DOI: 10.3745/KIPSTD.2009.16-D.1.1

유비쿼터스 컴퓨팅에서 RFID 태그 추적을 위한 태그 궤적 생성 기법

김 종 완[†] · 오 덕 신[††] · 김 기 천[†††]

요 약

RFID 시스템의 주요목적 중 하나는 사물에 태그(tag)를 부착하여 사물의 이동을 추적하는 것이다. 태그는 공간적 위치와 시간정보를 모두 갖기 때문에 기존의 시공간 객체와 같이 궤적을 색인화 할 수 있다. 태그 궤적을 구성하면 보다 효율적인 추적이 가능하지만 현재까지 태그궤적색인에 대한 연구는 많지 않다. 태그가 기존 시공간 객체와 다른 점은 리더(reader)에 진입(enter)한 후 이탈(leave) 함으로써 리더 별로 독립적인 궤적이 생성되는 것이다. 따라서, 리더 사이에는 태그를 찾을 수 없는 궤적 단절구간이 나타나며, 태그의 추적을 어렵게 한다. 또한 리더에 진입 한 후 이탈하지 않은 포인트 태그(point tag)는 궤적을 형성하지 못하기 때문에 질의에서 탐색되지 않는다. 우리는 이러한 문제를 해결하기 위해 각 리더에 분산된 궤적을 하나의 궤적으로 구성하고 포인트 태그를 탐색하는 태그궤적색인인 TR-tree(Tag trajectory R-tree in RFID system)를 제안한다. 실험결과는 리더별 궤적 단절구간을 극복한 TR-tree가 기존의 TPIR-tree 그리고 R-tree 보다 향상된 성능을 나타낸다.

키워드 : 무선전파식별, 태그, R-tree, 시공간 색인, 궤적 색인

Tag Trajectory Generation Scheme for RFID Tag Tracing in Ubiquitous Computing

Jongwan Kim[†] · Dukshin Oh[††] · Keecheon Kim[†††]

ABSTRACT

One of major purposes of a RFID system is to track moving objects using tags attached to the objects. Because a tagged object has both location and time information expressed as the location of the reader, we can index the trajectory of the object like existing spatiotemporal objects. More efficient tracking may be possible if a spatiotemporal trajectory can be formed of a tag, but there has not been much research on tag trajectory indexes. A characteristic that distinguishes tags from existing spatiotemporal objects is that a tag creates a separate trajectory in each reader by entering and then leaving the reader. As a result, there is a trajectory interruption interval between readers, in which the tag cannot be located, and this makes it difficult to track the tag. In addition, the point tags that only enter and don't leave readers do not create trajectories, so cannot be tracked. To solve this problem, we propose a tag trajectory index called TR-tree (tag trajectory R-tree in RFID system) that can track a tag by combining separate trajectories among readers into one trajectory. The results show that TR-tree, which overcomes the trajectory interruption superior performance than TPIR-tree and R-tree.

Keywords : RFID, Tag, R-tree, Spatiotemporal Index, Trajectory Index

1. 서 론

지금까지 바코드가 사물식별 기술로 광범위하게 사용되어 왔으나 바코드는 직접 인쇄 하거나 그려야 하고 종종 사람의 개입을 필요로 하기 때문에 불편하다[1]. 이러한 불편을 해소하고자 새로 나온 기술이 무선전파식별(RFID, Radio Frequency IDentification)이다. 사물을 관리하기 위해 태그(tag)는 재고관리, 항만 컨테이너 관리, 공급망 관리, 위험물 관리 등에서 다양하게 사용된다[2, 3, 9]. 태그가 부착된 사물의 이동을 추적하고 관리하기 위해서는 과거 및 현재 이동경로를 구성하는 궤적 색인(trajectory index)이 필요하다. 태그는 리더의 위치정보와 리더 안에서 경과된 시간에 따라 시공간 정보를 가지며, 이는 기존 이동 객체(moving object)의 시공간 데이터와 유사한 방법으로 관리가 가능하다. 이러한

※ 본 연구는 서울시 산학연 협력사업(CR070019)의 연구비지원에 의해 수행되었습니다.
† 정 회 원 : 건국대학교 컴퓨터공학부 연구교수
†† 정 회 원 : 삼육대학교 디지털경영학부 부교수
††† 종신회원 : 건국대학교 컴퓨터공학부 교수(교신저자)
논문접수 : 2008년 7월 7일
수 정 일 : 1차 2008년 9월 19일, 2차 2008년 11월 11일
심사완료 : 2008년 12월 30일

공통적 특성에도 불구하고 태그 추적을 위한 궤적 구성에 대한 연구는 많지 않다.

태그의 종류는 능동형(active)과 수동형(passive) 태그가 있다. 능동형 태그는 자체 전력으로 항만의 컨테이너와 같이 원거리에서도 리더에 정보를 전송한다. 그러나 본 논문의 연구대상인 수동형(passive) 태그는 자체 전력이 없기 때문에 태그의 데이터를 리더에게 직접 전송하지 못한다. 단지, 리더의 인식영역(interrogation area)에 들어가면 리더의 무선신호(radio frequency)를 받아 자신의 정보를 전송한다. 수동형 태그는 능동형 태그에 비해 가격이 저렴하며 옷, 식료품 등 일상생활에서 널리 사용되기 때문에 논문에서는 수동형 태그의 관리에 초점을 맞춘다.

RFID 시스템의 태그는 다음과 같이 이동 객체와 구별되기 때문에 기존의 시공간 색인을 사용할 수 없다. 즉, 이동객체는 GPS와 같은 위치식별 장치에 의해 자신의 위치 정보를 알고 있으며 주기적으로 자신의 위치를 서버에게 전송한다. 그러나 태그는 위치정보가 없으며 리더에서 읽혀질 때 리더의 위치가 태그의 위치로 인식된다. 이러한 차이점 때문에 새로운 태그 색인이 필요하다.

태그의 궤적은 이동 객체와 동일하게 시간과 공간 값을 갖는 3차원으로 표현된다. RFID시스템의 특성상 태그는 리더 위치를 가지며, 시간은 리더에 들어가는 진입(enter)과 리더에서 나가는 이탈(leave) 이벤트(event)로 생성된다. 이러한 이벤트는 한 리더 안에서 시간 축(axis)에 평행한 시간 궤적이 된다. 그러나 태그의 시간 궤적은 각 리더에서 독립적으로 생성되기 때문에 한 리더와 다른 리더의 궤적들 사이에서 시간 궤적이 끊기는 단절구간이 발생한다. 또한 리더에 진입한 후 이탈하지 않은 태그(point tag)는 시간 궤적이 형성되지 않아 질의에서 탐색되지 않는다. 이러한 시간 궤적의 문제점은 태그의 위치 궤적 형성에도 영향을 미치며 태그를 추적하기 어렵게 하는 요인이다. 기존의 태그 색인인 TPIR-tree[6]는 포인트 태그의 단점을 극복하고자 질의 시점에서 태그의 끝 시간을 현재 시간(now)으로 할당하여 시간 궤적을 형성한다. 그러나 리더 사이의 단절된 시간 궤적이 존재하기 때문에 탐색 성능이 좋지 않다.

본 논문에서는 리더 별로 생성된 태그의 궤적과 재구성 포인트 태그의 궤적을 재구성함으로써 태그 추적성을 향상시킨 R-tree기반의 TR-tree(Tag R-tree in RFID system) 색인을 제안한다. R-tree에서 공간객체를 2차원 데이터로 추정한 최소경계사각형(MBR, minimum bounding rectangle)[10]은 TR-tree에서 시공간을 연결하는 최소경계상자(MBB, minimum bounding box)로 표현한다. 즉, 각 차원을 연결하는 MBB는 색인에서 시공간에 대한 탐색을 보다 효율적으로 수행할 수 있도록 한다. 또한 제안한 색인은 빈번한 태그 이동에 대한 업데이트 성능을 보장하기 위해 해시로 구성된 2차 색인 TL(Tag Link)을 사용한다.

본 논문의 제안 기법은 RFID 태그의 추적성을 향상시키며 다음과 같은 장점을 갖는다.

- 태그 이동에 따른 시공간 궤적을 구성하여 태그 추적이 가능하다.
- 리더 사이에 단절된 태그 궤적을 완전한 궤적으로 구성하여 질의 성능을 향상 시킨다.
- 포인트 태그의 시간궤적을 재구성하여 질의 성능을 향상시킨다.
- 상향식(bottom-up) 갱신을 수행하도록 해시 기반의 2차 색인을 사용하여 동적인 태그이동에 따른 업데이트 성능을 보장한다.

논문의 구성은 다음과 같다. 2절에서는 태그와 궤적 색인을 기술한다. 특히, 태그 궤적을 관리하는 TPIR-tree의 단점을 나열한다. 3절은 RFID 시스템에서 태그 궤적의 특징을 설명하고 궤적의 단절 현상을 극복한 완전한 궤적을 표현한다. 4절은 TR-tree와 2차 색인의 구조에 대해 설명한다. 5절에서는 시뮬레이션을 통해 논문에서 제안한 TR-tree, TPIR-tree 그리고 R-tree를 비교하여 기존 궤적 색인보다 태그의 질의 성능이 더 향상되었음을 보인다. 논문에 대한 결론은 6절에서 요약한다.

2. RFID 태그와 궤적 색인

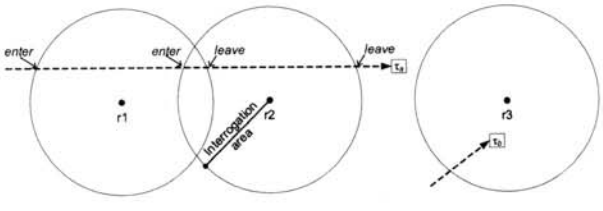
2.1 태그 인식

RFID 시스템에서 객체에 부착된 태그는 리더에 의해 인식된다. 태그는 (그림 1)과 같이 리더 영역으로 들어 갈 때 진입 이벤트가 발생하고 영역을 떠날 때 이탈 이벤트가 발생한다. 이 두 신호에 의해 태그가 리더 안에서 머문 시간을 알 수 있다. 태그의 공간적 위치는 리더와 동일하며, 특정 리더를 이탈하기 전까지는 리더 안에 머문다.

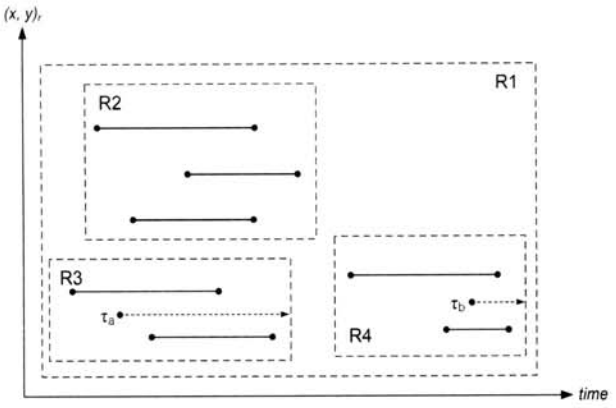
2.2 객체 궤적 색인

객체의 이동은 시간 변화에 따른 이동위치로 구별된다. 색인에서 궤적은 시간과 공간을 결합한 1+2차원으로 표현되며, 과거와 현재 정보를 추적할 수 있다. 공간 객체의 궤적에 대한 연구는 R-tree에 시간 축을 추가한 색인이 많이 연구되었다. 여기서는 앞에서 언급한 이동객체의 궤적을 관리하는 HR-tree[4], STR-tree[5], 그리고 태그 궤적 색인인 TPIR-tree[6]를 살펴본다.

HR-tree는 시간이 경과됨에 따라 또 하나의 색인을 구성하고, 이전 위치를 가리킨다. 이 기법은 객체의 위치가 자주 변경되지 않는다면 과거 위치를 추적하기에 좋다. 그러나 객체의 이동이 빈번하다면 질의 효율은 급격히 떨어지기 때문에 대량의 태그가 빈번히 이동하는 경우에 적합하지 않



(그림 1) 리더에서 태그 이동과 인식



(그림 2) TPIR-tree에서 리더 별로 단절된 궤적과 포인트 태그의 시간 확장

다. STR-tree는 이동 객체에 대한 궤적의 지향(orientation) 점을 찾아 질의를 쉽게 처리할 수 있도록 하였다. 궤적에 대해 시공간 질의 영역에서 어떠한 궤적이 진입(enter), 이탈(leave), 통과(bypass), 교차(cross) 궤적인지 판단할 수 있는 위상(topological) 질의를 지원하며, 동일한 궤적에서 부분 궤적에 대한 질의는 결합(combined) 질의를 사용하여 가능하다. 그러나 태그는 리더단위로 궤적이 유지되므로 앞의 두 가지 형태의 질의처리를 할 수 없다.

TPIR-tree는 태그가 리더에 진입하는 시간과 이탈하는 시간을 기준으로 궤적을 구성하는 색인이다. 그러나 (그림 2)의 MBB R3, R4에서 태그 a, b와 같이 태그가 리더에 진입하고 이탈하지 않은 경우 색인에서 포인트(point)로 존재하기 때문에 궤적이 만들어지지 않는다. 궤적이 만들어지지 않는 태그는 질의와 교차되는 궤적이 없어서 탐색되지 않는다. TPIR-tree는 이러한 단점을 해결하고자 진입 이벤트만 발생한 태그의 시간 축을 현재 시간까지 확장시켜서 질의 결과를 얻도록 하였다. 그러나 여전히 태그의 궤적은 각 리더에 독립적으로 존재하여 특정 시간 윈도우(time window)에서 궤적들이 분할되어있다. 이때, 질의를 위해 확장되는 시간 축은 상위 노드의 MBB까지 확장되는 결과를 가져오기 때문에 MBB들의 중첩이 발생하여 노드 접근 횟수도 증가한다.

TPIR-tree는 RFID 시스템에서 태그의 궤적을 관리하는 보기 드문 색인 중 하나이지만 다음과 같은 단점이 있다.

첫째, TPIR-tree에서 태그의 궤적이 리더 안에서만 존재하고 리더 사이를 이동할 때는 궤적이 존재하지 않는다. 즉, 궤적이 각 리더에만 나타나기 때문에 궤적이 단절된다.

둘째, 태그 궤적 사이의 단절구간을 해결하지 못하여 이동 정보 추적이 어렵다.

셋째, TPIR-tree에서 리더에 진입한 후 떠나지 않은 태그는 궤적이 생성되지 않기 때문에 질의 할 때 시간 축으로 확장하여 처리 한다. 그러나 이 경우 MBB가 확장되어 다른 MBB와 겹치면 태그를 탐색할 때 노드 방문횟수가 증가하여 탐색 성능이 떨어진다.

이러한 단점들은 본 논문에서 몇몇 정의와 실험을 통해 극복될 것이며, 구체적인 방법은 3절에서 자세히 다룬다.

3. RFID 태그 궤적

3.1 태그 궤적의 특징

RFID 시스템에서 태그의 진입과 이탈 신호는 리더를 통해 RFID 미들웨어에 전달된다. 응용레벨에서 태그의 궤적을 구성하기 위해서는 (x, y, t)의 3차원 정보가 필요하며, 이는 리더의 위치와 리더를 방문한 시간에 의해 결정된다.

이미 알고 있는 바와 같이 태그 궤적은 이동객체와 달리 RFID 시스템에서 몇 가지 다른 특징을 갖는다. 기존연구에서도 나타났듯이 리더의 궤적은 각 리더 안에서 시간 축에 평행하게 나타난다[6]. 이는 태그가 리더에 인식된 후 떠나는 시점을 기준으로 궤적이 형성되기 때문이다. 즉, 이동객체는 위치에 따른 경과된 시간을 연속적으로 연결하여 궤적을 구성하지만 태그는 각 리더 안에서 독립적인 궤적으로 나타난다. 본 논문에서는 [6]에서 제시한 태그 궤적을 보다 현실적으로 구성하기 위해 궤적에 대해 다음과 같이 정의한다. 논문에서 대괄호([])는 폐쇄시간 구간을 나타낸다.

[정의 1] 리더에 머문 태그 시간

t를 시간, e는 태그가 리더의 인식영역에 들어가는 이벤트, l 은 인식영역을 벗어나는 이벤트라 하자. 이때, 한 태그가 리더의 인식영역에 들어가는 시간은 t_e, 벗어나는 시간은 t_l 이며, 태그가 리더에서 머문 시간 t_r은 다음과 같다.

$$t_r = \{(t_e, t_l) \mid t_e \le t \le t_l)\} = [t_e, t_l]_r \quad (1)$$

[정의 2] 리더 태그 궤적

x_r을 리더의 x좌표, y_r을 리더의 y좌표라 하자. 태그가 리더에서 머문 시간이 t_r일 때, 리더 안에서 태그의 궤적은 (x_r, y_r, t_r)로 표현된다. 따라서 리더 안에서의 궤적은 (x, y, t)$_r$로 표현되며 태그 궤적 T는 다음과 같다.

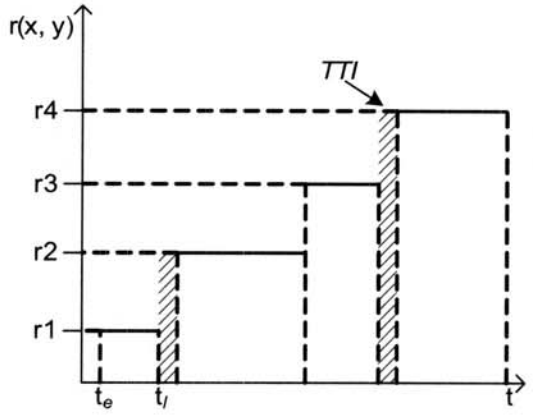
$$T_{r.tag} = \{(x,y,t)_r \mid x = x_r, y = y_r, t = t_r, t_e \le t_r \le t_l\} = (x, y, [t_e, t_l]) \quad (2)$$

태그 궤적은 이동객체의 궤적과 같이 (x_i, y_i, t_i)에서 (x_{i+n}, y_{i+n}, t_{i+n})까지를 연결한 궤적이다. 공간 좌표 (x, y)는 리더의 위치이므로 (그림 3)과 같이 2차원으로 나타내면 리더별로 존재하는 궤적이 더 명확히 표현된다.

태그의 궤적 사이에는 단절 구간이 존재하는데, 이는 각 리더 별로 생성된 태그 궤적이 서로 연속성을 갖지 못하기 때문이다. 태그의 단절된 궤적을 하나의 일관된 궤적으로 구성하기 위해 각 리더 별 궤적에 대해 시간의 전후 관계를 파악할 수 있도록 관리하면 태그의 추적성이 증가한다. 따라서 논문에서는 추적성을 증가시키기 위해 각 리더에서 생성된 궤적들이 시간적으로 연속성을 갖도록 구성한다. 궤적 단절 구간에 대한 정의는 다음과 같다.

[정의 3] Tagless Time Interval(TTI)

태그 τ가 지나가는 두 개의 리더를 각각 r_i, r_{i+1}라 하자. [정의 1]에 의해 τ가 각 리더에서 머문 시간은 [t_e, t_l]r_i, [t_e,



(그림 3) 리더 사이의 시간 궤적 단절 구간

$t_l]r_{i+1}$ 이다. 이때, 두 리더의 진입과 이탈 시간이 $r_i.t_l < r_{i+1}.t_e$ 라면 리더 r_i와 r_{i+1} 사이에는 시간의 단절이 발생한다. 시간 궤적의 단절 구간을 TTI (Tagless Time Interval)라 정의하면 다음과 같다.

$$TTI(tag) = \begin{cases} r_{i+1}.t_e - r_i.t_l & , r_i.t_l \le r_i.t_e \\ 0 & , otherwise \end{cases} \quad (3)$$

3.2 태그 궤적의 구성

이동객체에서는 객체의 위치가 변함에 따라 연속적으로 궤적이 형성된다. 그러나 태그에서는 (그림 3)과 같이 궤적 사이에 단절이 나타난다. 궤적은 한 객체가 이동한 경로의 시공간 좌표들이 연속적으로 연결되는 것이므로 리더를 지나간 태그의 시공간 경로에 단절이 없어야 태그를 추적할 수 있다.

시간의 전후 관계는 (그림 4)와 같이 $t_e < t_l$이며, 두 리더의 인식영역이 겹치지 않는다면 리더 r1과 r2에서의 시간의 관계는 다음과 같다.

$$(t_e, t_l)r_i \le (t_e, t_l)r_{i+1} \qquad (i = 1,2,...,n) \quad (4)$$

두 리더의 인식영역이 겹치면 각 리더에서 생성된 시간 궤적도 겹쳐지며 진입과 이탈 이벤트에 따라 다음의 관계를 갖는다.

$$(t_e)r_i < (t_e)r_{i+1} < (t_l)r_i < (t_l)r_{i+1} \qquad (i = 1,2,...,n) \quad (5)$$

[정의 4] 태그의 시공간 궤적 표현

궤적 단절구간 TTI의 길이를 Δt라 하면 $\Delta t = r_{i+1}.t_e - r_i.t_l$ 이다. 이때, [정의 2]에서 나타낸 궤적은 다음과 같이 확장된 자료구조로 표현된다.

$$<x, y, [\Delta t, t_e, t_l]>$$

시간 궤적의 구조를 $[\Delta t, t_e, t_l]$로 확장해도 태그의 시공간을 표현하는 3차원 구조에는 변함이 없다. [정의 4]의 시간 궤적은 다음과 같은 속성을 갖는다.

- **속성 1:** 리더에 처음 진입한 태그는 ϕ의 Δt와 t_l 을 갖는다.
  태그가 한 리더를 처음 방문하고 안에 머물러 있는 경우에는 [정의 4]에 따라 시간궤적은 $[\phi, t_e, \phi]$로 표현된다. 즉, 리더에 처음 방문한 태그는 궤적을 형성할 수 없는 포인트 정보만 갖는다.
- **속성 2:** 한 리더만 방문한 태그는 ϕ의 Δt를 갖는다.
  태그가 리더를 벗어나면 다른 리더로 진입하기 전까지는 $[\phi, t_e, t_l]$의 시간 궤적을 갖는다. 이때, Δt는 이전 리더와의 차이를 나타내므로 ϕ이다.
- **속성 3:** 리더를 방문한 후 다른 리더에 진입한 태그는 이전 리더와의 시간 간격인 Δt를 유지한다.
  여러 리더를 지나간 태그는 이전 리더에서의 시간 t_l 과 현재 리더에서의 시간 t_e에 대한 Δt가 존재한다. 따라서 시간궤적은 $[\Delta t, t_e, t_l]$ 이다. 특히, 여러 리더를 방문한 태그가 다른 리더에 진입한 경우에는 $t_l = \phi$이 된다.

[정의 5] 포인트 태그(Point Tag)의 궤적 표현

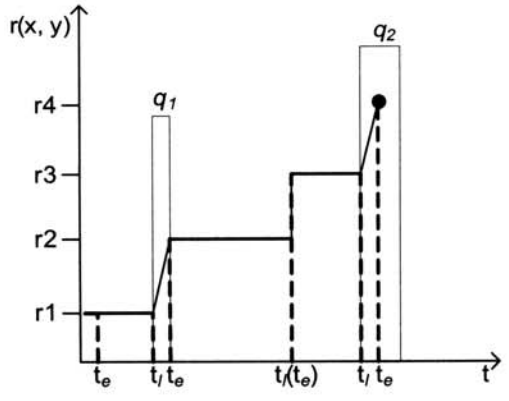
리더에 들어온 진입 이벤트만 있고 이탈 이벤트가 없는 태그를 포인트 태그(point tag) τ_p 라 하자. 이때, 리더에 머물러 있는 태그는 이탈 이벤트가 발생하지 않아서 $[\Delta t, t_e, \phi]$의 시간속성을 갖는다. 포인트 태그의 궤적은 다음과 같이 t_l을 해당 태그가 속한 MBB의 최대 시간 값으로 정의한다.

$$\tau_p.t_l = \text{MAX\_TIME(MBB)} . \quad (6)$$

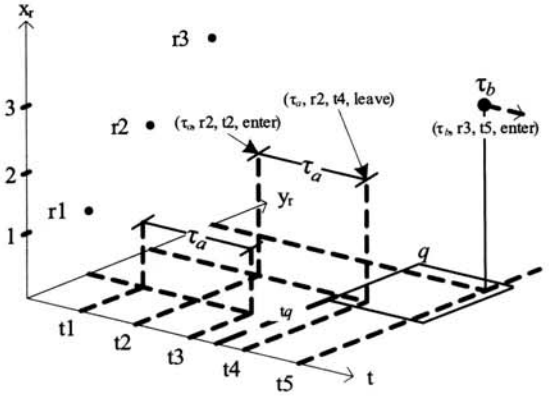
포인트 태그는 리더를 떠날 때야 비로소 궤적으로 표현된다. 그러나 태그가 새로운 리더로 이동한 경우에도 이전 위치와 연결되어야 궤적 구성이 가능하며, 질의 결과에도 포함될 수 있다. 본 논문에서는 [정의 5]와 같이 포인트 태그의 이탈 시간을 해당 MBB의 최대 시간으로 설정함으로써 색인 안에서 탐색이 가능하다. 이때, MBB는 확장되지 않으며 색인에서 상위 노드에 영향을 미치지 않는다. 또한 TPIR-tree에서와 같이 태그 탐색을 위한 포인트 태그의 시간 확장이 없으므로 탐색 연산이 빠르다. 포인트 태그가 리더를 벗어나는 경우 t_l 값은 실제 이탈 시간으로 변경되며, 논문에서 제안한 2차 색인인 Tag Link(TL)을 이용하여 단말 노드에 직접 접근하여 수정한다.

두 리더의 인식영역이 교차하는 경우에는 (그림 1)과 같이 미들웨어에서 시간적으로 빠른 곳의 리더 위치를 저장하고 리더를 벗어나면 최종 위치로 갱신한다. (그림 1)에서 리더 r3에 들어간 태그가 떠나지 않은 경우 태그 궤적은 (그림 5)의 τ_b와 같이 포인트로 표현된다. 이러한 포인트 태그는 궤적을 갖지 않아 질의 q에서 탐색되지 않는다[6]. 즉, 질의 q와 교차되는 시간 선(time line)은 질의 시간 t_q와 만나는 태그 τ_a를 반환한다.

[정의 4]의 속성들을 (그림 4)에 적용한 태그의 궤적은 <표 1>과 같다. 이때, 태그의 공간 좌표는 진입 이벤트가



(그림 4) 리더 별 궤적의 연결과 포인트 태그



(그림 5) 리더의 태그 궤적에 대한 3차원 표현

발생한 리더위치를 따른다. 한 리더를 벗어나 아직 다른 리더에 진입하지 않은, 즉, 어떠한 리더 인식영역에도 속하지 않은 태그는 데이터베이스에 이전 위치까지만 저장되어 있으며, 현재 위치를 묻는 질의에는 나타나지 않는다.

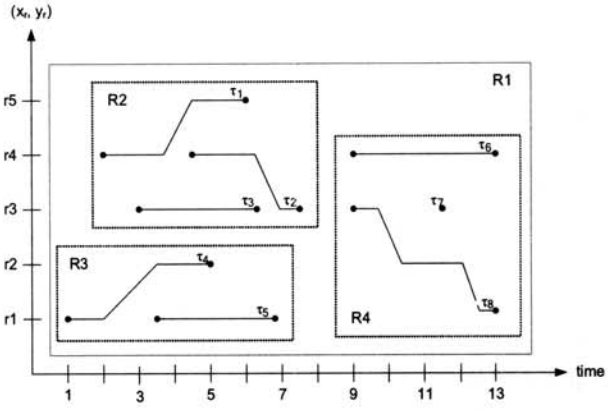
4. TR-tree

4.1 색인 구조

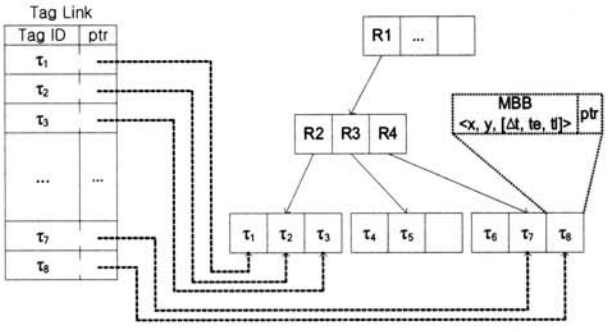
TR-tree는 R-tree를 기반으로 하는 높이 균형 트리이며, 태그 궤적을 저장하도록 R-tree를 확장하였다. (그림 6)은 리더 안에서 생성된 태그 색인을 2차원으로 나타낸 것이다. 이러한 궤적은 TR-tree에서 (그림 7)과 같이 저장된다. 트리의 각 노드는 최대 M개의 엔트리(entry)를 포함하며 단말

〈표 1〉 리더 궤적 표현

리더	Δt	t_e	t_l	(x, y)	비고
r1	ϕ	r1.t_e	R1.t_l	r1	리더통과
r2	r2. t_e-r1.t_l	r2.t_e	R2.t_l	r2	리더통과
r3	r3. t_e-r2.t_l	r3.t_e	R3.t_l	r3	리더통과
r4	r4. t_e-r3.t_l	r4.t_e	ϕ	r4	포인트 태그



(그림 6) 단절 없는 Tag trajectory 표현



(그림 7) TR-tree 구조와 TL 색인

노드의 엔트리는 <*MBB, t_{id}, object-pointer*>로 구성된다. *MBB*는 태그 t_{id}의 위치와 시간을 3차원으로 표현한 (x_r, y_r, t)의 구조를 갖는다. x_r, y_r은 태그가 이동한 공간 궤적을 구성하는 위치로써 리더의 위치를 의미한다. t는 태그가 리더 사이를 이동한 시간이며, [Δt, t_e, t_l]의 형식으로 관리된다. 데이터베이스에 저장된 태그 t_{id}의 실제 위치는 *object-pointer*가 가리킨다. 비단말 노드의 엔트리는 하위 노드를 포함하는 *MBB*와 하위 노드에 대한 포인터인 *child-pointer*로 구성되는 <*MBB, child-pointer*>형식을 갖는다. 한 노드의 크기는 디스크 페이지의 블럭 단위인 4KB이다.

전통적인 R-tree에서 갱신은 하향식(top-down) 방식으로 삭제와 삽입 연산을 수행한다. 이 방식은 루트 노드부터 객체가 저장된 단말 노드까지 탐색하여 갱신대상을 찾고, 해당 노드를 삭제한 다음 정보가 변경된 객체를 삽입한다. 따라서 한 객체의 정보를 변경하기 위해 단말 노드에 접근하는데 많은 노드 접근이 필요하다[7, 8]. 또한 새로운 객체가 단말 노드에 삽입된 경우 해당 노드에서 오버플로우(overflow)가 발생하면 노드가 분할(split)되며 그 영향은 상위 노드로 전파되어 많은 노드 접근이 발생한다.

논문에서 제안한 색인에서는 (그림 7)과 같이 태그의 빈번한 이동에 따른 TR-tree의 업데이트 성능을 향상시키고자 2차 색인인 Tag link(TL) 색인을 사용한다. TL은 태그 ID와 TR-tree의 단말 노드의 엔트리들과 연결되는 단말 노드에 대한 포인터를 갖는 해시 색인이다. 우리는 태그의 위

치 및 시간 갱신으로 발생하는 연산을 최소화하기 위해 TL을 이용하여 단말 노드에 직접 접근한다. 처음 색인을 구축할 때, 리더를 방문한 태그는 TR-tree의 루트 노드로 부터 해당 노드를 찾아 삽입되며, TL에도 저장된다. 이때, TL의 포인터 *ptr*은 TR-tree 색인에서 해당 태그가 있는 단말 노드의 엔트리를 가리킨다.

태그 이동에 의한 시간 갱신은 다음과 같이 진행된다. (그림 6)에서 보는 것과 같이 태그 τ1이 리더 4를 시간 t4에서 이탈하면 먼저, TL에서 태그를 찾아 단말 노드의 시간 속성을 [ϕ, t_e, t_l]로 변경한다. 이러한 과정을 R-tree에서 진행한다면 적어도 4번의 노드 접근비용이 발생하지만 여기서는 두 번의 노드 접근이면 충분하다. TL이 갖는 장점으로는 첫째, 태그의 이동에 따른 동적 갱신 비용이 저렴하다. 즉, 태그가 리더를 이탈할 때 변경되는 시간 및 위치 정보를 갱신하기 위해 루트 노드에서 단말 노드까지 찾는 것보다 훨씬 적은 횟수로 노드 접근이 가능하다. 둘째, 색인에서 하나의 태그를 찾는 다면 한번의 노드 접근으로 해당 태그를 찾을 수 있다.

4.2 알고리즘

R-tree계열에서 색인의 성능을 좌우하는 것은 노드 접근 횟수이다. 주로 삽입, 갱신에서 노드 접근이 발생하는데, 이는 디스크 입출력을 증가시켜 색인의 성능에도 영향을 미친다.

(그림 8)은 TR-tree의 갱신(update) 알고리즘이다. 입력은 태그ID, 리더의 위치(*newLocation*), 타임스탬프(*newTimestamp*)이다. 태그 정보를 갱신하기 위해서는 TL를 경유하여 단말 노드의 해당 엔트리에 접근한다 (줄 번호: 1-4). 먼저, 태그가 방문한 리더의 위치가 해당 노드의 MBB안에 있으면 해당 엔트리의 위치 정보를 수정한다 (줄 번호: 5). 이때, 노드 분할이나 MBB의 재조정은 발생하지 않는다. 태그의 시간 또한 MBB를 벗어나지 않았다면 새로운 값으로 수정하기만 한다 (줄번호: 6). 만약, *newLocation*과 *newTimestamp*가 MBB를 벗어난다면 엔트리를 갱신하고 루트 노드에서 하향식으로 탐색하여 엔트리를 삽입한다 (줄 번호: 7). 색인에 없는 태그가 추가되면 하향식으로 탐색하여 단말 노드와 TL에 삽입한다 (줄 번호: 10-11) 새로운 태그가 삽입된 노드의 엔트리수가 *n*(entry)=*M*인 경우 노드를 분할하고 TL의 포인터도 엔트리와 연결시킨다(줄번호: 12-15). 알고리즘에서는 태그의 경과 시간이 MBB의 최대 시간을 벗어나지 않으면 노드 분할이 일어나지 않는다. 즉, 한 태그의 시간이 MBB 범위에서 변하는 경우 해당 단말 노드에만 접근하면 되기 때문에 갱신은 지역적으로 발생한다. 물론, 시간이 현재 MBB를 벗어나거나 리더를 이탈하면 새로운 MBB를 구성한다.

TR-tree에서 태그 탐색 유형은 다음과 같이 4가지로 구분된다. (그림 9)는 질의 유형을 반영한 기본적인 탐색 알고리즘이다.

1. 태그 객체 질의(OBJECT query): Q=(τ_{id}), TL을 참조하여 태그 τ_{id}의 현재 위치와 시간 등 정보를 반환한다.
2. 시간 질의(TIME query): Q=(t_s, t_e), 지정한 시간 구간에 있는 모든 태그를 반환한다.
3. 궤적 질의(TRAJECTORY query): Q=(τ_{id}, [t/l]), 태그 τ_{id}에 대한 시간(time), 또는 위치(location) 궤적을 반환한다.
4. 구간 질의(SCOPE query): Q=((x_1, x_2), (y_1, y_2), [t_s, t_e]), 지정한 공간 범위에서 시간 구간에 해당하는 태그를 반환한다. 이때, 공간 좌표를 지정하지 않으면 시간 질의와 동일하며, 시간정보가 없으면 공간 범위의 모든 태그를 탐색한다.

```
Algorithm Tag trajectory update and insertion: Update and Insert a new index entry into TR-tree
Procedure TagUpdateInsert (tagID, newLocation, newTimestamp)
Input: tagID is a tag identification, newLocation is a position,
       newTimestamp is time that a tag's entered or leaved time in a reader.
Begin
1: Seek the tagID in the TL index;
2: if tagID exists in TL
3: then
4:     Access to the leafNode;
5:     Case newLocation < MBB : Update location in an entry;
6:     Case newTimestamp < MBB : Update timestamp in an entry;
7:     Case (newLocation, newTimestamp) > MBB : Issues a top-down update;
8:         return;
9: /* tagID is a new tag */
10: Traverse the index to insert tagID from the root node;. /*Top-down*/
11: Insert a tagID into TL;
12: If a leaf node overflows
13: then
14:     Split the node in TR-tree;
15:     The pointer in TL connects to a leaf entry;
End
```

(그림 8) TR-tree 갱신과 새로운 태그 삽입 알고리즘

```
Algorithm Tag search: Search a tag in TR-tree
Procedure TagSearch (tagID, tagLocation, tagTimeInterval)
Input: tagID is a tag identification, tagLocation is tag location, tagTimeIngterval is time of a tag
Begin
1: Load HEAD-pointer in memory;
2: Select Case option
3:     Case tagID : /* OBJECT query*/
4:         Seek the tagID in the TL index;
5:         if tagID exists in TL
6:         then return an object;
7:     Case tagTimeInterval : /* TIME query */
8:         Search tags between start time and end time of the tagTimeInterval;
9:         tagObjectList += object;
10:          return tagObjectList;
11:    Case (tagID, [tagLocation/tagTimeInterval]) : /* TRAJECTORY query */
12:        Seek the tagID in the TL index and access to an entry;
13:        return a trajectory in location or time;
14:    Case (tagLocation, tagTimeInterval) : /* SCOPE query */
15:        Traverse the TR-tree with location (x1, x2), (y1, y2) and time interval [ts, te];
16:        If found an object
17:        then tagObjectList += object;
18:             return tagObjectList;
19: End Select
End
```

(그림 9) TR-tree의 탐색 알고리즘

5. 실 험

5.1 실험 환경

우리는 R-tree와 TPIR-tree 그리고 본문에서 제안한 TR-tree를 C++로 구현하여 실험하였다. 실험은 Windows XP 운영체제가 설치된 1기가 메모리의 PentiumIV 2.6Ghz 컴퓨터에서 수행하였으며, R-tree는 시간을 저장할 수 있도록 엔트리를 수정하였다. 실험 데이터는 태그 생성기를 개발하여 만들었으며 <tag_id, (x, y), [enter_time, leave_time]>의 구조이다. 포인트 태그의 시간은 개방시간구간인 [enter_time, leave_time) 구조를 포함하며, 실험에서는 포인트 태그 비율을 30, 60, 90% 일 때로 구분하여 성능을 평가하였다. 태그의 시공간 데이터는 [0, 1]의 구간에 균일하게 분포한다고 가정한다. 실험에서는 기본적으로 <표 2>와 같은 옵션을 사용하였다. 그러나 각 실험에 따라 변경된 옵션들은 해당 실험에서 별도로 설명한다.

TR-tree의 강점인 포인트 태그 궤적에 대한 성능을 측정하고자 데이터 셋의 시간 구간은 폐쇄 시간구간인 [t1, t2]와 개방 시간구간인 [t1, t2)로 구성하였다. 폐쇄 시간구간은 리더에 들어간 진입과 이탈 이벤트가 모두 있는 태그이며, 개방 시간구간은 한 리더에서 들어간 후 이탈 이벤트가 발생하지 않은 태그이다. 개방시간을 갖는 태그가 색인에 삽입되면 해당 MBB의 최대 값이 t2에 저장되어 폐쇄구간이 설정된다. 이러한 태그들은 수정될 확률이 폐쇄구간의 태그들보다 높기 때문에 개방구간 태그의 비율은 색인의 성능에 영향을 미친다.

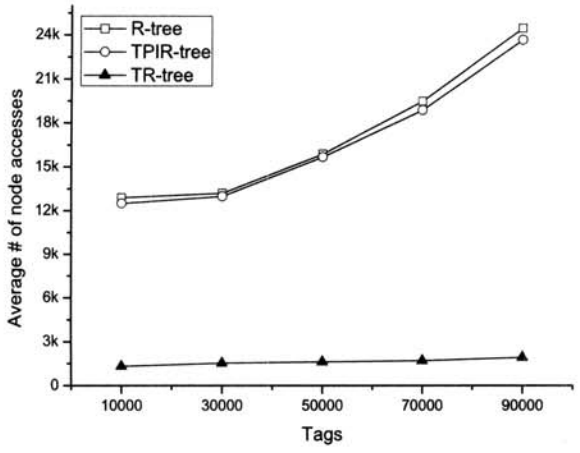
5.2 TR-tree의 탐색 비용

본 절에서는 논문에서 제안한 TR-tree와 태그 색인인 TPIR-tree, R-tree를 비교한 탐색 비용에 대해 설명한다. (그림 10)은 질의 유형 중 OBJECT 질의를 사용해 TR-tree, TPIR-tree, R-tree에서 하나의 태그를 탐색 할 때 평균 노드 접근 횟수를 비교한 것이다. 결과는 서로 다른 질의문 200개로 구성된 5개의 질의 집합(query workload)에 대한 평균이다. (그림 10)의 OBJECT 질의 결과에서 TL을 사용한 TR-tree의 경우 단말 노드에 직접 접근하기 때문에 평균적으로 동일한 결과를 보인다. 그러나 R-tree는 루트 노드에서부터 탐색이 시작되므로 노드 접근 횟수가 크게 증가한다. TPIR-tree 또한 R-tree와 동일한 구조에 각 태그의 위치를 저장하기 때문에 같은 모습으로 증가한다. 이때, TR-tree의 노드 접근 횟수가 약간 증가하고 있는데 이는 MBB 사이의 중첩에 의해 역탐색(backtracking)이 발생하기 때문이다.

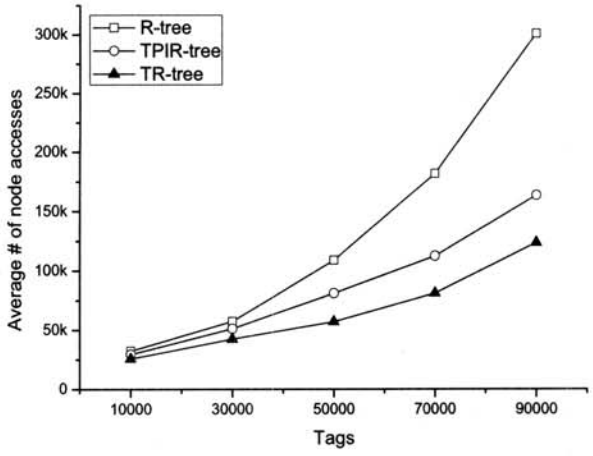
(그림 11)은 특정 시간 구간에 있는 모든 태그를 반환하

〈표 2〉 실험 파라미터

구분	설정	비고
태그 시간 구간	[t1, t2] 또는 (t1, t2)	폐쇄/개방 구간
태그 수	90,000	태그
포인트 태그 비율(%)	30, 60, 90	포인트 태그



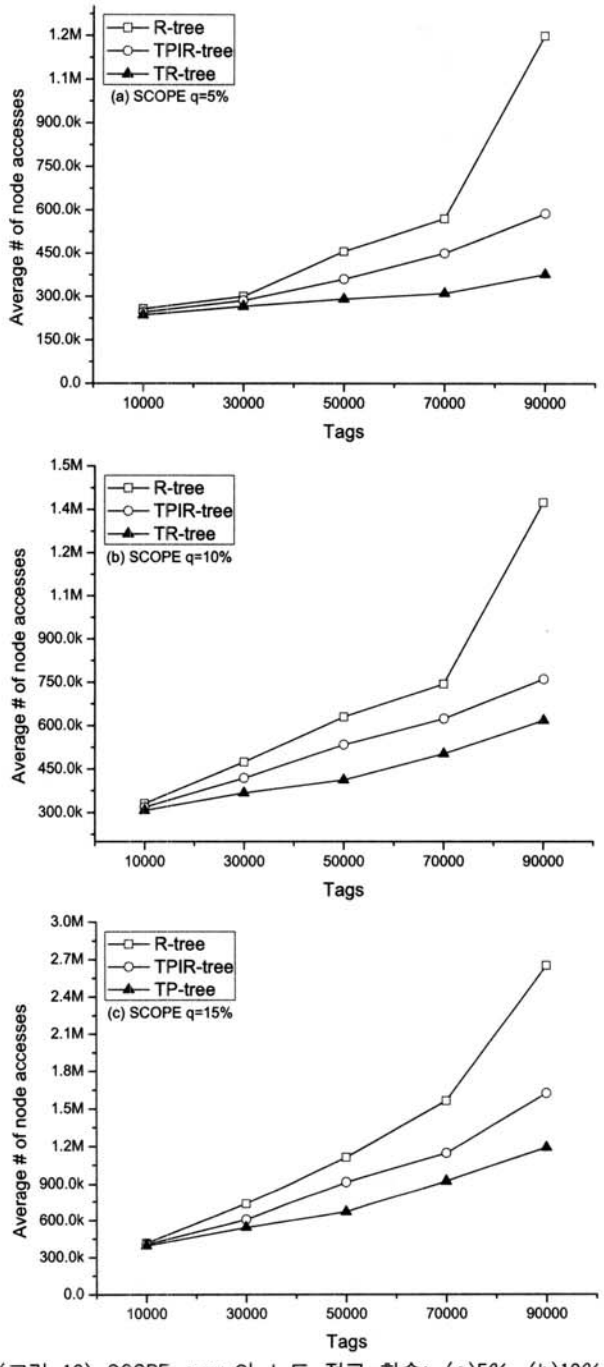
(그림 10) OBJECT 질의에서 태그 수에 따른 평균 노드 접근 횟수



(그림 11) TIME 질의에서 태그 수에 따른 노드 접근 횟수

는 TIME 질의 결과이다. 태그가 400개 일 때는 두 색인의 노드 접근 횟수의 차이가 근소하였으나 600개부터는 큰 차이로 벌어진다. 이는 태그 수가 증가할수록 TPIR-tree에서 MBB가 확장되는 현상이 나타나기 때문이다.

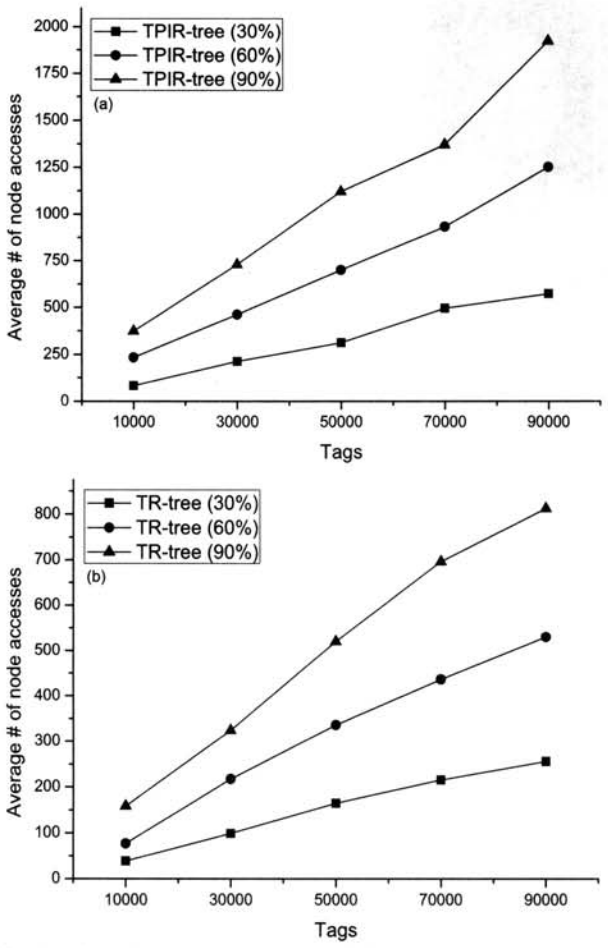
(그림 12)는 시공간 범위를 모두 지정한 SCOPE 질의에서 노드 접근 횟수이다. 질의 문은 500개이며, 각 차원을 중심으로 5%, 10% 15%의 범위 내에서 탐색하였다. 이 범위는 태그 개수가 증가함에 따라 동일하게 적용된다. R-tree는 시간을 저장할 수 있도록 엔트리를 확장하였으며, 시간축의 궤적은 TPIR-tree와 같이 리더 안에서의 궤적으로 구성하였다. (그림 12(a))에서 TR-tree는 가장 적은 노드 접근 횟수를 보인다. 또한 태그의 수가 증가하여도 노드 접근 횟수의 증가 기울기는 완만하게 증가한다. 그러나 차원의 범위를 넓힐수록 노드 접근 횟수와 기울기가 증가한다. 이는 찾아야 하는 태그가 많아 졌기 때문이며, 세 기법 중 R-tree의 노드 접근 횟수가 가장 많다. (그림 12)의 (a), (b), (c)에서 모두 큰 노드 접근 횟수를 기록했는데 이는 각 차원에서 중복이 발생하기 때문이다. TPIR-tree는 시간 축을 현재로 확장함으로써 MBB의 겹침이 증가한 결과를 보이지만 R-tree보다는 작다.



(그림 12) SCOPE query의 노드 접근 횟수: (a)5%, (b)10%, (c)15%

5.3 TR-tree의 갱신 비용

(그림 13)은 TPIR-tree와 TR-tree를 대상으로 포인트 태그 비율에 따른 갱신 비용을 측정한 것이다. 즉, 태그 데이터 집합에서 리더 안에 진입하여 머물러 있는 태그 비율에 따른 노드 접근 횟수를 측정하였다. 태그는 동적으로 이동하기 때문에 포인트 태그가 많이 발생한다. 이러한 태그의 특성을 반영하고자 포인트 태그의 비율을 기준으로 노드 접근 횟수를 측정한 것이다. 포인트 태그 비율이 높고 태그의 수가 증가 할수록 노드 접근 횟수는 높아진다. 그러나 TPIR-tree가 훨씬 높은 횟수를 보이는데, 이는 갱신 대상 태그를 찾기 위



(그림 13) 포인트 태그 비율에 따른 (a)TPIR-tree와 (b) TR-tree의 태그 개신 비용

해 TPIR-tree가 하향식으로 탐색하기 때문이다. 성능 저하의 또 다른 이유는 포인트 태그(TPIR-tree에서는 *nowEntrys*라 함)를 만나면 MBB가 현재 시간(*now*)까지 확장되어 역탐색이 발생하여 노드 접근 횟수가 증가하기 때문이다[6]. 이에 반해, TR-tree는 TL을 경유하여 갱신할 엔트리에 접근하기 때문에 현재 MBB를 벗어나지 않는 경우 노드 분할이 발생하지 않는다. 즉, MBB의 확장이 발생하지 않으므로 상대적으로 훨씬 적은 노드 접근 횟수를 갖는다.

기존 색인에서는 탐색의 효율성만을 강조하였으나. 논문에서 제안한 TR-tree는 탐색과 갱신을 모두 고려하였다. 갱신에서도 TR-tree가 좋은 성능을 나타내는 이유는 TR-tree는 TL색인을 경유해서 단말 노드에 직접 접근하여 갱신하기 때문이다. 또한 변경된 시간이 현재 MBB의 최대값을 초과하지 않으면 MBB 확장과 노드 분할이 발생하지 않기 때문에 갱신결과가 상위 노드에 전파되지 않는다. 따라서 갱신을 위한 TR-tree의 노드 접근 횟수는 매우 낮다.

6. 결 론

태그의 리더 의존적인 궤적은 태그의 궤적을 리더 단위로 생성하기 때문에 궤적이 단절되어 태그의 추적을 어렵게 한다. 이러한 문제를 해결하기 위해 우리는 논문에서 R-tree를 기반으로 한 TR-tree를 제안하였다. 이 색인은 리더 별로 단절된 태그 궤적을 이동객체의 궤적과 같이 연속적인 궤적으로 구성하여 단절구간을 없앴다. 또한 동적으로 이동하는 태그의 갱신이 효율적으로 수행되도록 단말 노드의 모든 엔트리와 연결되는 해시 기반의 TL색인을 부가적으로 사용하였다.

실험결과에서 알 수 있듯이 각 리더의 궤적을 연결하여 단절구간을 제거함으로써 질의 영역이 단절구간과 만날 경우 태그를 탐색할 수 없었던 단점을 해결하였다. 또한 리더 안에 머물러 있던 포인트 태그를 탐색할 수 있게 되었다. TR-tree는 포인트 태그 비율과 태그의 수가 증가해도 노드 접근 횟수가 기존 색인과 비교하여 점증적으로 감소하고 있는 것을 알 수 있다. 따라서, 태그를 빨리 추적하고자 하는 RFID시스템에서 TR-tree를 사용하면 많은 성능 향상을 기대할 수 있다.

참 고 문 헌

[1] H. Vogt, "Efficient Object Identification with Passive RFID Tags". LNCS2414, Springer, pp.98-113, 2002.

[2] K. Finkenzeller, 'RFID Handbook: Radio-Frequency Identification Fundamentals and Applications', John Wiley & Sons, 2000.

[3] R. Angeles, "RFID Technologies: Supply-chain Applications and Implementation Issues," Information systems management, 2005.

[4] M. A. Nascimento and J. R. O. Silva, "Towards historical R-trees," Proceedings of the 1998 ACM Symposium on Applied Computing, pp.235-240, 1998.

[5] D. Pfoser, C. S. Jensen and Y. Theodoridis, "Novel Approaches to the Indexing of Moving Object Trajectories," Proceedings of the 26th International Conference on Very Large Databases, pp.396-406, 2000.

[6] C. H. Ban, B. H. Hong and D. H. Kim, "Time Parameterized Interval R-tree for Tracing Tags in RFID Systems," LNCS 3588, Springer, pp.503-513, 2005.

[7] D. Kwon, S. Lee and S. Lee, "Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree," Proceedings of the Third International Conference on Mobile Data Management, pp.113-120, 2002.

[8] M. L. Lee, W. Hsu, C. S. Jensen, B. Cui and K. L. Teo, "Supporting Frequent Updates in R-Trees: A Bottom-Up Approach," Proceedings of the 29th international conference on Very large data bases, Vol.29, pp.608-619, 2003.

[9] 박재걸, 채홍석, "RFID 미들웨어 표준 아키텍처에 기반한 적응적 부하 분산 방법", 정보처리학회논문지D, 제15-D권 제1호, pp.73-86, 2008.

[10] A. Guttman, "R-trees: A dynamic index structure for spatial searching," ACM SIGMOD conference on Management of data SIGMOD, Vol.14, pp.47-57, 1984.

김 종 완

e-mail : wany@korea.ac.kr
1991년 삼육대학교 경영학과(학사)
2001년 숭실대학교 컴퓨터학과(공학석사)
2007년 고려대학교 컴퓨터학과(이학박사)
2007년~2008년 건국대학교 BK21 Post-Doc.
2008년~현 재 건국대학교 컴퓨터공학부 연구교수, 삼육대학교 외래교수
관심분야 : 모바일 데이터 관리, 위치기반서비스(LBS), RFID/USN, 지능형교통시스템(ITS) 및 VANET 등

오 덕 신

e-mail : ohds@syu.ac.kr
1982년 삼육대학교 경영학과(학사)
1988년 단국대학교 경영학과(석사)
2000년 상명대학교 경영학과 박사수료
1997~현 재 삼육대학교 디지털경영학부 부교수
관심분야 : 경영정보시스템, 전자상거래, e-Learning 등

김 기 천

e-mail : kckim@konkuk.ac.kr
1988년 서울대학교 전산학(학사)
1992년 Northwestern Univ. (미국) Computer Science(공학박사)
1992년 8월~1996년 4월 한국통신기술(주) 연구소 선임연구원
1996년 4월~1998년 2월 신세기통신(주) 차장/팀장(책임연구원)
1998년 3월~현 재 건국대학교 컴퓨터공학부 교수
관심분야 : 유비쿼터스 컴퓨팅, 미래인터넷, 모바일 통신, 지능형교통시스템(ITS) 및 VANET 등