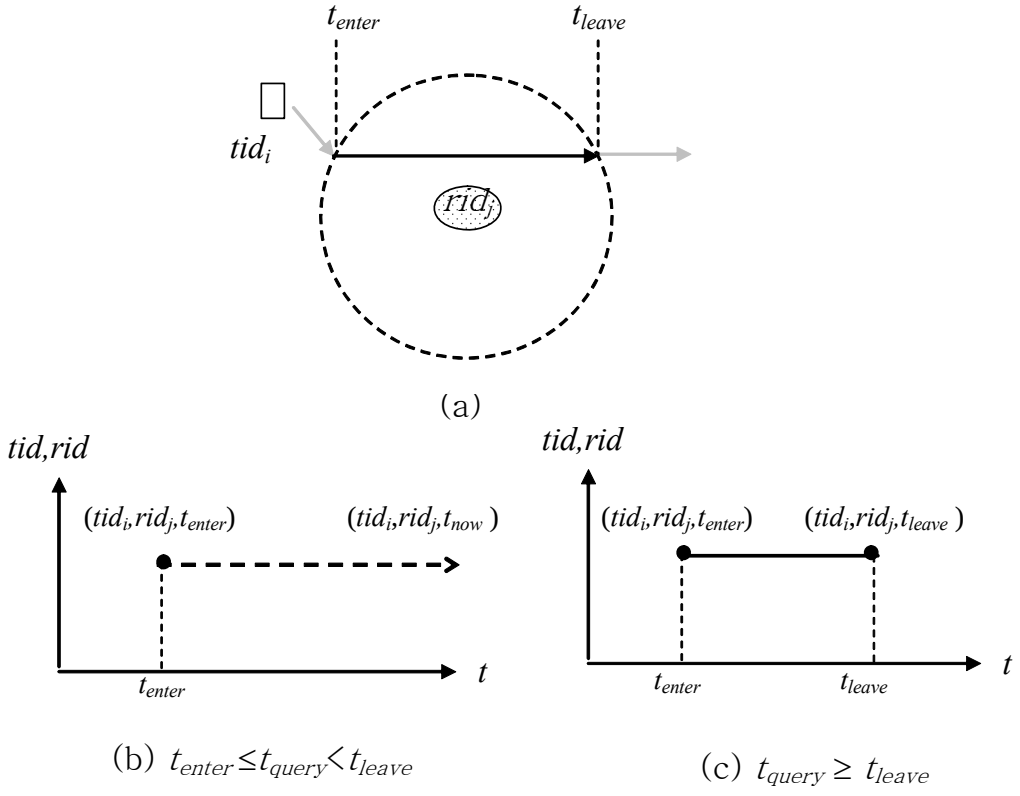
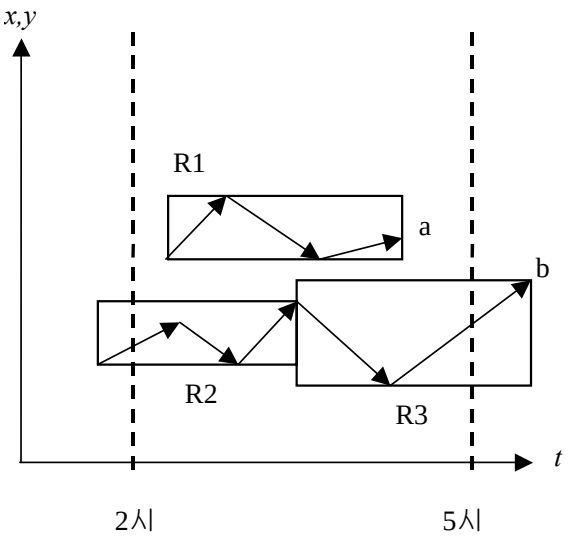


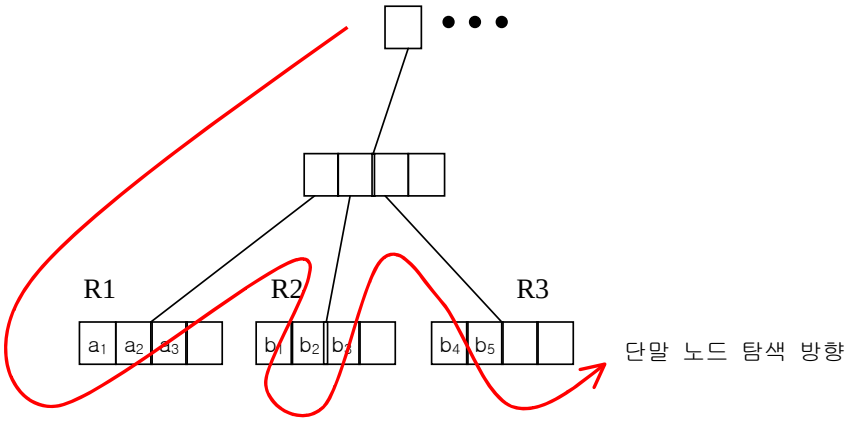
그림 23 정적 구간, 동적 구간의 예

4.3 태그 식별자

기존 이동 객체를 위한 색인들은 객체 식별자를 색인 도메인으로 사용하지 않는다. 대신에 3DR-tree와 같은 경우, 단말 노드의 엔트리에 속성으로서 저장된다. 비단말 노드는 자식 노드에 대한 포인터와 MBB만 가지고 있기 때문에 객체 식별자에 대한 정보를 유지하지 않는다. 따라서 객체 식별자를 인수로 가지는 질의를 처리하기 위해서는 주어진 시간에 포함되는 모든 객체를 탐색하고 객체 식별자로 여과해야 한다. 예를 들어 그림 24-(a)과 같이 이동 객체의 궤적이 분포되어 있을 때 “a 차량이 2시에서 5시에 사이의 위치를 검색하라”와 같은 질의가 주어질 경우, a 차량이 객체 식별자 목록이 되고, 2시에서 5시 사이가 시간 구간이 된다. 3DR-tree에서 이 질의를 처리하기 위해서 그림 24-(b)와 같이 먼저 2차원 전체 공간에서 주어진 시간 구간에 해당하는 객체를 모두 찾는다. 그 다음 결과 집합에서 a 차량을 검색해야 위치를 찾을 수 있다. 이러한 질의 처리는 색인의 전체 노드를 검색해야 하므로 높은 디스크 I/O로 인해 비효율적이다.



(a) 3DR-tree에서 객체 식별자에 관한 질의



(b) 단말 노드 탐색 후 필터링

그림 24 3DR-ree에서 객체 식별자에 관한 질의 처리 방법

마찬가지로 태그의 궤적 색인에서도 위와 같은 문제가 발생한다. 특히, 앞에서 언급한 것과 같이 Find, History, With 질의가 태그의 객체 식별자를 매개

변수로 하기 때문에 이러한 질의 처리 시에 다수의 단말 노드에 접근하므로 매우 비효율적이다. 따라서 이 논문에서는 태그의 구간을 태그의 객체 식별자 차원을 추가하여 표현한다.

t 축이 연속적인(continuous) 데이터를 나타내는 축인 것과 달리, 태그의 식별자는 연속적인 데이터를 나타내는 축이 아니다. 즉, 태그의 식별자 사이의 데이터는 의미를 가지지 않으며 존재할 수 없다. 그러나 태그의 구간은 항상 시간 축 *t*에 평행하므로 태그의 식별자가 비연속적이더라도 표현에는 문제가 발생하지 않는다. 예를 들어 그림 25-(a)와 같이 태그가 판독기를 이동한다고 가정하자. 이동으로 인해 발생되는 구간을 *tid* 축에 투영시키게 되면 모든 궤적은 시간 축에 평행하게 생성된다. 따라서 그림 25-(b)와 같이 태그의 식별자 값을 가지는 선분이 시간 축에 평행하며 그림 25-(c)와 같이 각 식별자 사이를 걸치는 선분이 존재하지 않으므로 태그 식별자가 비연속적이라도 구간을 표현하는데 문제가 발생하지 않는다.

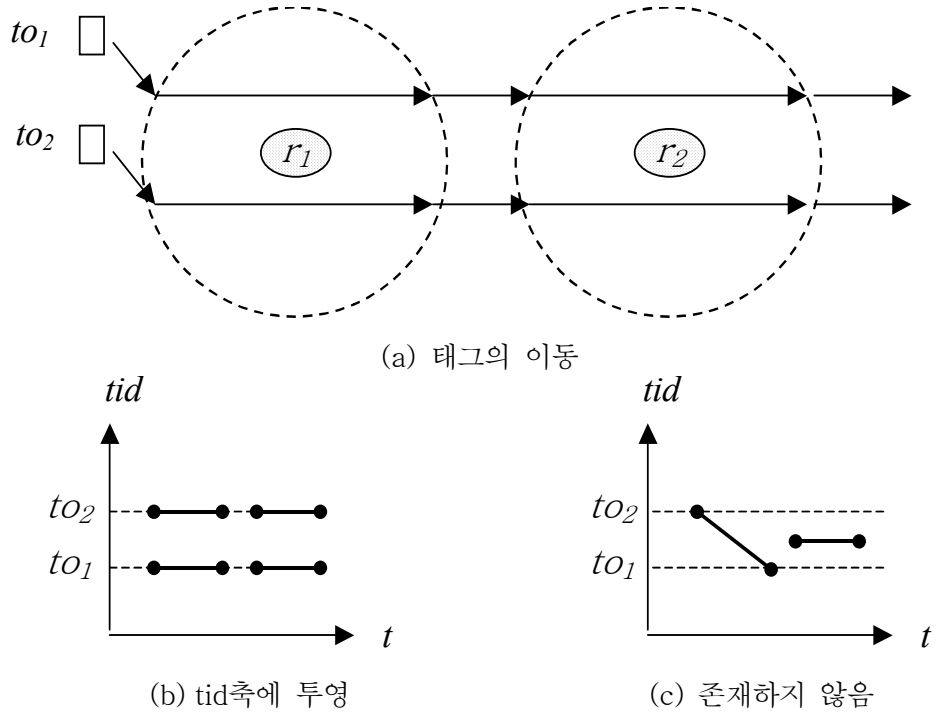


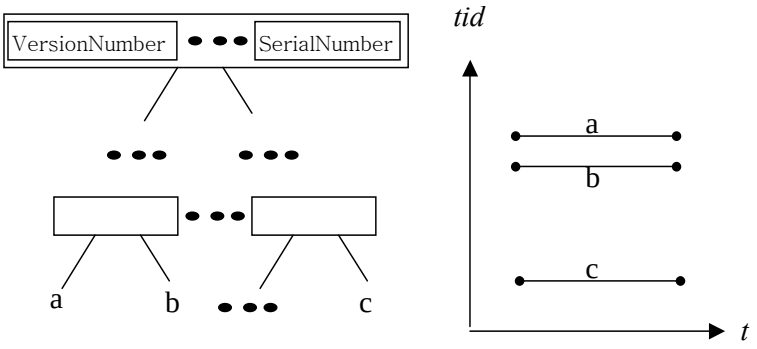
그림 25 태그 식별자의 비연속성

t 축에 존재하는 데이터들간에는 관련성을 가진다. 예를 들면 시간적으로 가까운 데이터들끼리 축에서도 가깝게 위치한다. 그러나 태그 식별자의 경우 축에서 서로 가깝게 위치한다고 해서 식별자끼리 관련성이 크다고는 할 수 없다. 그러나 식별자를 계층적으로 구성하면 관련성을 부여할 수 있다. 예를 들어, EPC(Electronic Product Code)[4]의 경우 계층적으로 구성하여 같은 품목끼리는 EPC를 그룹핑하여 표현한다. 그림 26은 GID-96으로서 태그의 식별자를 96 bit로 표현하는 방법이다. 그림과 같이 태그 식별자는 헤더를 제외하고 회사, 생산품, 시리얼번호 등을 나타내도록 식별자를 구성한다.

	Header	General Manager Number (Company)	Object Class (Product)	Serial Number (Serial)
GID-96	8	28	24	36
	0011 0101 (Binary Value)	268,435,455 (Max decimal)	16,777,215 (Max decimal)	68,719,476,735 (Max decimal)

그림 26 태그 식별자 EPC의 패턴

따라서 이러한 방식으로 태그의 식별자를 구성하면 식별자 축에서 가까운 것들끼리는 같은 품목일 관련성이 높아진다. 따라서 태그 식별자 *tid*축도 위치하는 데이터 간에 관련성을 가진다고 할 수 있다. 예를 들어 그림 27에서와 같이 유사한 품목의 a, b 객체를 EPC로 구성하게 되면 식별자 또한 유사하게 되고, 이를 식별자 축에 대응시키면 가깝게 위치한다. 따라서 이와 같이 식별자를 계층적으로 구성하면 데이터간의 관련성을 표현할 수 있다.



(a) EPC의 구성 (b) tid 축에서의 관련성

그림 27 태그 식별자의 관련성

4.4 구간 데이터 모델의 질의 처리 비용 비교

구간 데이터 모델을 이용하여 질의를 처리하는 경우에 질의 처리 비용을 비교하기 위하여 [29]에서 제시한 비용 모델을 이용하여 비교한다. [29]에서는 점질의 비용을 색인을 구성하는 모든 단말 노드의 면적의 합으로 정의하였으며 영역질의는 점질의의 확장으로 모든 단말 노드를 질의 영역만큼 확장한 후 면적을 합한 것으로 정의하였다. 이 논문에서도 이러한 개념을 사용하여 구간 데이터 모델을 사용할 때와 사용하지 않을때의 질의 처리 비용을 비교한다.

$$P(0,0) = \sum_{i=1}^{N} n_{i,x} \times n_{i,y} \text{ ---------- (1)}$$

$$P(q_x, q_y) = \sum_{i=1}^{N} \left(n_{i,x} + q_x\right) \times \left(n_{i,y} + q_y\right) \text{----(2)}$$

그림 28은 구간 데이터 모델을 사용하지 않은 경우에 질의 처리 비용을 나타낸 것이다. 수식을 단순화 하기 위하여 시간과 판독기 식별자 축으로 나타내었다. 그림과 같이 질의 영역이 q_{rid}, q_t인 경우에 판독기에 머무는 태그를 찾기 위하여 질의 영역을 시간축으로 태그의 평균 구간 길이만큼 확장하여야 질의처리가 가능하다. 따라서 구간 데이터 모델을 사용하지 않는 경우에 질의 처리 비용은 다음과 같다.

$$P(q_{rid}, q_t) = \sum_{i=1}^{N} \left(n_{i,rid} + q_{rid}\right) \times \left(n_{i,t} + q_t + l\right) \text{----} \quad (3)$$

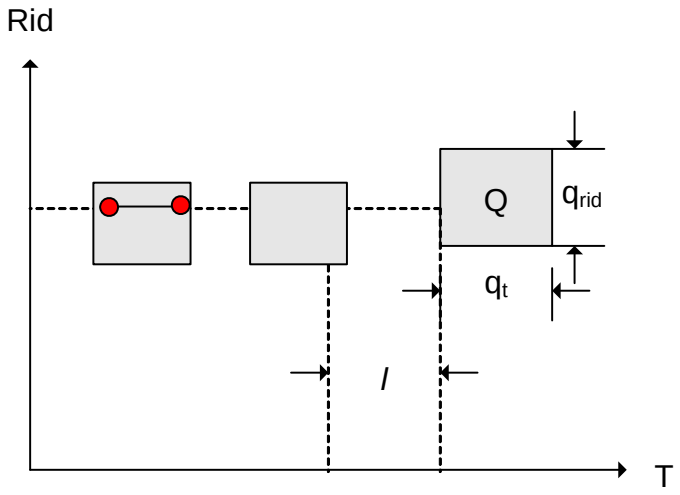


그림 28 구간 데이터 모델을 사용하지 않는 경우 질의 처리 비용

그림 29는 구간 데이터 모델을 사용하여 판독기에 머무는 태그를 동적 구간으로 표현한 경우에 질의 처리 비용을 나타낸 것이다. 그림에서와 같이 동적 구간을 포함하는 노드의 수를 *j*라고 하면 이 노드들은 동적 구간을 포함하므로 *d*만큼 확장되게 된다. 또한 나머지 노드들은 동적 구간을 포함하지 않으므로 확장되지 않는다. 따라서 구간 데이터 모델을 사용하는 경우에 질의 처리 비용은 다음과 같다.

$$P(q_{rid}, q_t) = \sum_{i=1}^{j} \left(n_{i,rid} + q_{rid}\right) \times \left(n_{i,t} + q_t + d\right) + \sum_{i=j}^{N} \left(n_{i,rid} + q_{rid}\right) \times \left(n_{i,t} + q_t\right) -- \quad (4)$$

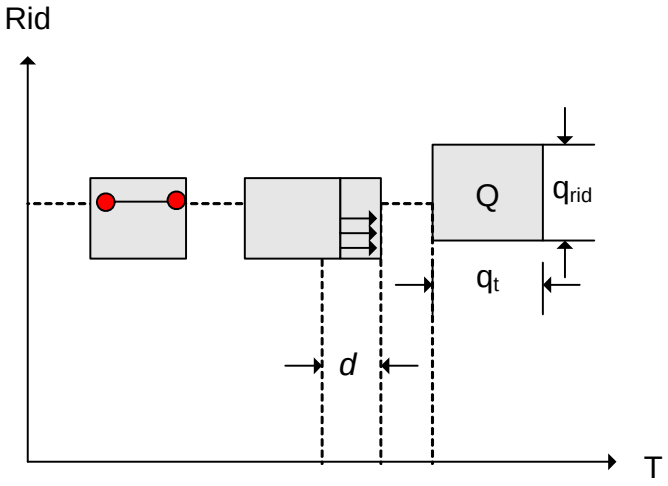


그림 29 구간 데이터 모델을 사용하는 경우 질의 처리 비용

수식 (4)에서 (3)을 감산하면

$$(4)-(3)$$

$$=\sum_{i=1}^{j}\left(n_{i,rid}+q_{rid}\right)\times\left(n_{i,t}+q_t+d\right)+\sum_{i=j}^{N}\left(n_{i,rid}+q_{rid}\right)\times\left(n_{i,t}+q_t\right)$$

$$-\sum_{i=1}^{N}\left(n_{i,rid}+q_{rid}\right)\times\left(n_{i,t}+q_t+l\right)$$

$$=\sum_{i=1}^{j}\left(n_{i,rid}+q_{rid}\right)\times\left(n_{i,t}+q_t+d\right)+\sum_{i=j}^{N}\left(n_{i,rid}+q_{rid}\right)\times\left(n_{i,t}+q_t\right)$$

$$-\sum_{i=1}^{j}\left(n_{i,rid}+q_{rid}\right)\times\left(n_{i,t}+q_t+l\right)+\sum_{i=j}^{N}\left(n_{i,rid}+q_{rid}\right)\times\left(n_{i,t}+q_t+l\right)<0$$

$$\because d\le l,\ \ l>0$$

따라서 구간 데이터 모델을 사용하여 판독기에 머무는 태그의 궤적을 동적 구간으로 표현하는 경우에 질의 처리 비용이 적음을 수식을 통해 알 수 있다.

5. IR-tree(Interval R-tree)

이 장에서는 태그의 궤적을 위한 R-tree 기반의 IR-tree를 소개한다. 먼저 4장에서 제안된 구간을 이용하여 색인에서 사용되는 데이터구조와 탐색 알고리즘을 제시한다. 또한 질의의 효율적인 처리를 위해 구간의 특성을 고려한 새로운 삽입과 분할 알고리즘을 제시한다.

5.1 데이터 구조

IR-tree의 단말 노드는 <*MBB*>의 형태의 구간을 엔트리로 포함한다. *MBB*는 3차원 최소경계박스로서 기존 R-tree 계열의 색인에서는 그림 30-(b)와 같이 <[$tid^{\vdash}$,$tid^{\dashv}$],[$rid^{\vdash}$,$rid^{\dashv}$],[$t^{\vdash}$,$t^{\dashv}$]>의 형태를 갖는다. 그러나 IR-tree에서는 시간 구간 [$t^{\vdash}$, $t^{\dashv}$]에서 식별자와 공간 위치가 변하지 않으므로 그림 30-(a)와 같이 <*tid, rid,* [$t^{\vdash}$,$t^{\dashv}$]>의 형태를 가진다.

만약 태그가 판독기에 들어와 머무는 경우에 구간은 시간의 시작 값만이 보고되고 시간의 끝 값은 보고되지 않았으므로 시간 축 길이가 시간에 종속적이다. 따라서 이러한 구간은 <*tid, rid,* t_{enter}, *now*>의 형태로 단말 노드에 삽입되며 동적 구간(dynamic interval)이라고 하며 이 논문에서는 *dI*로 표시한다. 만약 태그가 판독기에 들어와 나가는 경우에 판독기에 들어온 시간 값과 나간 시간 값이 모두 보고되므로 구간은 <*tid, rid,* t_{enter}, t_{leave}>의 형태로 단말 노드에 삽입되며 정적 구간(static interval)이라고 하며 *sI*라 표기한다.

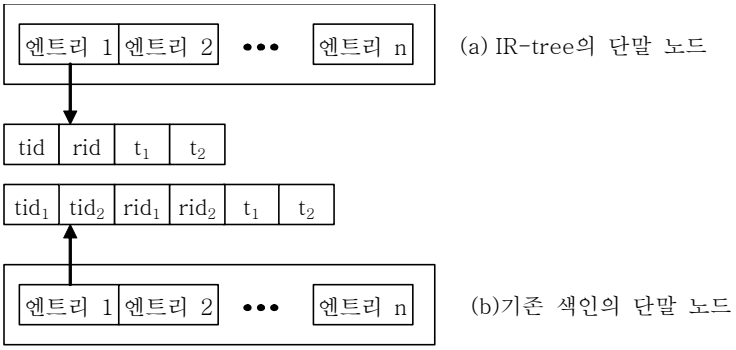


그림 30 IR-tree의 단말 노드 구성

예를 들어, 그림 31과 같이 #3번 물건이 창고 #7에 5:00에 들어와 7:00에 나갔다고 가정하자. 물건이 창고에 5:00에 들어오면 (#3, #7, 5:00, now)가 보고되고 이 구간이 단말 노드에 삽입된다. 이 구간은 시간의 끝 값이 아직 보고되지 않고 시간에 종속적이므로 *dI*가 되고 단말 노드에 삽입된다. 그 후, 물건이 창고에서 7:00에 나가면 (#3, #7, 5:00, 7:00)가 보고되고 이 구간이 단말 노드에 삽입된다. 이 경우에는 이미 삽입된 *dI*를 제거하고 *sI*를 삽입하게 된다.

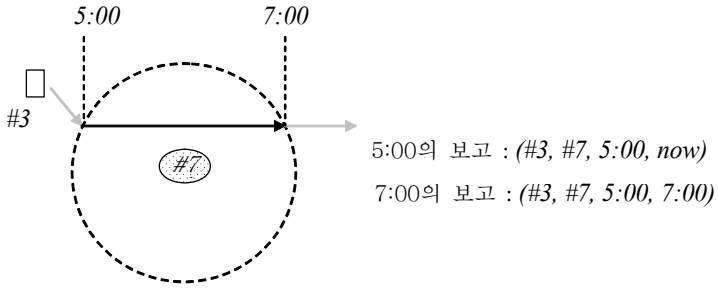


그림 31 단말 노드 엔트리의 예

비단말 노드는 <*child-pointer*, *state*, *MBB*>의 형태를 갖는 엔트리를 포함한다.

여기서 *child-pointer*는 자식 노드를 가리키는 포인터, *state*는 엔트리의 상태를 나타내며 *MBB*는 <[$tid^{\vdash}$,$tid^{\dashv}$],[$rid^{\vdash}$,$rid^{\dashv}$],[$t^{\vdash}$,$t^{\dashv}$]> 형태의 3차원 최소경계박스를 나타낸다.

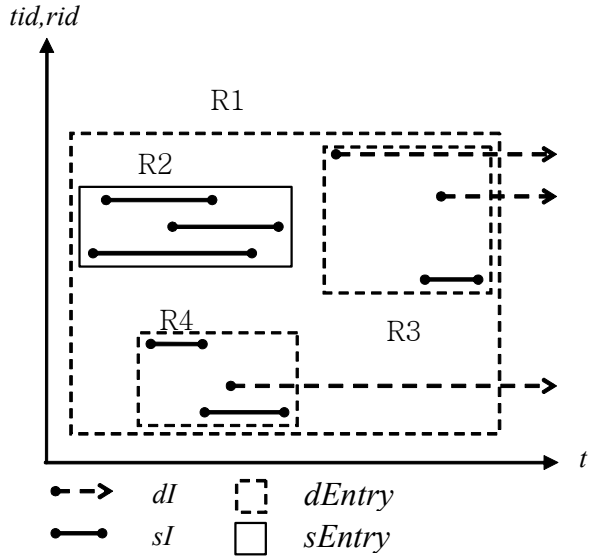
엔트리의 상태 *state*는 s*Entry*와 *dEntry*의 두 가지로 분류된다. 먼저 *sEntry*는 자식 노드가 포함하는 엔트리가 모두 s*Entry* 이거나 *sI* 인 경우이며 *MBB*는 기존 방법과 동일하게 자식 노드의 모든 엔트리들을 포함하는 두 개의 3차원 시공간 점으로 구성된다.

자식 노드가 포함하는 엔트리가 하나 이상의 *dEntry*나 *dI*인 경우에 *state*는 *dEntry*가 된다. 이 경우에 단말 노드에서의 *MBB*는 s*I* 선분 전체와 *dI*의 시작점만을 포함하는 영역을 가지게 된다. *dI*의 시작점만을 포함하는 이유는 시간 구간이 정해지지 않았기 때문이다. 따라서 이러한 *dEntry*인 단말 노드는 질의 처리 시에 현재 시간 값으로 시간 구간을 확장해서 처리해야 한다. 이것은 뒷절에서 자세히 설명한다.

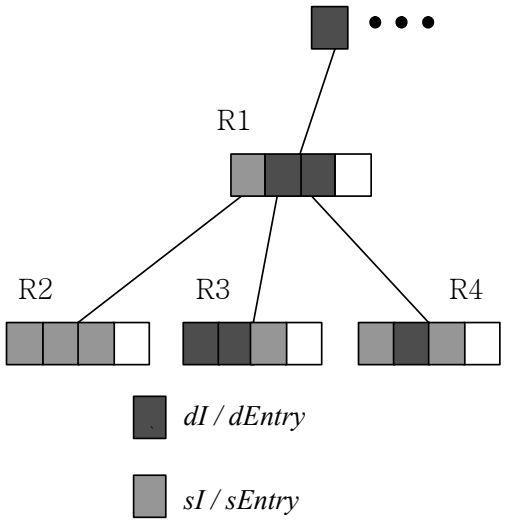
비단말 노드가 *dEntry*인 경우에 *MBB*는 *dEntry*와 *sEntry*를 포함하는 영역이 된다. 이러한 *MBB*는 하위 노드에 *dEntry*나 *dI*를 포함하므로 질의 처리 시에 현재 시간 값으로 시간 구간을 확장해서 처리해야 한다.

예를 들어 그림 32-(a)와 같이 3차원 공간에 s*I*와 *dI*가 존재한다고 가정하자. 단말 노드 R2는 그림 32-(a)와 같이 세 개의 *sI*를 포함하므로 이것은 *sEntry*가 되며 모든 *sI*를 포함하는 *MBB*를 가지게 된다. 따라서 부모 노드 R1에서 R2를 가리키는 엔트리의 *state*는 *sEntry*가 된다. R3는 하나의 *sI*와 두 개의 *dI*를 포함하므로 *dEntry*가 되며 이것의 *MBB*는 *sI* 전체와 각 *dI*의 시작점

만을 포함하게 된다. 이와 같은 방식으로 R4와 R1은 모두 d*Entry*가 된다.



(a) 3차원 공간의 구간



(b) IR-tree의 구조

그림 32 IR-tree의 구성 예

5.2 탐색

탐색은 기존의 R-tree와 유사하게 색인의 루트에서부터 하위방향으로 탐색해 나간다. 그러나 IR-tree에서는 노드의 *state* 값에 따라 탐색 방법이 달라지게 된다. 먼저 단말 노드가 나올 때까지 *sEntry*의 경우에는 기존 탐색과 동일하게 *MBB*가 질의 영역과 교차하면 하위 노드로 탐색을 수행한다. 그러나, *dEntry*의 경우에는 먼저 *MBB*의 시간 축을 현재 시간인 *now*까지 확장하여 질의 영역과 교차하는지를 검사하여 교차하면 하위 노드로 탐색을 수행한다.

단말 노드가 발견되면 포함된 *sI*와 *dI*를 검사하여 질의의 결과로 반환하게 된다. 이 경우에도 *dI*는 그 시간 축을 현재 시간인 *now*까지 확장하여 질의 영역과 교차하는지를 검사한다. 이처럼 *dEntry*와 *dI*의 시간 축을 확장하는 이유는 IR-tree에서 단말 노드를 구성할 때 *dI*의 경우 시작점만으로 노드를 구성하기 때문이다. 따라서 질의 처리 시에는 이러한 *dI*와 이것을 포함하는 *dEntry*를 현재 시간 값으로 확장하여 질의를 처리해야 한다. 다음은 탐색 알고리즘이다.

Algorithm Search(Node *root*, Query *q*)

```
1    Set N to be the root
2    IF N is a leaf
3        FOR EACH entry e
4        IF e is sI AND e intersects q
```

```
5           return e
6       ELSE IF e is dI AND extension of e intersects q
7           rerurn e
8   ELSE
9       FOR EACH entry e
10      IF e is sEntry AND e intersects q
11          Search(e, q)
12      ELSE IF e is dEntry AND extension of e intersects q
13          Search(e, q)
```

알고리즘 1 Search()

예를 들어, 그림 33과 같이 IR-tree가 구성되고 질의 Q를 수행한다고 가정하자. 먼저 노드 R1을 검사하는데 R1은 질의 영역 Q와 교차하므로 하위 노드로 탐색을 수행하게 된다. 하위 노드인 R2, R3, R4를 차례로 검사하는데 R2는 질의 영역과 교차하지 않으므로 탐색에서 제외한다. R3의 경우에 *MBB*는 질의 영역 Q와 교차하지 않지만 R3의 상태가 *dEntry*이므로 *MBB*를 확장하여 질의 영역과 교차하는지를 검사하게 된다. 이 경우에 R2는 교차하므로 하위 노드로 탐색을 수행하게 된다. R4의 경우 질의 영역과 교차하지 않으므로 탐색에서 제외한다. 최종적으로 R3에 포함되는 세 개의 엔트리를 검사하게 되며, 하나의 *dI*가 확장되어 질의 영역과 교차하게 되므로 질의의 결과로 반환된다.

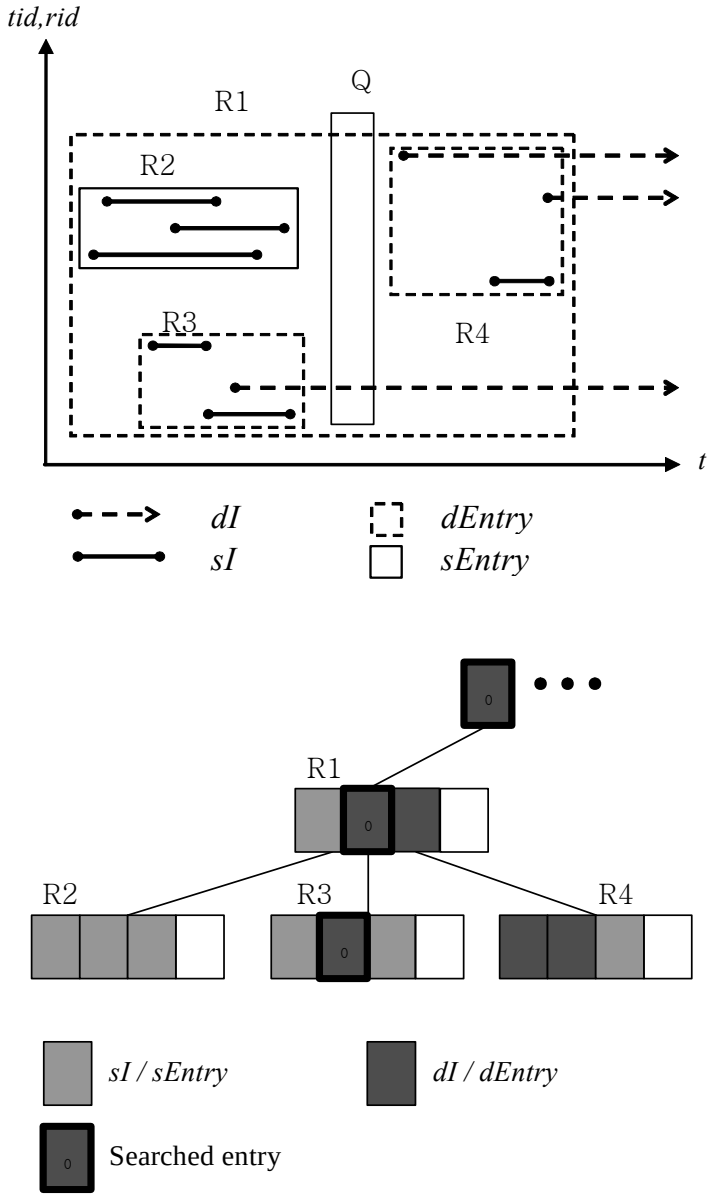


그림 33 IR-tree 탐색의 예

5.3 삽입

이 논문에서는 태그를 위한 질의 처리를 효율적으로 수행하기 위하여 기존의 R-tree 계열[6][7]에서 적용하는 삽입 방법을 개선한 알고리즘을 제시한다. 먼저 IR-tree에서 구간의 삽입은 다음과 같이 새로운 구간이 들어갈 단말 노드를 찾는 *ChooseSubtree()*와 구간을 삽입하는 *InsertDataImpl()*, 삽입 후의 노드 *MBB*를 조절하고 분할을 상위 노드로 전파하는 *AdjustTree()* 알고리즘으로 구성된다.

Algorithm InsertData(Interval I_{new})

---

1 Get root node *rootNode*

2 Node *node* := *ChooseSubtree*(*rootNode*, I_{new})

3 InsertDataImpl(*node*, I_{new})

알고리즘 2 InsertData()

구간을 노드에 단말 노드에 삽입하는 *InsertDataImpl()*의 경우 알고리즘 3와 같이 먼저 단말 노드에 삽입할 공간이 있으면 삽입 후 조정에 들어가게 된다. 그러나 삽입할 공간이 없다면 노드를 분할하는 *SplitNode*()을 적용하여 노드를 분할하고 그 후에 조정에 들어가게 된다. 노드를 분할하는 *SplitNode*()는 다음 절에서 자세히 설명한다.

Algorithm InsertDataImpl(Node *node*, Interval I_{new})

```
1  IF node have a room
2    Insert I_new into node
3    IF node is not root node
4      AdjustTree(node)
5  ELSE
6    SplitNode(Node node, Node l, Node r)
7    IF node is root node
8      Create a new root node whose children are l and r
9    AdjustTree(l, r)
```

알고리즘 3 InsertDataImpl()

기존 R-tree에서 사용된 *AdjustTree()* 알고리즘은 삽입으로 인해 노드의 *MBB*가 변경됨에 따라 노드를 참조하는 상위 엔트리의 *MBB*를 조정한다. 그러나 IR-tree에서는 *MBB*의 조정뿐만 아니라 엔트리의 상태도 조정하게 된다. 즉, 노드가 하나 이상의 *dI*나 *dEntry*를 포함하는 경우에 알고리즘 4과 같이 상위 엔트리의 상태는 *dEntry*로 조정된다.

```
Algorithm AdjustTree(Node n)
1  Get parent node p of node n
2  Find entry e in p points to n
3  IF p.MBB do not contains n.MBB OR p.MBB touches e.MBB
4    Adjust p.MBB so that it tightly encloses all entry rectangles in p
5    Adjust e.state considering the state of n
```

```
6   IF adjusted AND p is not root node
7     AdjustTree(p)
```

```
Algorithm AdjustTree(Node l, Node r)
1   Get parent node p of node l
2   InsertDataImpl(p, r.cp, r.state, r.MBB)
3   Find entry e in p points to l
4   IF p.MBB do not contains l.MBB OR p.MBB touches e.MBB
5     Adjust p.MBB so that it tightly encloses all entry rectangles in p
6     Adjust e.state considering the state of l
7   IF adjusted AND p is not root node
8     AdjustTree(p)
```

알고리즘 4 AdjustTree()

R-tree에서의 *ChooseSubtree()*는 최소영역확장(least area enlargement) 정책을 이용하여 수행하게 된다. 그러나 IR-tree에 이러한 정책을 사용하게 되면 *dEntry*가 질의 처리 시에 확장되는 특성을 고려하지 않기 때문에 노드들의 면적과 겹침이 증가되어 질의 처리가 비효율적이다.

예를 들어 그림 34와 같이 새로운 *dI*를 삽입하기 위하여 색인의 루트로부터 삽입하고자 하는 엔트리를 선택한다고 가정하자. 기존 방법에서는 그림 34-(a)와 같이 삽입의 대상이 되는 엔트리를 선택하기 위하여 최소면적확장 정책을 사용하여 삽입 후에 면적의 증가가 최소인 엔트리 ②을 선택하게 된다. 이 경우에 엔트리 ②는 *dI*의 삽입으로 인하여 상태가 *sEntry*에서 *dEntry*로 변경되며, 그림 34-(b)와 같이 질의 처리 시에 2개의 엔트리가 확장되고 접근

된다.

그러나 그림 34-(c)와 같이 삽입되는 구간의 종류와 삽입의 대상이 되는 엔트리의 상태를 고려하여 비록 삽입 후에 증가되는 면적이 크더라도 엔트리의 상태가 변하지 않는 엔트리 ①에 구간을 삽입한다고 가정하자. 이 경우에는 그림 34-(d)와 같이 질의 처리 시에 단 한번의 엔트리의 확장과 접근으로 질의를 처리할 수 있게 된다. 따라서 엔트리의 선택 시 선택된 엔트리의 상태가 변하지 않도록 삽입을 수행해야 한다.

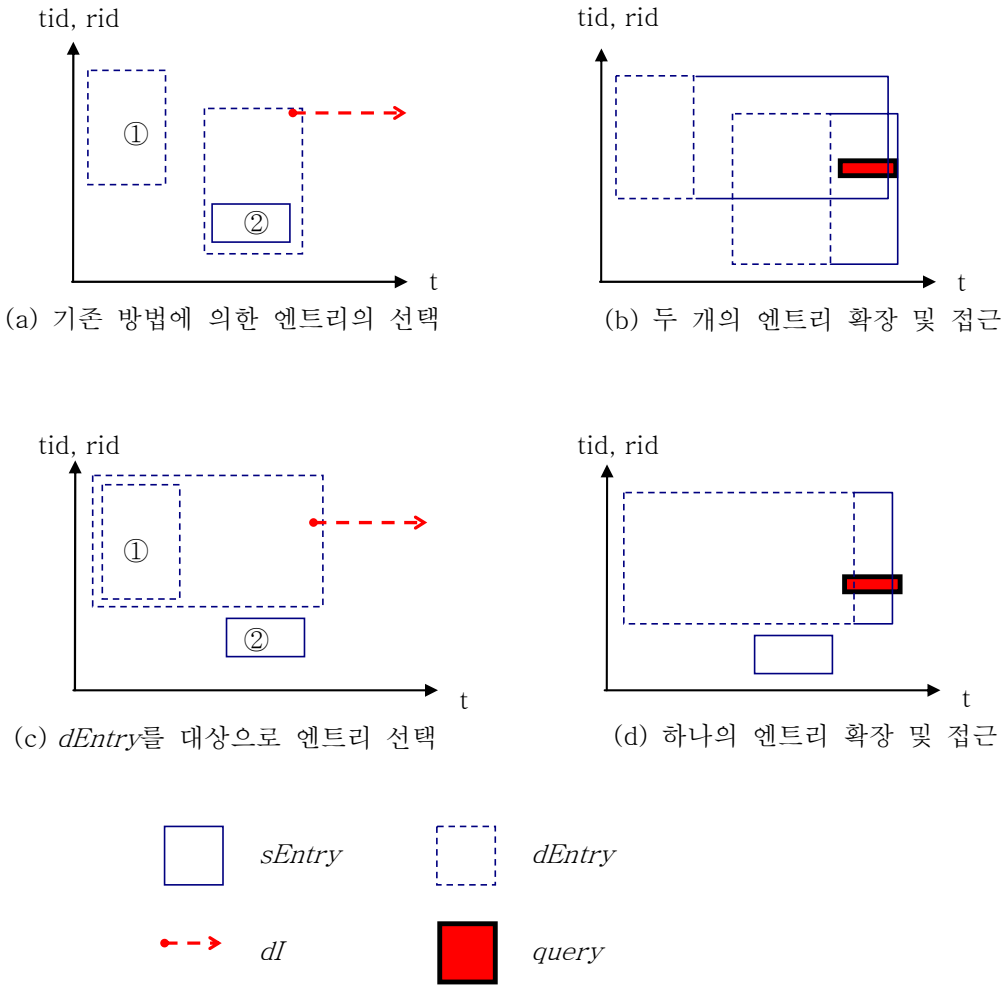


그림 34 확장된 면적만을 고려한 삽입 방법의 문제점

그러나 엔트리의 상태 변화를 배제한 방법을 사용하더라도 질의 처리 시에 다수의 엔트리에 접근하는 문제가 발생한다. 예를 들어 그림 35-(a)와 같이 비록 확장되는 면적이 크더라도 삽입하고자 하는 엔트리의 상태를 변화시키지 않기 위하여 엔트리 ①을 선택하는 경우에 엔트리 간의 중복이 심해지며 질의 처리 시 다수의 엔트리에 접근해야 한다. 이 경우에 그림 35-(c)와

같이 확장되는 면적이 최소가 되도록 엔트리를 선택하는 기존의 방법을 사용한다면 그림 35-(d)와 같이 질의 처리 시에 하나의 엔트리만을 접근하여 처리할 수 있다.

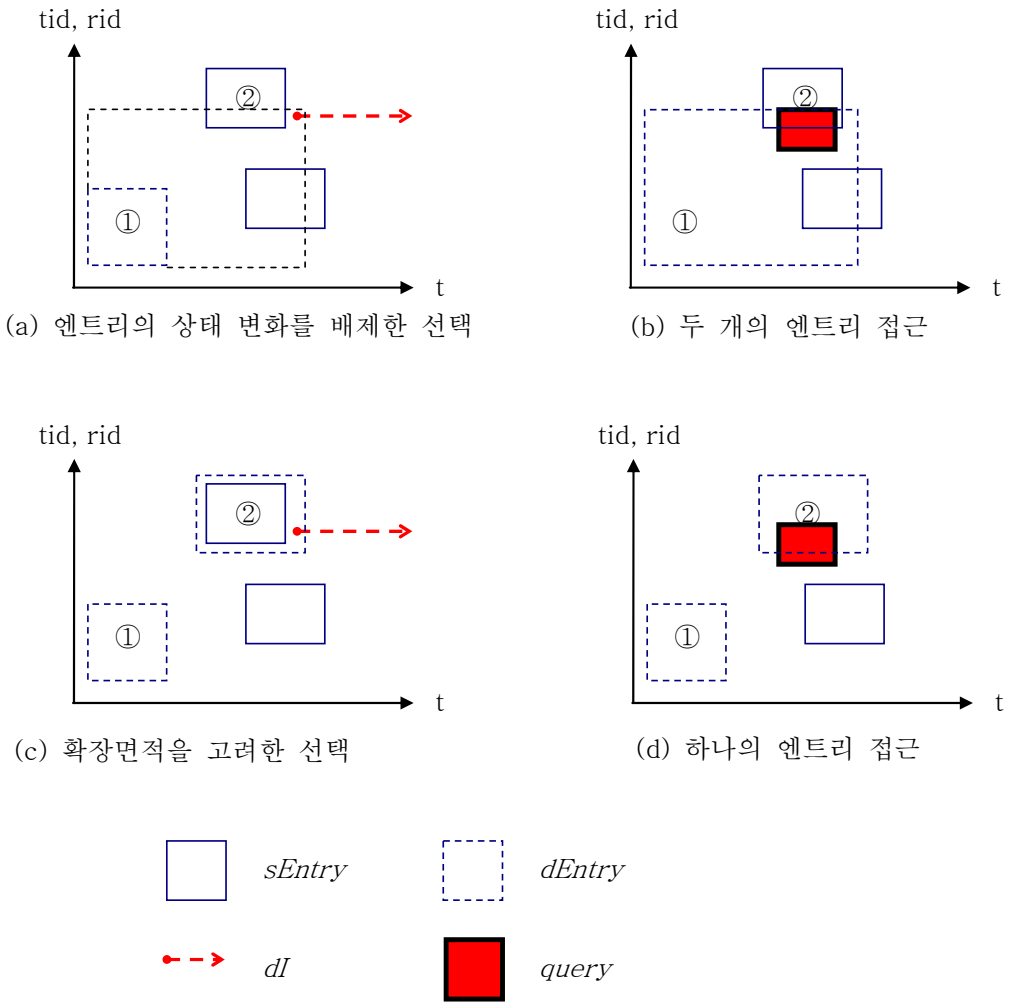
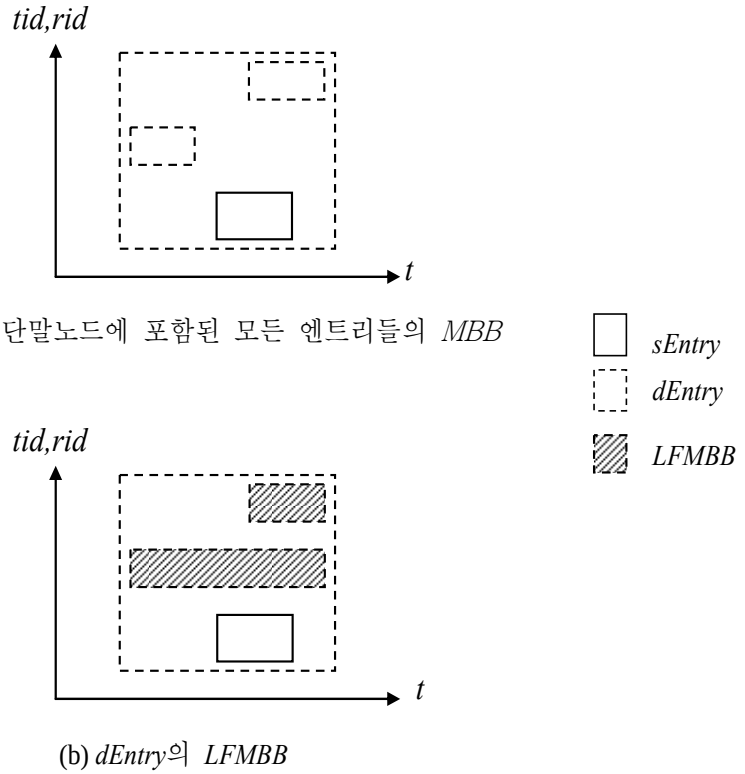


그림 35 노드 상태 변화를 배제한 삽입 방법의 문제점

따라서 구간을 삽입하는 경우에 첫째, 삽입되는 구간의 종류, 둘째 삽입의

대상이 되는 엔트리의 속성 변화, 마지막으로 삽입으로 인해 확장된 엔트리의 면적을 동시에 고려한 삽입 방법이 필요하다. 이를 위하여 이 논문에서는 먼저 *Local Fixed MBB(LFMBB)*를 아래와 같이 정의한다.

정의 4: *dEntry*의 *LFMBB* 는 *dEntry*를 포함하는 노드가 가지는 최대 시간 값으로 *dEntry*의 시간 값을 고정시킨 *MBB*



(a) 비단말노드에 포함된 모든 엔트리들의 *MBB*

(b) *dEntry*의 *LFMBB*

그림 36 *LFMBB*의 예

LFMBB 의 예는 그림 36과 같다. 그림 36-(a)와 같이 비단말 노드에 포함된 모든 엔트리들의 *MBB* 중에서 *dEntry*들의 *MBB*를 그림 36-(b)와 같이 포함하는 노드의 시간 축 최대값으로 고정시킨 것을 *LFMBB*라고 한다. 이것은 *dEntry*가 확장되는 성질을 일정으로 보장해 주므로 삽입 수행 시 이러한 *LFMBB*를 사용하여 최소영역확장 방법을 사용한다.

표 1은 삽입되는 구간의 타입에 따라 삽입 시 선택되는 엔트리들의 영역 확장 계산 방법을 제시하고 있다. 표에서 *Area( )*는 영역을 계산하는 함수이며 *MBB'*와 *LFMBB'*는 각각 삽입되는 구간을 포함한 후에 엔트리들의 *MBB*와 *LFMBB*를 나타낸다. *sI*가 삽입되는 경우 *sEntry*의 영역확장 계산은 기존 알고리즘과 동일하게 삽입 후의 *MBB* 면적에서 삽입 전 *MBB*를 감산하여 계산한다. 그러나 *dEntry*의 경우에는 앞에서 언급한 것처럼 *LFMBB*를 사용하여 삽입 후의 *LFMBB* 면적에서 삽입 전 *LFMBB*를 감산하여 계산한다.

표 1 엔트리들의 영역확장

삽입되는 구간	삽입시 선택되는 엔트리	영역확장
sI	*sEntry*	*Area(MBB') − Area(MBR)*
	dEntry	*Area(LFMBB') − Area(LFMBB)*
dI	*sEntry*	*Area(LFMBB') − Area(MBB)*
	dEntry	*Area(LFMBB') − Area(LFMBB)*

예를 들어, 그림 37-(a)와 같이 *sI*가 삽입되는 경우에 삽입 대상이 되는 엔

트리를 선정한다고 가정하자. 먼저 엔트리가 그림 37-(b)와 같이 *sEntry*인 경우에 삽입 후의 *MBB*(①+②)에서 삽입 전의 *MBB*(②)를 감산한 면적이 영역확장 값이 된다. 그러나 그림 37-(c)와 같이 *dEntry*의 경우에는 원래의 *MBB*(②)가 *LFMBB*(②+③)로 확장된 후에 계산이 되므로 삽입 후의 *LFMBB*(①+②+③)에서 삽입 전의 *LFMBB*(②+③)를 감산한 면적이 영역확장 값이 된다.

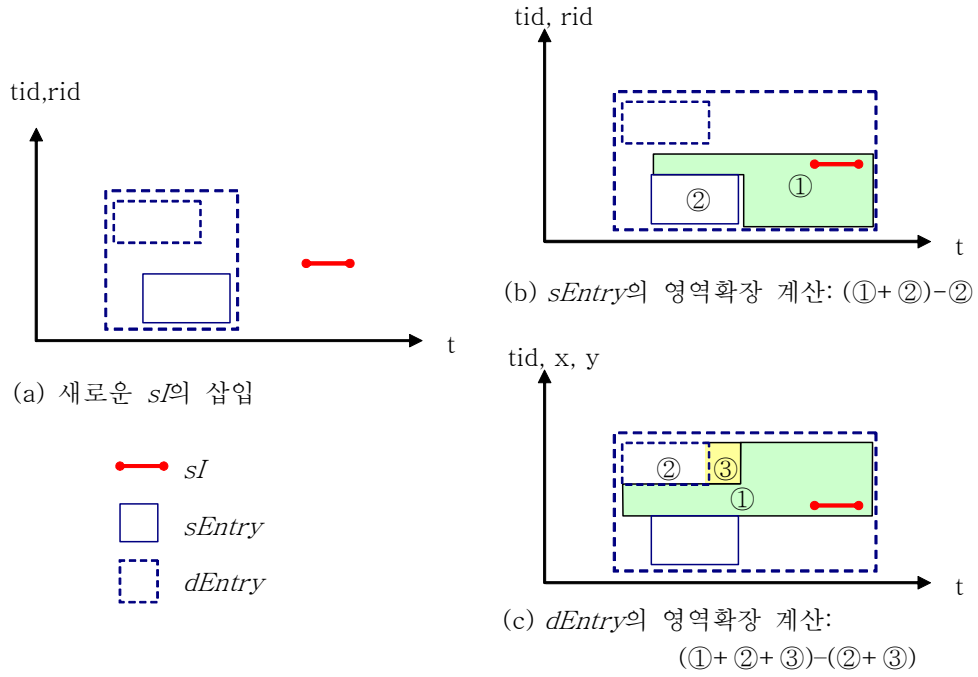


그림 37 *sI* 삽입 시 엔트리의 상태에 따른 영역확장의 계산

*dI*가 삽입되는 경우에 선택되는 엔트리가 *dEntry*이라면 삽입되는 구간으로 인해 엔트리의 상태가 변하게 되므로 삽입 후의 *LFMBB*에서 삽입 전의 *MBB*를 감산하여 계산한다. *dEntry*의 경우에는 *sI*가 삽입되는 경우와 동일하다.

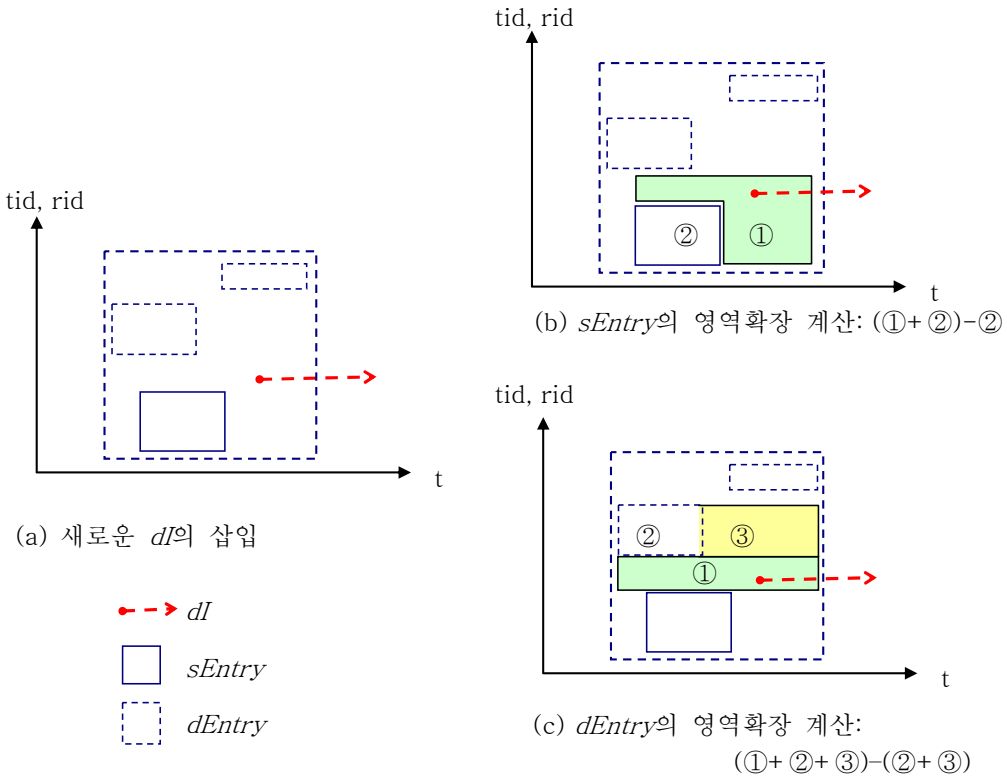


그림 38 *dI* 삽입 시 엔트리의 상태에 따른 영역확장의 계산

예를 들어, 그림 38-(a)와 같이 *dI*가 삽입되는 경우에 삽입 대상이 되는 엔트리를 선정한다고 가정하자. 먼저 엔트리가 그림 38-(b)와 같이 *sEntry*인 경우에 삽입 후에는 *dEntry*로 상태가 변하므로 삽입 후의 *LFMBB*(①+②)에서 삽입 전의 *MBB*(②)를 감산한 면적이 영역확장 값이 된다. 그러나 그림 38-(c)와 같이 *dEntry*의 경우에는 원래의 *MBB*(②)가 *LFMBB*(②+③)로 확장된 후에 계산이 되므로 삽입 후의 *LFMBB*(①+②+③)에서 삽입 전의 *LFMBB*(②+③)를 감산한 면적이 영역확장 값이 된다.

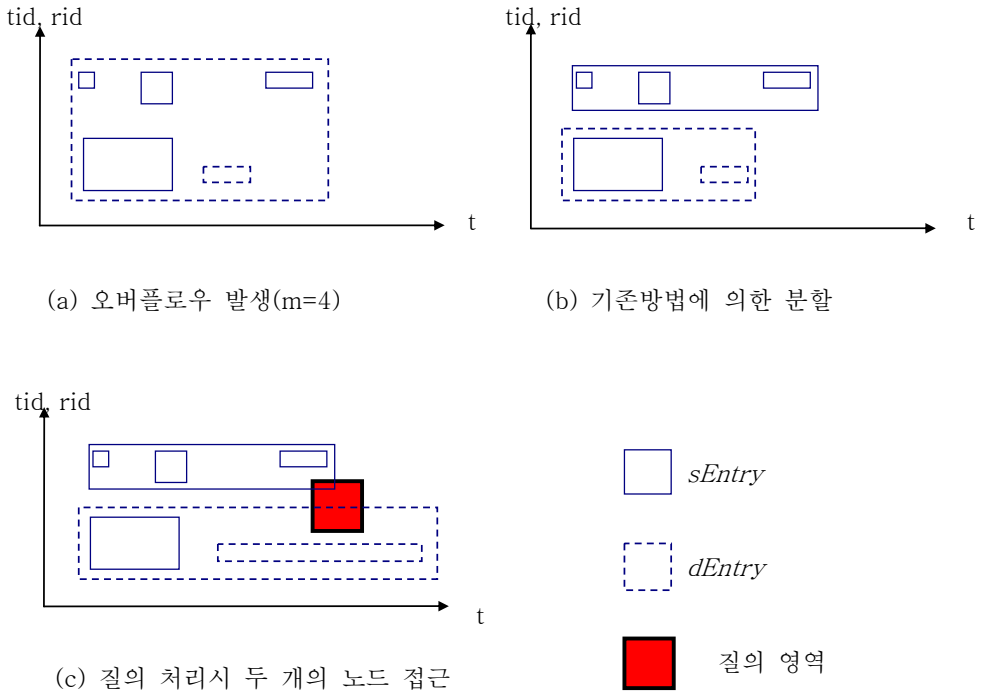
위의 영역확장 계산 방법을 적용한 *ChooseSubtree()* 알고리즘은 아래와 같다. 1행과 같이 단말 노드가 나올 때까지 노드를 하향으로 탐색한다. 각 노드에서 삽입되는 구간과 엔트리의 상태에 따라 새로운 영역확장 값을 계산하여 최소 값을 가지는 엔트리를 선택한다. 그리고 6행과 같이 선택된 엔트리가 참조하는 자식 노드로 탐색을 계속하게 된다.

```
Algorithm ChooseSubtree(Node n, Interval I_new)
1   IF n is leaf
2     Return n
3   ELSE
4     Find entry e in n whose MBB needs new least area enlargement to include
5     I_new
6   Get child node child of node e
7   ChooseSubtree(child, I_new)
```

알고리즘 5 ChooseSubtree()

5.4 분할

노드에 오버플로우가 발생하게 되면 기존 방법에서는 마진을 사용하여 분할 축을 설정하고 겹침이 최소화되도록 노드를 분할하였다[7]. 그러나 IR-tree에 이러한 정책을 사용하게 되면 *dEntry*와 *dI*가 질의수행 시 확장되는 특성을 고려하지 않았기 때문에 노드들 간에 겹침이 증가되어 비효율적이다.



(a) 오버플로우 발생(m=4)

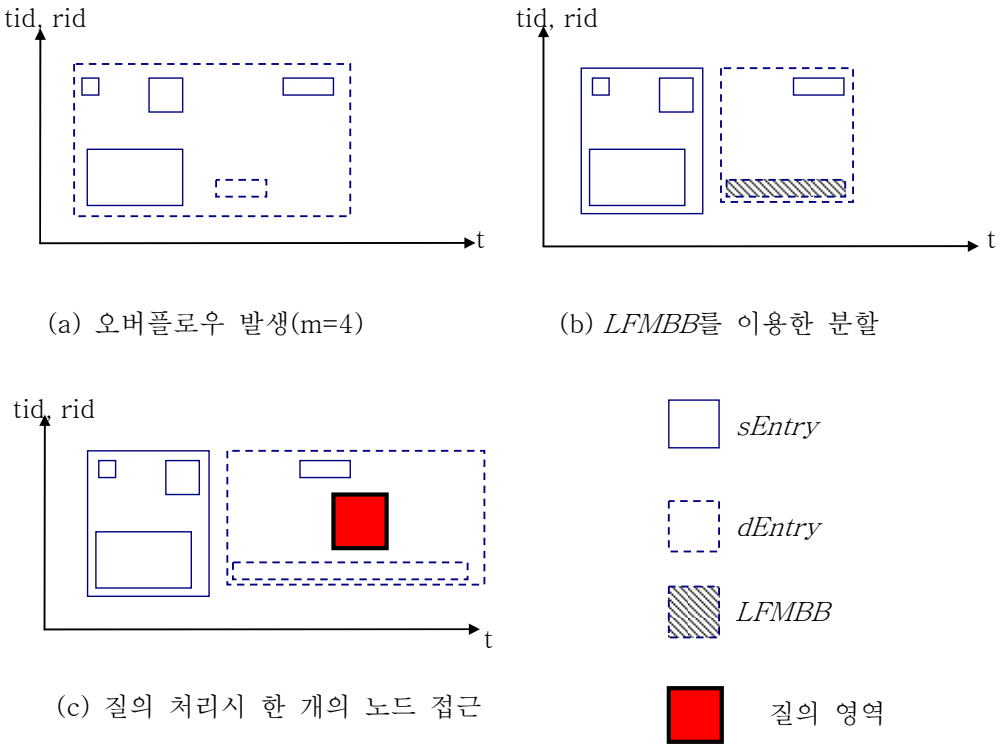
(b) 기존방법에 의한 분할

(c) 질의 처리시 두 개의 노드 접근

그림 39 기존 방법에 의한 비단말 노드 분할의 문제점

먼저 비단말 노드의 분할의 경우에 기존 방법을 사용하면 질의 처리 시

다수의 노드에 접근하는 문제가 발생한다. 예를 들어, 그림 39-(a)와 같이 적재용량이 4인 노드에 오버플로우가 발생했다고 가정하자. 이 경우에 기존 방법에서는 그림 39-(b)와 같이 분할되고 그림 39-(c)와 같은 질의 처리 시에 두 개의 노드에 접근하게 된다. 이것은 노드에 포함된 *dEntry*가 질의 처리 시에 확장된다는 특성을 고려하지 않았기 때문이다.



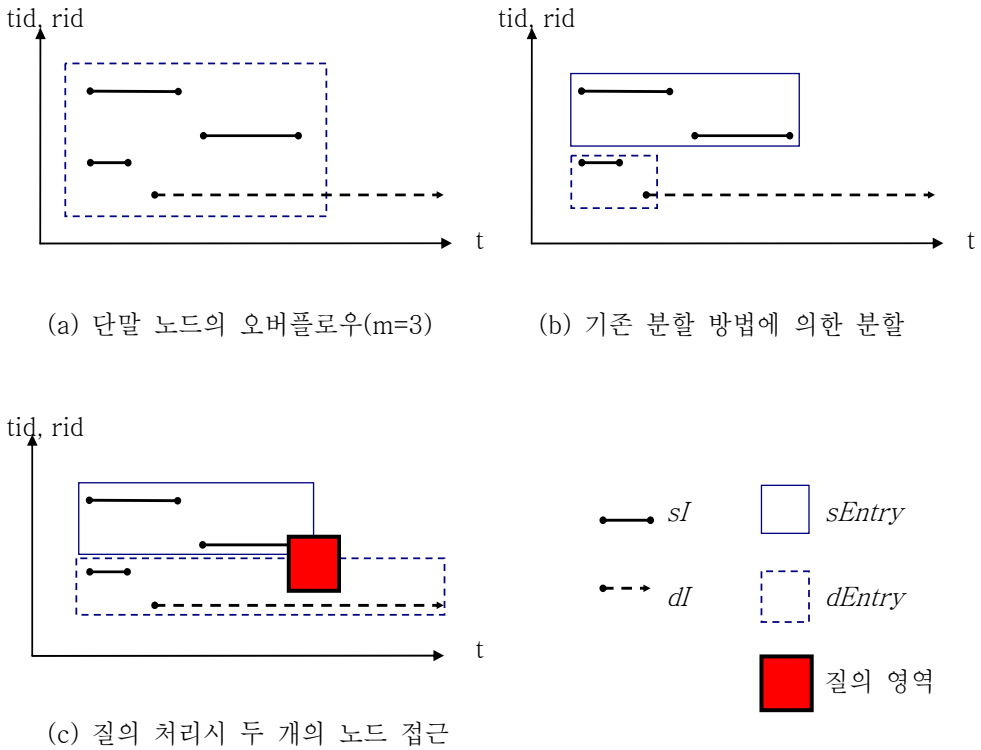
(a) 오버플로우 발생(m=4)　　(b) *LFMBB*를 이용한 분할

(c) 질의 처리시 한 개의 노드 접근

그림 40 *LFMBB*를 이용한 비단말 노드의 분할

따라서, 비단말 노드의 분할 시에 이러한 문제를 해결하기 위하여 앞절에서 삽입 시 사용했던 개념인 *LFMBB*를 사용한다. *LFMBB*는 *dEntry*가 확장된

다는 성질을 일정 보장해 주므로 비단말 노드의 분할 시에 *dEntry*의 *LFMBB*를 사용하여 기존 R*-tree에서의 분할 방법과 동일한 방법으로 비단말 노드를 분할한다. 예를 들어, 그림 40-(a)와 같이 오버플로우가 발생하는 경우에 *dEntry*는 *LFMBB*로 확장하고 이것을 사용하여 그림 40-(b)와 같이 비단말 노드를 분할한다. 이 방법을 사용하면 *dEntry*가 확장된다는 성질을 고려하므로 질의를 효율적으로 처리할 수 있다.



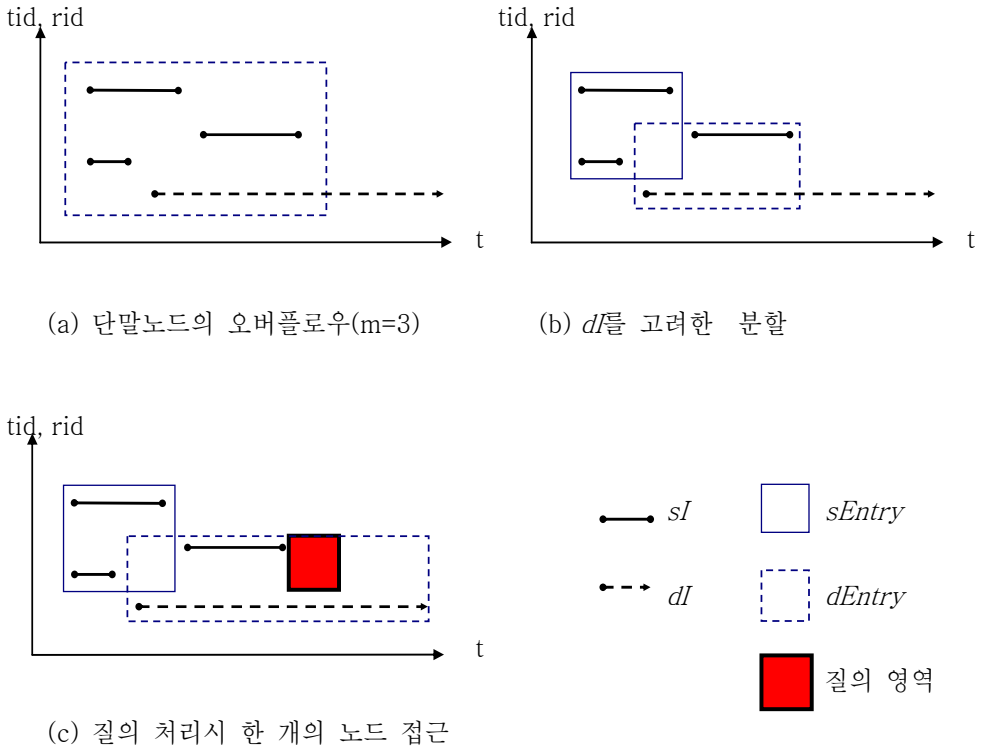
(a) 단말 노드의 오버플로우(m=3)　(b) 기존 분할 방법에 의한 분할

(c) 질의 처리시 두 개의 노드 접근

그림 41 기존 방법에 의한 단말 노드 분할의 문제점

단말 노드 분할의 경우에 기존의 방법을 사용하면 질의 처리 시 다수의

노드에 접근하는 문제가 발생한다. 예를 들어, 그림 41-(a)와 같이 적재용량이 3인 단말 노드에 오버플로우가 발생했다고 가정하자. 이 경우에 기존 방법에서는 그림 41-(b)와 같이 분할되고 그림 41-(c)와 같은 질의 처리 시에 두 개의 단말 노드에 접근하게 된다. 이것은 단말 노드에 포함된 *dI*가 질의 처리 시에 확장된다는 특성을 고려하지 않았기 때문이다.



(a) 단말노드의 오버플로우(m=3)　　(b) *dI*를 고려한 분할

(c) 질의 처리시 한 개의 노드 접근

그림 42 *dI*의 특성을 고려한 단말 노드의 분할

그러나 *dI*가 질의 처리 시에 확장된다는 특성을 고려하여 그림 42-(b)와 같이 분할을 수행하게 되면 노드의 접근을 최소화하여 질의를 처리할 수 있다.

이를 위하여 이 논문에서는 *Local Fixed dI(LFdI)*를 아래와 같이 정의하고 이를 이용하여 단말 노드의 분할을 수행한다.

정의 5: *LFdI*는 단말 노드의 최대 시간 값으로 *dI*의 시간 구간을 고정 시킨 구간

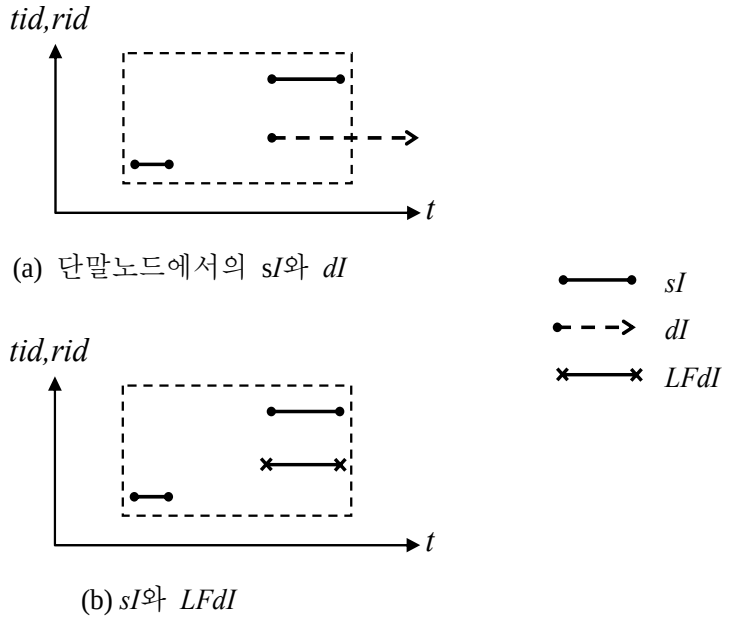


그림 43 *LFdI*의 예

*LFdI*의 예는 그림 43과 같다. 그림 43-(a)와 같이 단말 노드에 *sI*와 *dI*가 있다고 가정하자. 이 노드를 분할하는 경우에 포함되는 모든 *dI*를 그림 43-(b)와 같이 *LFdI*로 만들고 난 뒤에 분할을 수행하게 된다. 이것은 *dI*가 확장되는 성질을 일정으로 보장해 주므로 질의를 효율적으로 처리할 수 있다.

다음은 이 논문에서 제시하는 분할 알고리즘이다. 분할하기 전에 *dI*는 *LFdI*로 *dEntry*는 *LFMBB*로 각각 변환하여 분할을 수행한다. 또한 분할 수행 후, 분할된 노드를 가리키는 부모노드의 엔트리들의 *state* 값은 변경되어야 되며 이것은 루트까지 전달되어야 한다.

Algorithm SplitNode(Node N_{old}, Node N_{new1}, Node N_{new2})

```
1    IF N_old is leat node
2      Change dIs to LFdI in N_old
3      Choose aixs of splitting
4      Distribute M+1 entries into N_new1 and N_new2
5      Set state of entries which indicate N_new1, N_new2 in parent node
6    ELSE
7      Change MBBs of dEntrys to LFMBB in N_old
8      Choose aixs of splitting
9      Distribute M+1 entries into N_new1 and N_new2
10     Set state of entries which indicate N_new1, N_new2 in parent node
```

알고리즘 6 SplitNode()

6. 성능 평가

이 장에서는 논문에서 제안하는 색인의 성능을 평가하기 위하여 기존 R-tree 계열의 삽입 및 분할 방법과 비교한다. 색인의 성능 비교의 기준은 각 색인의 구축 비용과 각 질의의 처리 비용이다. 실험에 사용된 색인은 디스크 기반이기 때문에 각 실험의 비용은 디스크 I/O로 평가된다.

이 논문에서 실험에 사용된 색인의 종류는 표 2와 같다. 실험에 사용된 색인은 구간으로 태그의 궤적을 표현한다. 즉, R-tree의 경우 본래의 R-tree가 아닌 정적 구간과 동적 구간으로 태그의 궤적을 표현하며, 따라서 질의 처리 시에 *dEntry*나 *dI*는 확장된다. 단지 삽입 및 분할 알고리즘의 경우 본래의 R-tree에서 사용된 알고리즘을 사용한다. 결론적으로 실험 평가에 사용되는 색인들은 구간 모델을 기반으로 구현하되 삽입, 분할 알고리즘이 각각 R-tree, R*-tree, IR-tree의 알고리즘을 사용하는 색인이다.

표 2 실험에 사용된 색인과 색인 인자

색인 종류	색인 인자	색인 약어
R-tree	fill factor: 0.7	QUADRATIC
R*-tree	reinsert factor: 0.3 split distribution factor: 0.5	RSTAR
IR-tree	split distribution factor: 0.5	IR

6.1 실험 환경

이 논문에서 제안한 IR-tree는 Microsoft MFC 라이브러리와 Visual C++ 6.0을 사용하여 구현하였고, Windows 2003 서버에서 512MB 메인 메모리, CPU Pentium IV 2.6Ghz를 장착한 PC를 이용하여 실험하였다.

색인의 성능을 검증하기 위해서 다양한 환경에서 색인의 성능을 평가할 수 있는 실험 환경이 필요하다. 이동 객체를 위한 대부분의 색인은 GSTD[62]를 이용하여 다양한 조건에서 실험을 수행한다. 그러나 태그의 경우, 이동 객체와 다른 위치 정보를 요구하므로 새로운 실험 환경이 필요하다.

6.2 실험 데이터

이 논문에서 제안하는 새로운 색인의 실험을 위하여 태그의 위치 데이터를 생성하기 위한 태그 데이터 생성기(Tag Data Generator, TDG)를 개발하여 사용하였다. 기본적인 알고리즘은 GSTD알고리즘과 유사하지만, 차이점은 위치 보고 시점이 호출 영역 안으로 들어갈 때와 밖으로 나갈 때만 데이터를 생성한다. 태그가 2개 이상의 호출 영역이 중첩된 지역 안으로 들어갈 경우, 동일 시간에 하나 이상의 위치를 가지도록 설계하였으며 판독기의 식별자는 임의로 부여하였다. 그림 44는 이 논문에서 제시한 색인의 실험을 위해 개발한 태그 위치 생성기이다. 그림에서 원은 판독기의 호출 영역을 나타내며, 공간상으로 근접한 다수의 호출 영역이 중첩되어 나타날 수도 있다. 호출 영역은 태그가 이동할 수 있는 지역에 설치된다. 그림에서 격자 모양이 나타나

지 않는 곳은 건물의 벽이나 잔디밭과 같은 지역이며 태그가 이동할 수 없는 지역이다.

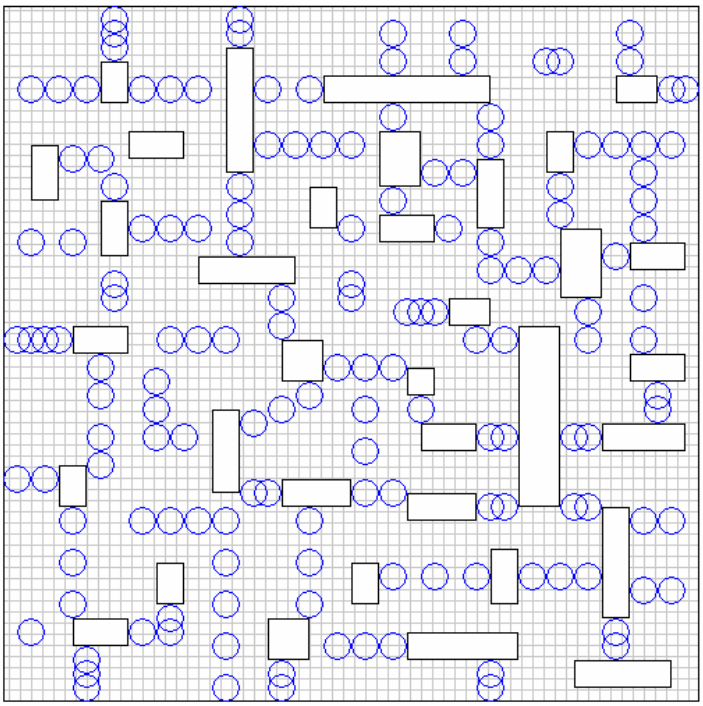
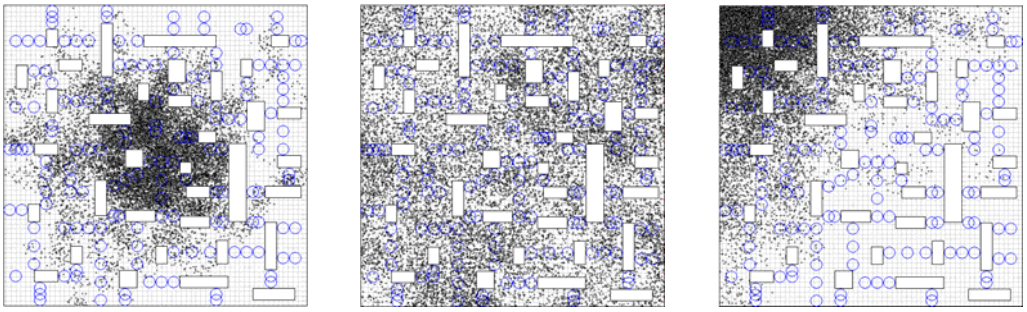


그림 44 태그 데이터 생성기

TDG는 작업 공간의 전체 크기를 임의로 지정할 수 있으며, 태그의 이동 구간, 위치 보고 시간 구간, 보고 횟수 등을 설정할 수 있다. 그리고 태그가 작업 공간에서 초기에 위치할 수 있는 초기 분포와 시간이 지남에 따라 공간과 시간에 대한 분포를 다양하게 지정할 수 있다.



a) 가우시안 분포　　b) 균등 분포　　c) 사향 분포

그림 45 실험에 사용된 3가지 데이터 분포

제공되는 분포로는 그림 45와 같이 가우시안 분포(Gaussian distribution), 균등 분포(Uniform distribution), 사향 분포(Skewed distribution)이며, 사용자는 인자를 조절하여 다양한 태그의 위치 데이터를 생성할 수 있다. 실험에서 사용된 데이터 집합의 종류는 표 3과 같다. 참고로 각 데이터는 4차원 공간에 모든 축을 [0, 1]로 정규화하여 실험을 수행하였다.

표 3 실험에 사용된 데이터 집합의 종류

종류	초기 분포	태그 수	Enter 수	Leave 수
G1	Gaussian	50	5,822	5,724
G2	Gaussian	100	11,262	11,253
G3	Gaussian	200	22,334	22,295
G4	Gaussian	500	57,152	57,058
G5	Gaussian	1,000	90,709	90,512
U1	Uniform	1,000	83,530	83,348
S1	Skewed	1,000	64,581	64,405

6.2 색인 구축 성능 비교

색인 구축 성능을 비교하기 위하여 TDG로 생성한 데이터의 분포에 따라 실험을 수행하였다. 각 색인에서 노드의 크기는 1024바이트이며 노드의 헤드 정보를 64바이트로 가정한다. 변수는 8바이트 Double을 사용하고 노드를 가리키는 포인터를 4바이트 Long을 사용한다고 가정한다. 이 경우에 단말 노드는 총 4개의 변수인 객체식별자, 판독기식별자, 시간 구간을 가지므로 최대 30개의 엔트리를 가질 수 있다. 비단말 노드의 경우 8개의 변수와 하나의 포인터 그리고 상태를 나타내는 1비트 정보를 필요로하므로 최대 18개의 엔트리를 가질 수 있다. 색인 구축 비용은 데이터 삽입과 갱신에서 발생한 노드 접근 횟수의 평균 값이다.

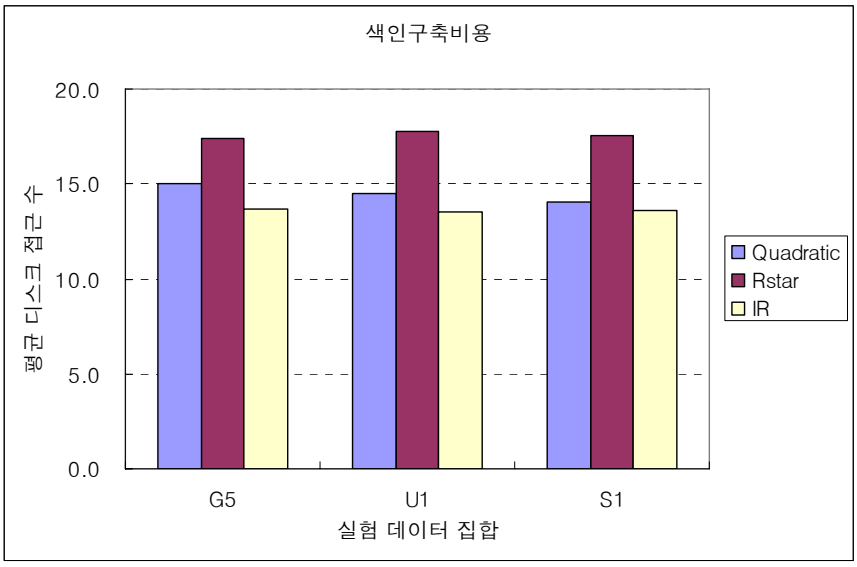


그림 46 데이터 집합에 따른 색인 구축 비용

그림 46과 같이 데이터 집합에 따른 색인 구축 비용은 IR-tree < R-tree < R*-tree의 순으로 나타났다. IR-tree의 경우 질의 처리 능력이 우수하므로 갱신 시 이전 정적 구간을 찾을 때 노드의 접근이 상대적으로 작다. R*-tree는 노드 오버플로우가 발생할 시에 노드를 분할하지 않고 데이터 재삽입을 수행하기 때문에 색인 구축 비용에서 성능이 가장 낮다.

6.3 데이터 분포에 따른 질의 성능 비교

이 논문에서 제안한 IR-tree의 단말 노드의 삽입 및 분할 정책의 성능을 측정하기 위해서 기존 R-tree의 삽입 및 분할 정책과 R*-tree의 삽입 및 분할 정책을 비교하였다. Find와 Look 질의를 처리하는 경우에 각 질의는 1000번을 수행하였으며 접근된 노드 수로써 질의 성능을 평가하였다. 사용된 데이터 집합은 가우시안, 균등, 사향 분포를 사용하여 각 데이터 집합에서의 질의 성능을 비교하였다. 실험에 사용된 질의는 과거 및 현재 위치 질의를 동시에 수행하였다.

6.3.1 가우시안 분포 데이터 집합

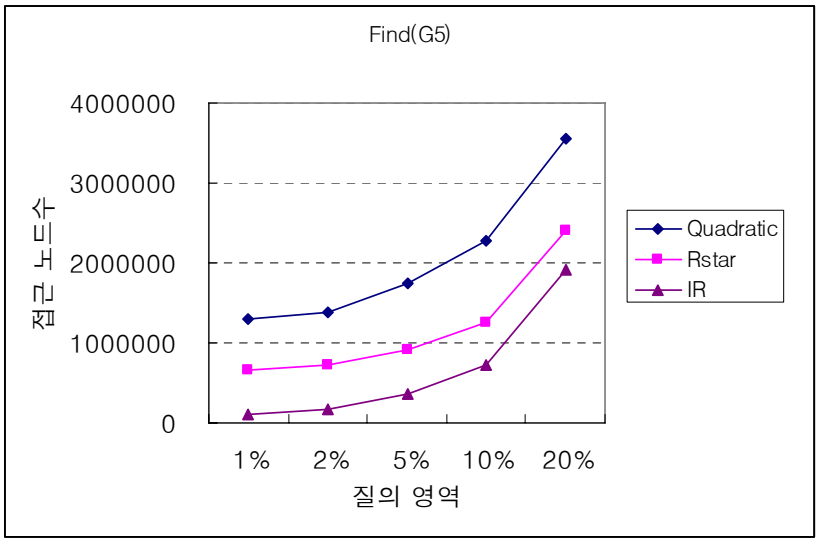


그림 47 가우시안 분포 데이터 집합 G5에서 Find 질의 성능 비교

그림 47은 가우시안 분포 데이터 집합 G5에 대하여 Find 질의를 수행한 결과이다. 그림에서와 같이 IR-tree의 성능이 가장 우수했고 R-tree가 가장 낮은 성능을 보였다. IR-tree가 우수한 성능을 보이는 이유는 질의 처리 시 확장되는 *dI*와 *dEntry*의 특성을 고려한 삽입 및 분할 알고리즘을 적용하였기 때문이다.

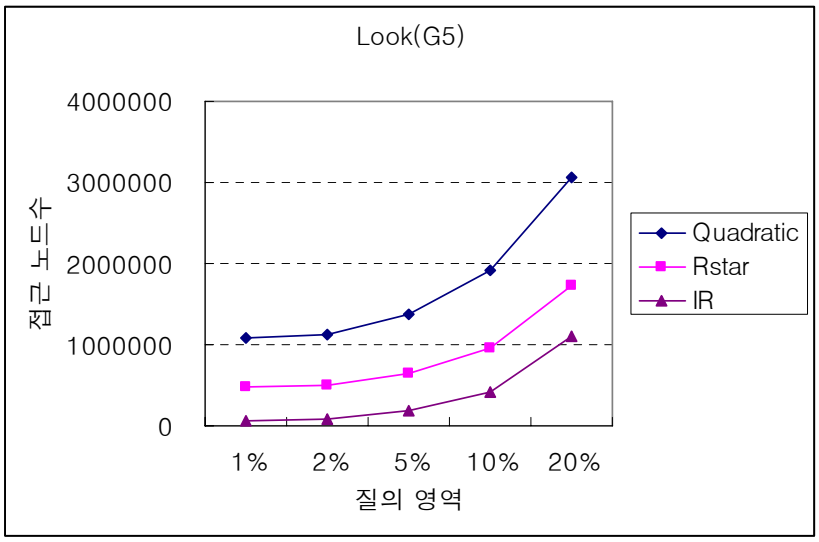


그림 48 가우시안 분포 데이터 집합 G5에서 Look 질의 성능 비교

그림 48은 Look 질의에서 3가지 색인의 성능을 보여준다. Find 질의에서와 마찬가지로 IR-tree는 다른 색인에 비해 우수한 성능을 보인다. Find 질의에서와 마찬가지로 *dI*와 *dEntry*의 특성을 고려한 삽입 및 분할 알고리즘을 적용하였기 때문이다.

주목할만한 것은 동일한 데이터 집합에 대하여 Find 질의가 Look 질의에 비해 더 많은 노드를 접근한다는 것이다. RFID 응용에서는 태그를 장착한 화물 여러 개가 동시에 동일한 창고에 들어가거나 나오는 경우가 빈번하다. 즉, 다수의 태그 객체가 동시에 동일한 판독기에 Enter하거나 Leave하는 경우가 빈번하므로 판독기별로 즉, *rid* 축으로 분할이 빈번하게 일어난다. 따라서 질의의 제약(constraint)가 *rid*와 시간인 Look 질의가 Find 질의보다 우수한 성능을 가진다.

6.3.2 균등 분포 데이터 집합

균등 분포의 데이터 집합에서 실험한 결과는 가우시안 분포와 비슷하다. Find와 Look 질의에서 모두 IR-tree의 성능이 가장 우수했고 R-tree가 가장 낮은 성능을 보였다. 또한 동일한 데이터 집합에 대하여 Look 질의가 Find 질의에 비해 성능이 우수하다.

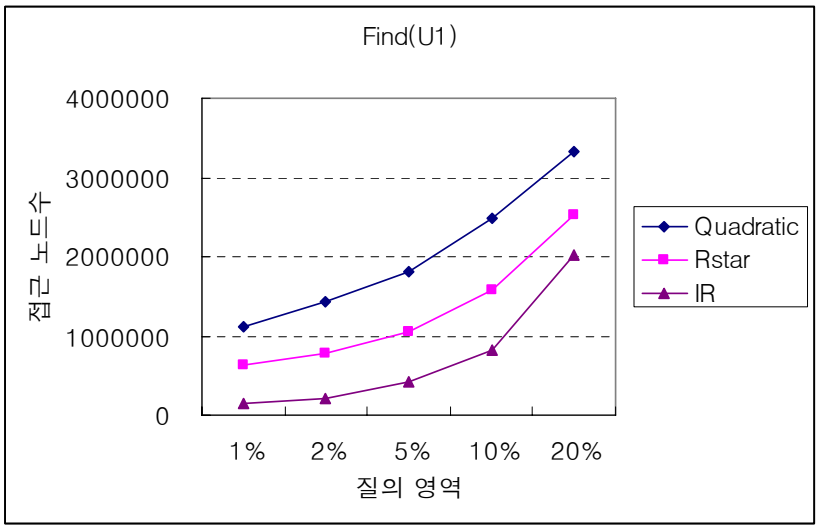


그림 49 균등 분포 데이터 집합 U1에서 Find 질의 성능 비교

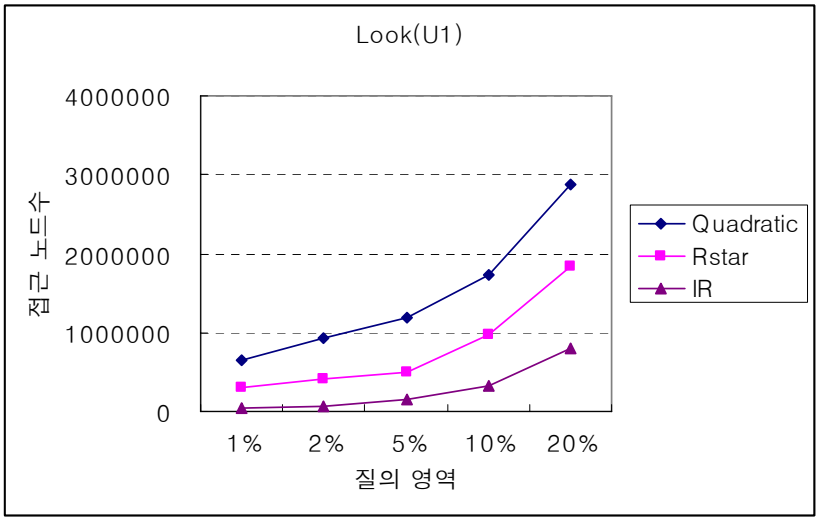


그림 50 균등 분포 데이터 집합 U1에서 Look 질의 성능 비교

6.3.3 사향 분포 데이터 집합

사향 분포의 데이터 집합에서 실험한 결과는 가우시안 및 균등 분포와 비슷하다. Find와 Look 질의에서 모두 IR-tree의 성능이 가장 우수했고 R-tree가 가장 낮은 성능을 보였다. 또한 동일한 데이터 집합에 대하여 Look 질의가 Find 질의에 비해 성능이 우수하다.

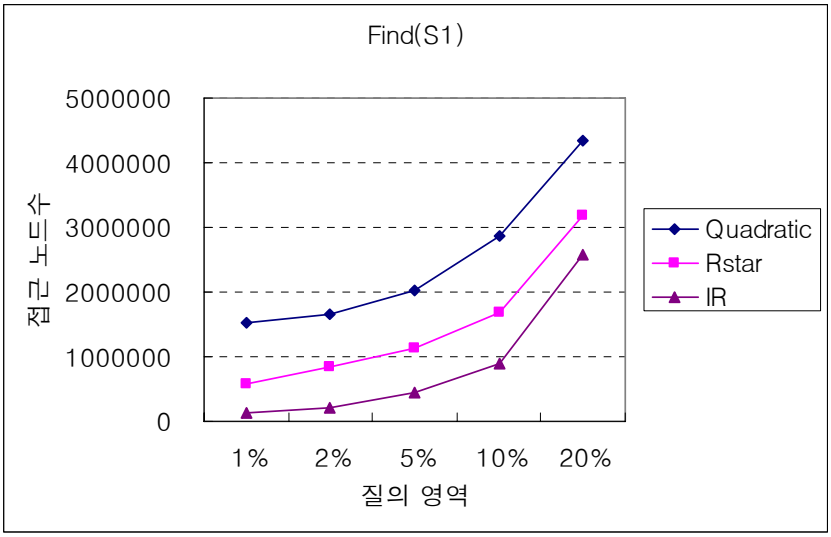


그림 51 사향 분포 데이터 집합 S1에서 Find 질의 성능 비교

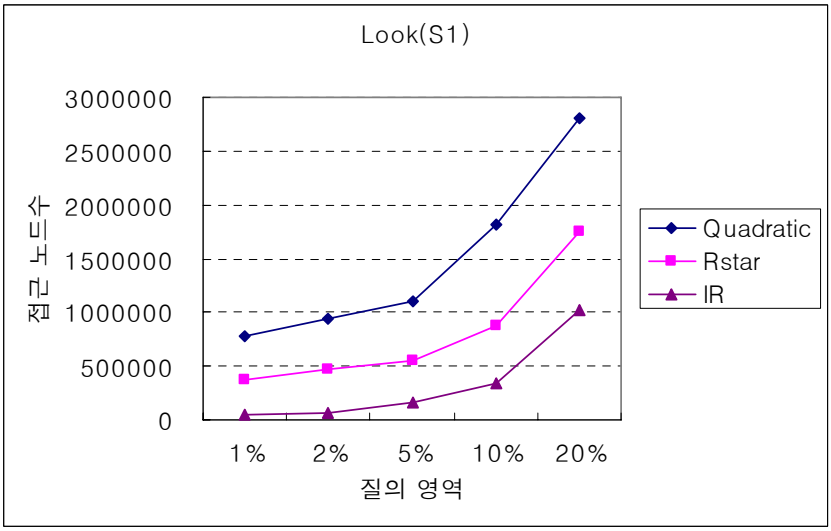


그림 52 사향 분포 데이터 집합 S1에서 Look 질의 성능 비교

6.4 데이터 집합 및 질의 영역의 변화에 따른 성능 비교

이 논문에서는 각 데이터 집합에서 질의 영역에 따른 Find와 Look 질의의 성능을 비교하였다. 각 질의는 1, 2, 5, 10, 20%의 질의 영역을 가지며 각각 1000번을 수행하였으며 접근된 노드 수로써 질의 성능을 평가하였다. 실험에 사용된 질의는 과거 및 현재 위치 질의를 동시에 수행하였다.

그림 53은 가우시안 분포 데이터 집합 G1에 대하여 Find 질의를 수행한 결과이다. 그림에서와 같이 태그의 수가 적은 G1 데이터 집합에서는 IR-tree와 R*-tree의 성능이 거의 비슷함을 알 수 있다. 이것은 태그 수가 적으므로 데이터의 삽입 및 분할이 상대적으로 적게 일어나기 때문에 두 색인의 성능 차가 거의 없는 것이다.

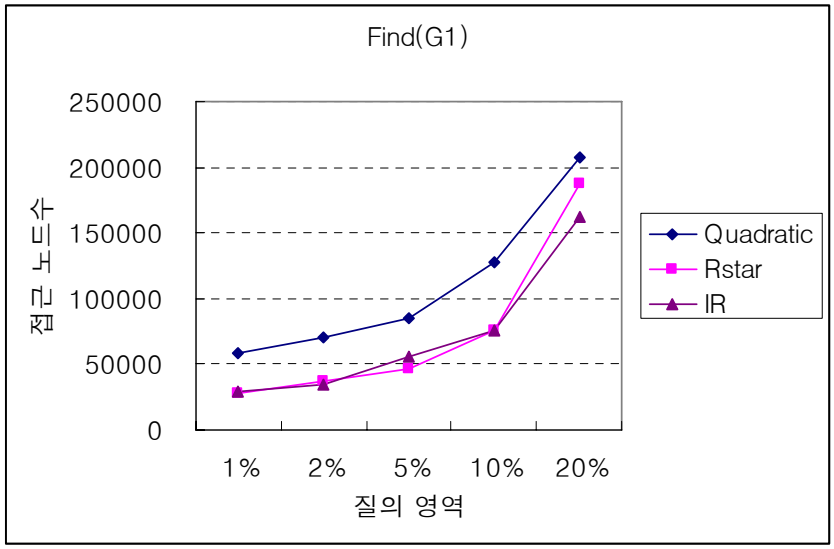


그림 53 데이터 집합 G1에서 질의 영역에 따른 Find 질의 성능 비교

그림 54는 데이터 집합 G1에서 질의 영역의 변화에 따라 Look 질의의 성능을 나타낸다. Find 질의에서와 마찬가지로 IR-tree와 R*-tree의 성능이 거의 비슷함을 알 수 있다. 또한 Find 질의가 Look 질의에 비해 더 많은 노드를 접근하는데, 그 이유는 Find 질의가 공간 차원 *x*, *y* 축을 모두 탐색하는 반면에 Look 질의는 식별자 차원인 *tid* 축만을 탐색하기 때문이다.

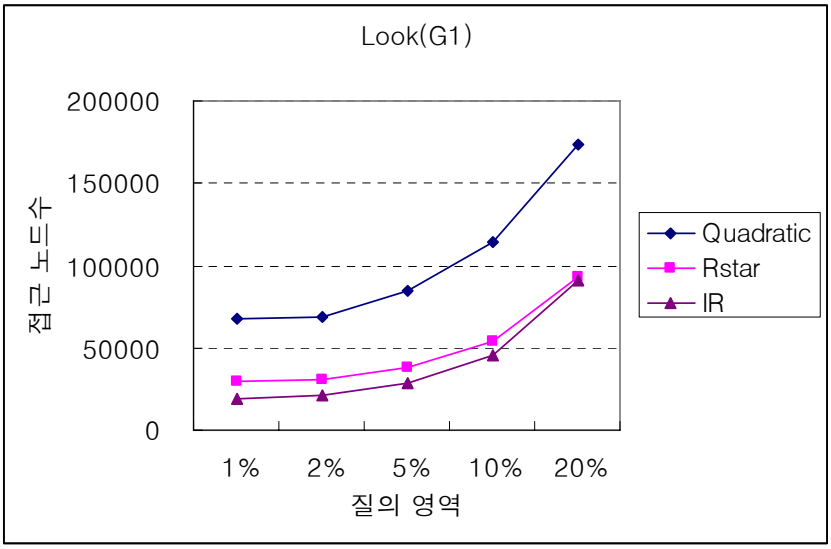


그림 54 데이터 집합 G1에서 질의 영역에 따른 Look 질의 성능 비교

그림 55와 그림 56은 데이터 집합 G2에서 질의 영역의 변화에 따라 Find와 Look 질의의 성능을 나타낸다. 그림에서와 같이 IR-tree, R*-tree, R-tree의 순으로 질의 성능이 우수함을 보인다. 주목할 만한 것은 질의 영역이 커짐에 따라 IR-tree와 R*-tree의 차가 좁혀진다는 것이다. 이러한 이유는 질의 영역이 커질수록 색인의 거의 모든 노드를 접근하기 때문에 그 차가 적어지는 것이다.

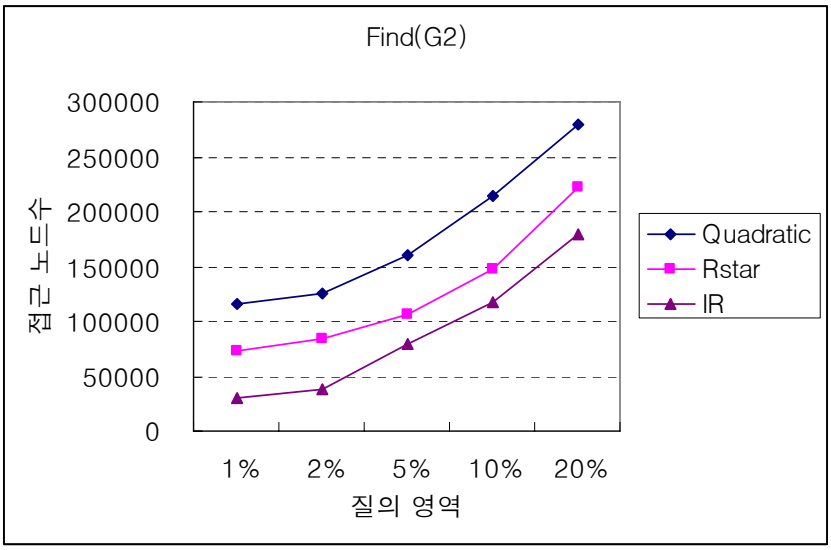


그림 55 데이터 집합 G2에서 질의 영역에 따른 Find 질의 성능 비교

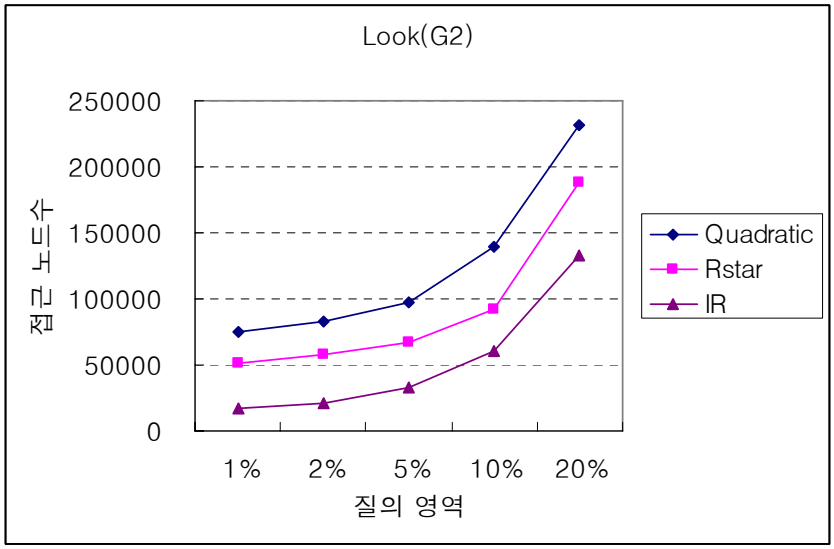


그림 56 데이터 집합 G2에서 질의 영역에 따른 Look 질의 성능 비교

그림 57과 그림 58은 데이터 집합 G3에서 질의 영역의 변화에 따라 Find와 Look 질의의 성능을 나타낸다. 그림에서와 IR-tree, R*-tree, R-tree의 순으로 질의 성능이 우수함을 보인다. 또한 질의 영역이 커질수록 색인의 거의 모든 노드를 접근하기 때문에 IR-tree와 R*-tree의 차가 좁혀진다. 또한 동일한 데이터 집합에 대하여 Look 질의가 Find 질의에 비해 성능이 우수하다.

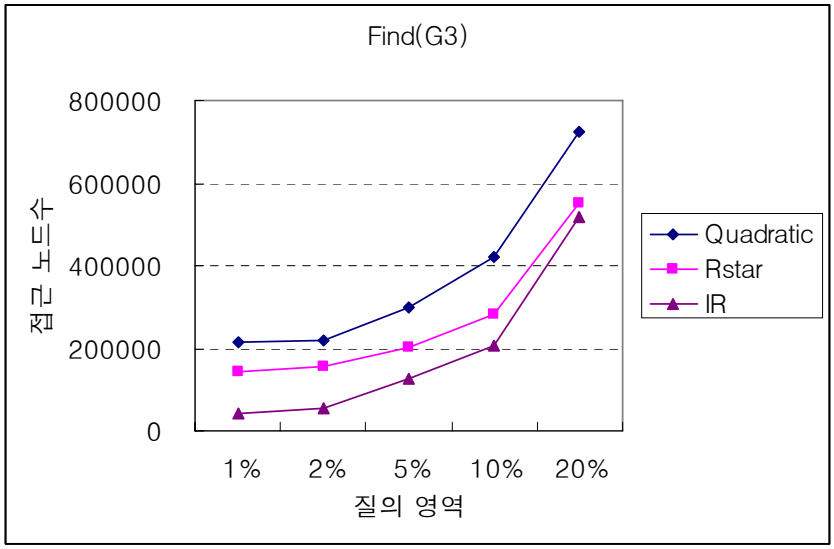


그림 57 데이터 집합 G3에서 질의 영역에 따른 Find 질의 성능 비교

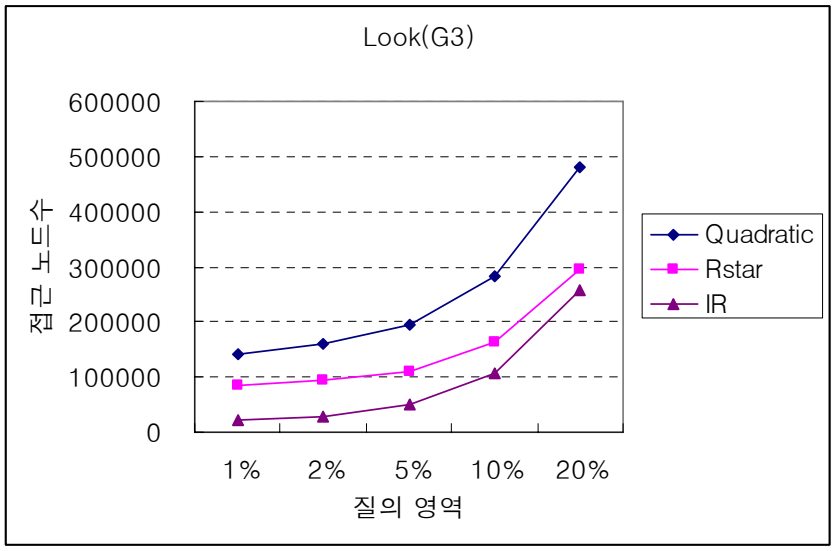


그림 58 데이터 집합 G3에서 질의 영역에 따른 Look 질의 성능 비교

그림 59와 그림 60은 데이터 집합 G4에서 질의 영역의 변화에 따라 Find와 Look 질의의 성능을 나타낸다. 이 데이터 집합 또한 다른 데이터 집합과 동일하게 IR-tree, R*-tree, R-tree의 순으로 질의 성능이 우수함을 보인다. 결론적으로 어느 질의 영역에서나 IR-tree의 성능이 다른 색인에 비해 우수하였으며 특히 질의 영역의 크기가 커질수록 IR-tree와 R*-tree의 격차가 적어짐을 알 수 있었다.

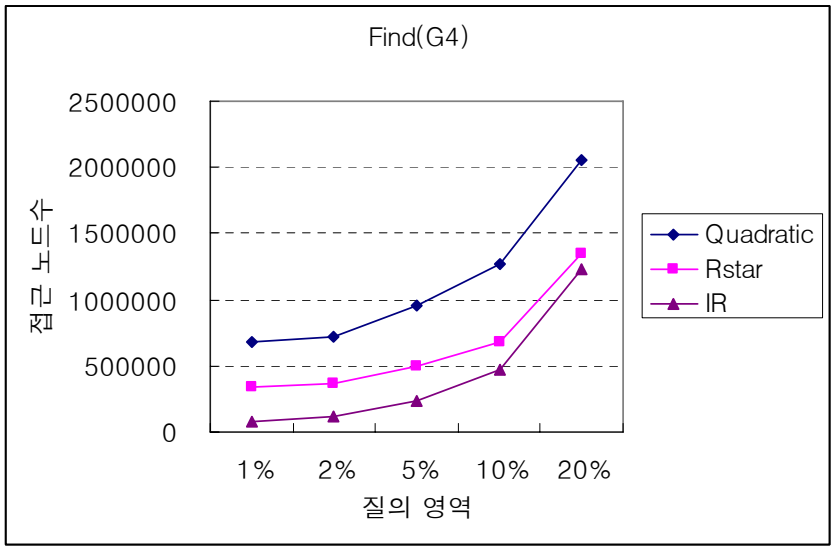


그림 59 데이터 집합 G4에서 질의 영역에 따른 Find 질의 성능 비교

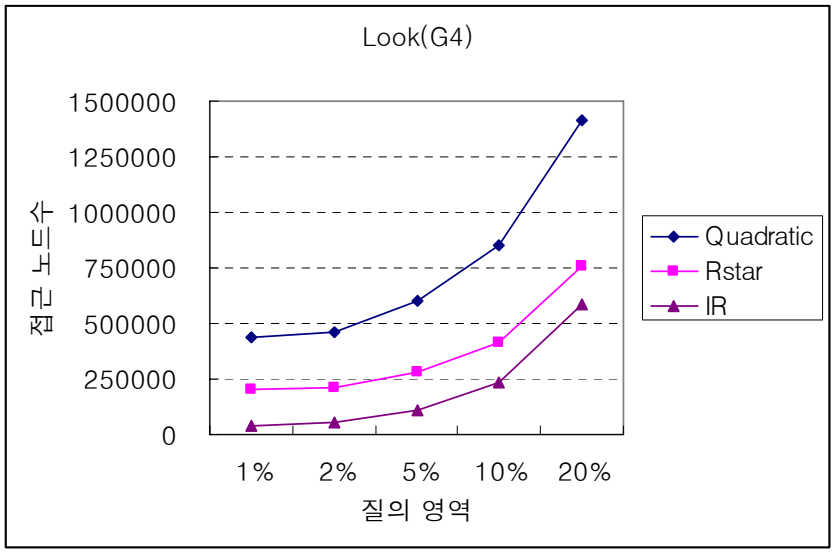


그림 60 데이터 집합 G4에서 질의 영역에 따른 Look 질의 성능 비교

6.5 태그 수의 증가 따른 질의 성능 비교

이 논문에서는 태그 수의 증가에 따른 Find와 Look 질의의 성능을 비교하였다. 태그의 객체 수를 각각 50, 100, 200, 500, 1000으로 늘려서 실험을 수행하였으며 각 태그는 Enter와 Leave를 1000번씩 보고한다. 20%의 질의 영역을 가지는 질의를 1000번을 수행하였으며 접근된 노드 수로써 질의 성능을 평가하였다. 실험에 사용된 질의는 과거 및 현재 위치 질의를 동시에 수행하였다.

그림 61은 가우시안 분포 데이터 집합에서 태그의 수의 증가에 따른 Find 질의를 수행한 결과이다. 그림에서와 같이 전 구간에서 IR-tree, R*-tree, R-tree의 순으로 성능을 나타내었다. 또한 태그의 수가 증가할수록 성능의 차가 더욱 큼을 알 수 있었다.

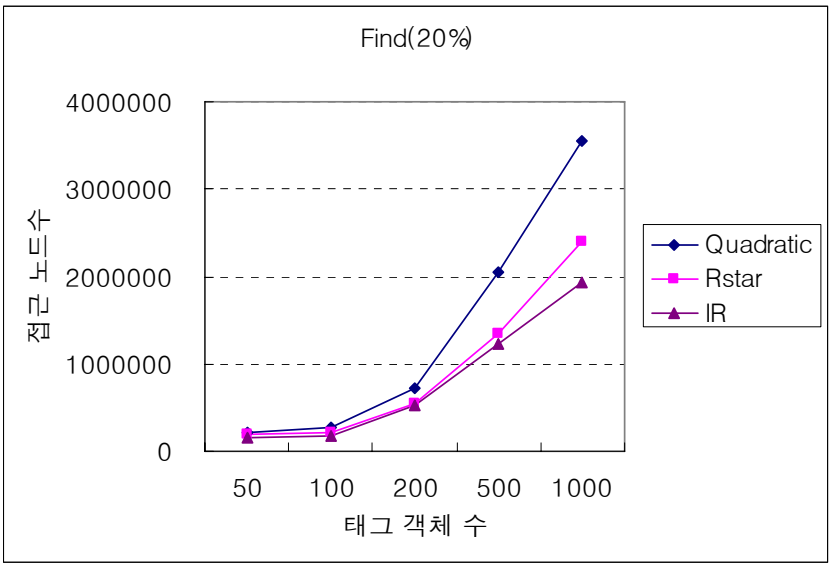


그림 61 태그 수의 증가에 따른 Find 질의 성능 비교

그림 62는 가우시안 분포 데이터 집합에서 태그의 수의 증가에 따른 Look 질의를 수행한 결과이다. Find 질의에서와 마찬가지로 IR-tree가 전 구간에서 다른 색인보다 그 성능이 우수함을 알 수 있었다. 또한 동일한 데이터 집합에 대하여 Look 질의가 Find 질의에 비해 성능이 우수하다. 그 이유는 다수의 태그 객체가 동시에 동일한 판독기에 Enter하거나 Leave하는 경우가 빈번하므로 판독기별로 분할이 빈번하게 일어나 제약(constraint)이 *rid*인 Look 질의가 우수한 성능을 가진다.

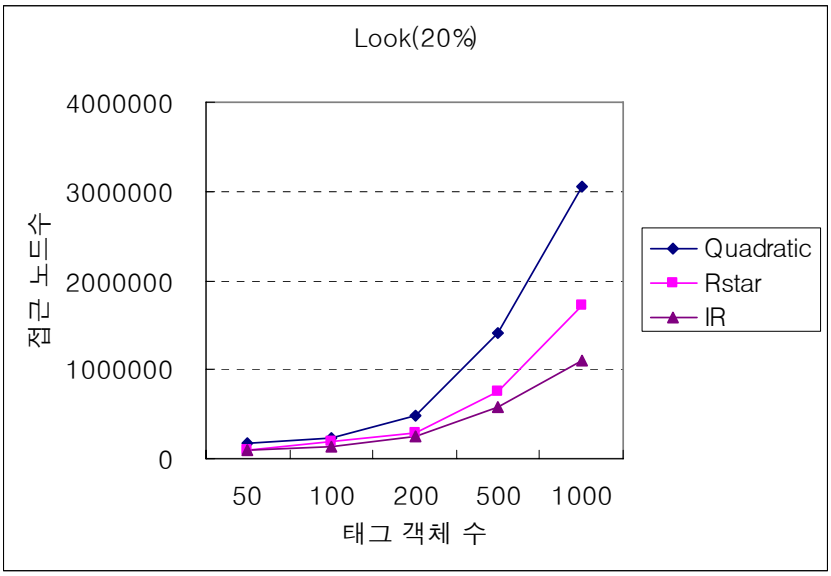


그림 62 태그수의 증가에 따른 Look 질의 성능 비교

7. 결론 및 향후 연구

태그와 이동 객체는 시간에 따라 위치가 변하는 공통된 특성을 가지므로 태그의 궤적을 추적하기 위한 색인은 이동 객체의 색인을 사용하여 구축할 수 있다. 그러나 판독기에 머물고 있는 태그는 궤적을 표현할 수가 없으므로 질의 시 이러한 태그를 검색할 수 없다. 또한 태그의 식별자에 관한 질의를 처리하는 경우에 식별자에 대한 색인이 구축되지 않았으므로 질의 처리 시에 많은 시간이 걸리는 문제가 발생한다.

이 논문에서는 위와 같은 문제를 해결하기 위하여 RFID 태그의 궤적을 위한 구간 데이터 모델을 정의하였다. 이 모델에서는 태그의 궤적을 판독기에 들어올 때 생성되는 시간에 종속적인 선분인 동적 구간(dynamic interval)과 판독기에 나올 때 생성되는 시간에 고정적인 정적 구간(static interval)으로 표현하였다. 따라서 판독기에 머무는 태그의 궤적은 시간에 종속적인 동적 구간으로 표현되므로 이러한 태그를 찾을 수 있게 한다.

또한 태그의 추적을 위한 질의를 분류하였다. 분류된 질의는 태그의 위치를 찾는 Find 질의, 판독기에 위치한 태그를 찾는 Look 질의, 특정 태그와 같이 위치한 태그를 찾는 With 질의, 태그의 이동 경로를 찾는 Trace 질의이다. 이 논문에서는 이러한 질의를 효율적으로 처리하기 위해 질의의 매개 변수로 태그의 식별자가 들어간다는 특성을 고려하여 객체 식별자를 색인의 도메인으로 추가하였다.

제시한 태그의 구간 데이터 모델에 적합한 R-tree 기반 색인 구조인 IR-tree(Interval R-tree)를 제시하였다. 단말 노드에서는 구간을 저장하는데 구간의

특성을 고려하여 적은 정보만으로도 구간을 표현할 수 있게 하였다. 또한 단말 노드와 비단말 노드는 포함하는 엔트리의 속성에 따라 *dI/sI, dEntry/sEntry*로 분류하였으며 특히 *dI, dEntry*의 경우 질의 처리 시에 확장되어 질의를 처리하게 설계 하였다.

또한 효율적인 질의 처리를 위해 시간에 종속적인 *dI*와 *dEntry*의 특성을 고려한 새로운 삽입 및 분할 알고리즘을 제안한다. 삽입 알고리즘에서는 *dEntry*가 질의 처리 시에 현재 시간 값으로 시간 구간이 확장된다는 특징을 고려하여 현재 *dEntry*의 *MBB*를 시간 구간으로 확장 시킨 *LFMBB*를 제안하고 새로운 영역 확장 계산 방법을 제시하였다. 이러한 삽입은 실험 결과 노드의 크기를 줄여 질의 처리를 빠르게 하였다.

분할 알고리즘은 *dI*가 질의 처리 시에 현재 시간 값으로 시간 구간이 확장된다는 특징을 고려하여 *dI*를 확장 시킨 *LFdI*를 제안하여 단말 노드를 분할 하였다. 분할 방법은 이러한 *LFdI*를 이용하여 R*-tree처럼 마진을 최소로 분할 축을 설정하고 영역을 최소로 분할한다. 역시 이러한 분할 방법은 단말 노드의 크기를 줄여 질의 처리를 빠르게 하였다.

실험을 위하여 태그의 위치 데이터 생성기인 TDG를 GSTD 알고리즘에 기반하여 설계 및 구현하였으며, 다양하게 생성된 데이터를 사용하여 성능 평가를 수행하였다. 실험 평가는 질의 영역과 태그 수 및 보고 횟수를 변경하면서 수행하였다. 이 논문에서 제안하는 IR-tree가 기존 색인들보다 Find 질의와 Look 질의에서 우수한 질의 성능을 보였다.

향후 연구로서는 태그의 이동 특성을 고려한 삽입, 분할 알고리즘의 개발

이 필요하다. 태그는 복합 객체로서 계층적으로 구성할 수 있다. 예를 들어, 컨테이너에 다수의 제품을 실어 나르는 경우 제품에 태그가 부착되어 있다면 태그는 그룹 단위로 동일한 궤적을 가지면서 이동한다. 따라서 이러한 특성을 고려한 삽입, 분할 정책이 필요하다. 다른 향후 연구로서는 다양한 고급 질의 예를 들어, 최소근접질의나 위상 질의 등의 수행에 관한 연구가 필요하다.

참고 문헌

[1] K. Romer, T. Schoch, F. Mattern and T. Dubendorfer, “Smart Identification Frameworks for Ubiquitous Computing Applications” *Proc. of Pervasive Computing and Communications*, pp. 256-262, 2003

[2] S. E. Sarma, S. A. Weis, and D. W. Engels. “RFID Systems and Security and Privacy Implications”. *Springer-Verlag*, pp. 454-469, 2002

[3] K. Romer, T. Schoch. “Infrastructure Concepts for Tag-Based Ubiquitous Computing Applications”. *Proc. of Ubicomp*, pp. 253-262, 2002

[4] EPC Tag Data Standard Work Group, “EPC Tag Data Standards Version 1.23”, EPC Global, 2005

[5] J. L. Bently. “Algorithms for Klee’s Rectangle Problems”. *Computer Science Department, Carnegie-Mellon University*, 1977.

[6] H. Edelsbrunner. “A New Approach to Rectangle Intersections”. *International Journal of Computer Mathematics*, pp.209-229, 1983.

[7] E. M. McCreight. “Priority Search Trees”. *SIAM Journal of Computing*, pp.257-276, 1985.

[8] M. Abdelguerfi, J. Givaudan, K. Shaw, and R. Ladner, “The 2-3TR-tree, A trajectory-oriented index structure for fully evolving valid-time spatio-temporal datasets,” *Proc. of the ACM Int’l symposium on advances in geographic information systems*, pp. 29-34, 2002.

[9] P. K. Agarwal, L. Arge, and J. Erickson, “Indexing moving points,” *Proc. of the ACM Symposium on Principles of Database Systems*, pp. 175-186, 2000.

[10] L. Arge, K. Hinrichs, J. Vahrenhold, and J. S. Vitter, “Efficient bulk operations on dynamic R-trees,” *Proc. of the ACM Symposium on Principles of Database Systems*, pp. 328-348, 1999.

[11] J. Basch, L. J. Guibas, and J. Hershberger, “Data structures for mobile data,” *Proc. of the ACM-SIAM Symposium on Discrete Algorithms*, pp. 747-756, 1997.

[12] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer, “An asymptotically optimal multiversion B-Tree,” *VLDB Journal*, vol. 5, no. 4, pp. 264-275, 1996.

[13] N. Beckmann and H. P. Kriegel, “The R*-tree: An efficient and robust access method for points and rectangles,” *Proc. of the ACM SIGMOD Int’l Conf. on Management of Data*, pp. 332-331, 1990.

[14] C. Kolovson and M. Stonebraker. “Segment Indexes: Dynamic Indexing Techniques for Multi-Dimensional Interval Data”, *Proc. ACM SIGMOD*, pp.138-147, 1991.

[15] S. Berchtold, D. A. Keim, and H. P. Kriegel, “The X-tree : An index structure for high-dimensional data,” *Proc. of Int’l Conf. on Very Large Data Bases*, pp. 28-39, 1996.

[16] R. Bliujute, C. S. Jensen, S. Saltenis, and G. Slivinskas, “R-tree based indexing of now-relative bitemporal data,” *Proc. of Int’l Conf. on Very Large Data Bases*, pp.

345-356, 1998.

[17] T. Brinkhoff “A framework for generating network-based moving objects,” *Proc. of the Int’l Conf. on Scientific and Statistical Database Management*, pp. 253-255, 2000.

[18] D.-S. Cho and B.-H. Hong, “Optimal page ordering for region queries in static spatial databases,” *Proc. of Int’l Conf. on Database and Expert Systems Applications*, pp. 366-375, 2003.

[19] J. Clifford, C. E. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass, “On the semantics of “now” in databases,” *ACM Transactions on Database Systems*, vol. 22, no. 2, pp. 171-214, 1997.

[20] M. Erwig, R. H. Guting, M. Schneider, and M. Vazirgiannis, “Abstract and discrete modeling of spatio-temporal data types,” *Int’l symposium on Advances in Geographic Information Systems*, pp. 131-136, 1998.

[21] M. Erwig, R. H. Guting, M. Schneider, and M. Vazirgiannis, “Spatio-temporal data types: An approach to modeling and querying moving objects in databases,” *Int’l Journal on Advances of Computer Science for Geographic Information Systems*, vol. 3, no. 3, pp. 269-296, 1999.

[22] L. Forlizzi, R. H. Guting, E. Nardelli, and M. Schneider, “A data model and data structures for moving objects databases,” *Proc. of the ACM SIGMOD Int’l Conf. on Management of Data*, pp. 319-330, 2000.

[23] J. Green, D. Betti, and J. Davison, Mobile Location Services: Market Strategies,

Ovum Ltd, 2000.

[24] D. Greene, “An implementation and performance analysis of spatial data access methods,” *Proc. of Int’l Conf. on Data Engineering*, pp. 606-615, 1989.

[25] R. H. Guting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis, “A foundation for representing and querying moving objects,” *ACM Transactions on Database Systems*, vol. 25, no. 1, pp. 1-42, 2000.

[26] A.Guttman, “R-trees: A dynamic index structure for spatial searching,” *Proc. of the ACM SIGMOD Int’l Conf. on Management of Data*, pp. 47-54, 1984.

[27] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos, “Efficient indexing of spatiotemporal objects,” *Int’l Conf. on Extending Database Technology*, pp. 251-268, 2002.

[28] B. Jun, B. Hong, and B. Yu, “Dynamic splitting policies of the adaptive 3DR-tree for indexing continuously moving objects,” *Int’l Conf. on Database and Expert Systems Applications*, September, 2003.

[29] Kamel and C. Faloutsos, “On packing R-trees,” *Int’l Conf. on Information and Knowledge Management*, pp. 490-499, 1993.

[30] G. Kollios, D. Gunopulos, and V. J. Tsotras. “On indexing mobile objects,” *Proc. of the ACM Symposium on Principles of Database Systems*, pp. 261-272, 1999.

[31] G. Kollios, V. J. Tsotras, D. Gunopulos, A. Delis, and M. Hadjieleftheriou, “Indexing animated objects using spatiotemporal access methods,” *IEEE*

Transactions on Knowledge and Data Engineering, vol. 13, no. 5, pp. 758-777, 2001.

[32] A. Kumar, V. J. Tsotras, and C. Faloutsos, "Designing access methods for bitemporal databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 1, pp. 1-20, 1998.

[33] D. Kwon, S. Lee, and S. Lee, "Indexing the current positions of moving objects using the lazy update R-tree," *Int'l Conf. on Mobile Data Management*, pp. 113-120, 2002.

[34] Lazaridis, K. Porkaew, and S. Mehrotra, "Dynamic queries over mobile objects," *Int'l Conf. on Extending Database Technology*, pp. 269-286, 2002.

[35] S. T. Leutenegger and M. A. Lopez, "The effect of buffering on the performance of R-Trees," *Proc. of the Int'l Conf. on Data Engineering*, pp. 164-171, 1998.

[36] M. A. Nascimento and J. R. O. Silva, "Towards historical R-trees," *ACM symposium on Applied Computing*, pp. 235-240, 1998.

[37] M. A. Nascimento, J. R. O Silva, and Y. Theodoridis. "Access structure for moving points," TimeCenter Technical Report TR-33, 1998.

[38] M. A. Nascimento, J. R. O. Silva, and Y. Theodoridis, "Evaluation of access structures for discretely moving points," *Proc. of Int'l Workshop on Spatio-Temporal Database Management*, pp. 171-188, 1999.

[39] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "Then Grid Files: An adaptive, symmetric multikey file structure," *ACM Transactions on Database Systems*,

vol.9, no. 1, pp. 38-71, 1984.

[40] Orenstein, “Redundancy in spatial databases,” *Proc. of the ACM SIGMOD Int’l Conf. on Management of Data*, pp. 294-305, 1989.

[41] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer, “Towards an analysis of range query performance in spatial data structures,” *Proc. of the ACM Symposium on Principles of Database Systems*, pp. 214-221, 1993.

[42] D. Papadias and Y. Theodoridis, “Spatial relations, minimum bounding rectangles, and spatial data structures,” *Int’l Journal of Geographic Information Systems*, vol. 11, no. 2, pp. 111-138, 1997.

[43] A. Papadopoulos and Y. Manolopoulos, “Performance of nearest neighbor queries in R-Trees,” *Proc. of Int’l Conf. on Database Theory*, pp. 394-408, 1997.

[44] D. Pfoser and C. S. Jensen, “Capturing the uncertainty of moving-object representations,” *Proc. of Int’l Symposium on Spatial Databases*, pp. 111-132, 1999.

[45] D. Pfoser, Y. Theodoridis, and C. S. Jensen, “Indexing trajectories in query processing for moving objects,” Chorochronos Technical Report, CH-99-3, October, 1999.

[46] D. Pfoser, C. S. Jensen, and Y. Theodoridis, “Novel approaches in query processing for moving objects,” *Proc. of Int’l Conf. on Very Large Data Bases*, pp. 395-406, 2000.

[47] D. Pfoser, “Indexing the trajectories of moving objects,” *IEEE Data Engineering*

Bulletin, vol. 25, no. 2, pp. 3-9, 2002.

[48] Porkaew, I. Lazaridis, and S. Mehrotra, "Querying mobile objects in spatio-temporal databases," *Proc. of Int'l Symposium on Spatial and Temporal Databases*, pp. 59-78, 2001.

[49] T. Robinson, "The K-D-B-tree: A search structure for large multidimensional dynamic indexes," *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pp. 10-18, 1981.

[50] Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pp. 71-79, 1995.

[51] M. Saglio and J. Moreira, "Oporto: A realistic scenario generator for moving objects," *Proc. DEXA Workshop on Spatio-Temporal Data Models and Languages*, pp. 426-43, 1999.

[52] S. Saltenis and C. S. Jensen, "R-Tree based indexing of general spatio-temporal data," TimeCenter Tech. Report TR-45, 1999.

[53] S. Saltenis, C. S. Jensen, S.T. Leutenegger, and M. A. Lopez, "Indexing the positions of continuously moving objects," *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pp. 331-342, 2000.

[54] H. Samet, The Design and Analysis of Spatial Data Structures, *Addision-Wesley*, Reading, MA, 1990.

[55] T. K. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-Tree: A dynamic index for multi-dimensional objects," *Proc. of Int'l Conf. on Very Large Data Bases*, pp.

507-518, 1987.

[56] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, “Modeling and querying moving objects,” *Int’l Conf. on Data Engineering*, pp. 422-432, 1997.

[57] Z. Song and N. Roussopoulos, “Hashing moving object,” *Int’l. Conf. on Mobile Data Management*, pp. 161-172, 2001.

[58] Y. Tao and D. Papadias, “MV3R-Tree: A spatio-temporal access method for timestamp and interval queries,” *Proc. of Int’l Conf. on Very Large Data Bases*, pp. 431-440, 2001.

[59] J. Tayeb, O. Ulusoy, and O. Wolfson. “A quadtree based dynamic attribute indexing method,” *The Computer Journal*, vol. 41, no. 3, pp. 185-200, 1998.

[60] Y. Theodoridis, M. Vazirgiannis, and T. K. Sellis, “Spatio-temporal indexing for large multimedia applications,” *IEEE Int’l Conf. on Multimedia Computing and Systems*, pp. 441-448, 1996.

[61] Y. Theodoridis, T. K. Sellis, A. Papadopoulos, and Y. Manolopoulos, “Specifications for efficient indexing in spatiotemporal databases,” *Int’l Conf. on Scientific and Statistical Database Management*, pp. 123-132, 1998.

[62] Y. Theodoridis, J. R. O Silva, and M.A Nascimento, “On the generation of spatiotemporal datasets,” *Proc. of Int’l Symposium on Spatial Databases*, pp. 147-164, 1999.

[63] O. Wolfson, B.Xu, S. Chanmberlain, and L. Jiang. “Moving objects database: issues and solutions,” *Proc. of Int’l Conf. on Scientific and Statistical Database*

Management, pp. 111-122, 1998.

[64] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha, “Updating and querying databases that track mobile units,” *Distributed and Parallel Databases*, vol. 7, no. 3, pp. 257-387, 1999.

[65] O. Wolfson, A. P. Sistla, Bo Xu, J. Zhou, and S. Chamberlain, “DOMINO: Databases for moving objects tracking,” *Proc. of the ACM SIGMOD Int’l Conf. on Management of Data*, pp. 547-549, 1999.

[66] X. Xu, J. Han, and W.Lu, “RT-tree: An improved R-tree index structure for spatiotemporal databases,” *Proc. of the Int’l Symposium on Spatio Data Handling*, pp. 1040-1049, 1990.

[67] T. S. Yeh and Y. H. Viemont, “Temporal aspects of geographical databases,” *European Conf. and Exhibition on Geographical Information Systems*, pp. 320-328, 1992.

[68] J. Zoble, A. Moffat, and K. Ramamohanarao, “Guildlines for presentation and comparison of indexing techniques,” *ACM SIGMOD Record*, vol. 25, no. 3, pp. 10-15, 1996.

[69] ChaeHoon Ban, BongHee Hong, DongHyun Kim, "Time Parameterized Interval R-Tree for Tracing Tags in RFID Systems", *In Proc. DEXA*, pp. 503-513, 2005

[70] 반재훈, 홍봉희, "RFID시스템에서 태그의 위치 추적을 위한 시간 매개변수 간격 모델링 기법", 한국정보처리학회 춘계학술발표논문집, 제12권 제1호, pp. 129-132, 2005

[71] 반재훈, 전희철, 안성우, 김진덕, 홍봉희, "이동 객체의 과거, 현재 및 미래 위치 질의 처리를 위한 통합 색인의 설계 및 구현", 한국공간정보시스템학회 논문지, 제 7권 제 1호, pp. 77-89, 2005

[72] 반재훈, 김진곤, 전봉기, 홍봉희, "이동 객체를 위한 R-트리 기반 색인에서의 궤적 클러스터링 정책", 한국정보처리학회 논문지 제 12-D권 제 4호, pp. 507-520, 2005

Efficient Index Structures of Intervals for RFID Tag Objects

Chae Hoon Ban

Dept. of Computer Engineering, Pusan National University

Abstract

RFID is a labeling method in which electronic tags are attached to physical objects and identified when they enter and leave the vicinity of antenna connected to a device known as a reader. There are many applications for RFID systems, such as automated manufacturing, inventory tracking and supply chain management that need to trace trajectories as well as monitor present locations of the tags.

A spatiotemporal index can be constructed for tracing trajectories of tags because they move continuously like moving objects. As the moving objects report periodically their locations while moving, an index can be constructed with trajectories represented as the lines by connecting spatiotemporal locations. A similar index for tags can be constructed with spatiotemporal locations captured when tags move between readers

The problem with using any of the above index schemes for tags is that tags that enter a reader but do not leave cannot be found in the index. The trajectory of a tag is represented as a line by connecting two spatiotemporal locations captured when the tag enters and then leaves the vicinity of a reader. If a tag enters but does not leave a reader, its trajectory is

represented only as a point captured at entry. Because the information that a tag stays in a reader is missing from the trajectory represented only as a point, it is impossible to find the tag that remains in a reader. Therefore, trajectory of the tag should be treated differently in the index.

In this thesis, we propose an interval data model of tag object's trajectory in order to solve the problem. Trajectories of tags are represented as two kinds of intervals; dynamic intervals which are time-dependent lines and generated when tags enter readers and static intervals which are fixed lines and generated when tags leave readers. Although a tag may only be reported when entering a reader, it is still possible to process queries because its trajectory is represented not as a point but as a time-dependent line.

We also classify queries for tracing tag objects. There are four kinds of queries in RFID systems. Find query finds location of tag objects, Look finds tags which stay specific readers, With finds tags which stay with specific tag and History finds the trajectory of specific tag. In this thesis, we treat identifications of tag objects as fourth dimension for processing queries efficiently, because they are parameters of the queries.

For the interval data model, we propose a new index scheme called the IR-tree(Interval R-tree) and algorithms of insert and split for processing query efficiently. We also evaluate the performance of the proposed index scheme and compare it with the previous indexes.

RFID 태그 객체를 위한 효율적인 구간 색인 구조

반 재 훈

부산대학교 컴퓨터공학과

요 약

RFID(Radio Frequency IDentification)는 무선 주파수를 이용하여 태그(tag)를 장착한 객체를 판독기(reader)가 자동으로 인식하고 확인하는 기술로서 태그를 장착한 객체의 위치 추적이나 객체의 현재의 상태를 감시하는 자동화 생산(automated manufacturing), 재고 관리(inventory tracking), 공급망 관리(supply chain management) 등과 같은 다양한 응용분야에서 사용된다.

태그는 이동 객체(moving object)와 유사하게 시간에 연속적으로 이동하므로 태그의 궤적을 추적하기 위해서 이동 객체를 위한 시공간 색인을 적용할 수 있다. 즉, 이동 객체는 이동하면서 일정한 시간에 위치를 보고하므로 보고된 두 개의 시공간 위치를 연결하는 선분으로 궤적을 표현하고 색인을 구성할 수 있다. 마찬가지로 태그가 판독기에 들어올 때와 나갈 때 보고되는 시공간 위치를 이용하여 태그의 궤적을 선분으로 표현하고 색인을 구성할 수 있다.

그러나 이러한 궤적의 모델링 기법을 사용하는 경우에 판독기의 인식영역에 들어와 머무는 태그의 현재 위치를 찾을 수 없는 문제가 발생한다. 태그의 궤적

은 태그가 판독기에 들어올 때와 나갈 때 보고하는 두 개의 시공간 위치를 연결한 선분으로 표현된다. 만약 태그가 판독기에 머무는 경우에 궤적은 태그가 판독기에 들어갈 때 보고하는 시공간 점으로만 구성된다. 따라서 태그가 판독기에 머문다는 정보를 시공간 점으로는 표현할 수 없으므로 판독기에 머무는 태그의 현재 위치를 찾을 수 없는 문제가 발생한다. 따라서 판독기에 머무는 태그를 표현할 수 있는 새로운 방법이 제시되어야 한다.

이 논문에서는 위와 같은 문제를 해결하기 위하여 RFID 태그의 궤적을 위한 구간 데이터 모델을 정의한다. 이 모델에서는 태그의 궤적을 판독기에 들어올 때 생성되는 시간에 종속적인 선분인 동적 구간(dynamic interval)과 판독기에 나올 때 생성되는 시간에 고정적인 정적 구간(static interval)으로 표현한다. 따라서 판독기에 머무는 태그의 궤적은 시간에 종속적인 동적 구간으로 표현되므로 이러한 태그를 찾을 수 있게 한다.

또한 태그의 추적을 위한 질의를 분류한다. 분류된 질의는 태그의 위치를 찾는 Find 질의, 판독기에 위치한 태그를 찾는 Look 질의, 특정 태그와 같이 위치한 태그를 찾는 With 질의, 태그의 이동 경로를 찾는 Trace 질의이다. 이 논문에서는 이러한 질의를 효율적으로 처리하기 위해 질의의 매개 변수로 태그의 식별자가 들어간다는 특성을 고려하여 객체 식별자를 색인의 도메인으로 추가한다.

제시한 태그의 구간 데이터 모델에 적합한 R-tree 기반 색인 구조인 IR-tree(Interval R-tree)를 제시하며 효율적인 질의처리를 위해 시간에 종속적인 동적 구간의 특성을 고려한 새로운 삽입 및 분할 알고리즘을 제안한다. 마지막으로 다양한 데이터 집합에서 제안된 색인과 기존 알고리즘을 사용하는 색인과의 성능비교를 통하여 색인의 우수성을 입증한다.