RFID 태그 객체의 간격 데이터 색인*

이기형° 반재훈+ 김동현++ 홍봉희°
부산대학교 컴퓨터공학과° 경남정보대학 인터넷응용계열+ 동서대학교 소프트웨어전문대학원++
{lievebejust, bhhong}@pusan.ac.kr, chban@kit.ac.kr+ pusrover@dongseo.ac.kr++

Indexing of Interval Data of RFID Tag Objects

KiHyung Lee° ChaeHoon Ban+ DongHyun Kim++ BongHee Hong°
Dept. of Computer Engineering, Pusan National University°
Subdivision of Internet Application, KyungNam College of Information & Technology+
Graduate School of Software, Dongseo University++

요 약

최근 유비쿼터스 환경에서 객체에 태그를 장착하여 위치를 추적하는 응용분야가 늘어가고 있는 추세이며, 이러한 응용에서 빈번히 사용되는 질의는 객체의 위치를 찾는 find와 특정 위치의 객체를 찾는 look 질의가 있다. 두 질의에서 처리되는 데이터는 시간과 공간을 포함한 다차원 대용량 데이터이며 과거 및 현재 상태의 검색을 지원해야 하므로 효율적인 질의 처리를 위해서는 태그 객체를 위한 새로운 데이터 모델과 색인이 필요하다.

본 논문에서는 태그 객체를 간격 데이터로 정의하고 과거 및 현재 데이터에서 find와 look 질의를 처리할 수 있는 색인 구조를 제안한다. 제안하는 색인에서 노드에 오버플로우가 발생할 경우 새로운 단말 노드 분할 정책을 사용하여 분할하며 성능 평가를 통해서 기존 정책보다 우수함을 증명한다.

1. 서론

RFID(Radio Frequency IDentification)는 무선 주파수를 이용하여 태그를 장착한 객체를 자동으로 인식하고 확인하는 기술로서 최근 유비쿼터스 환경에서 RFID를 사용하는 응용분야가 정부와 관련업체의 주목을 받고 있다.

이러한 응용분야에서 빈번히 사용되는 질의는 find와 look 질의가 있다. find 질의는 "22번 컨테이너가 9시에 있었던 위치는?"과 같이 객체 식별자와 시간이 주어질 때 위치를 찾는 질의이며, look 질의는 "9번 톨게이트를 3시에 지나간 컨테이너는?"과 같이 특정 위치와 시간이 주어질 때 해당되는 객체를 찾는 질의이다.

위의 질의에서 처리되는 데이터는 공간과 시간을 포함하는 다차원 대용량 데이터이며, 응용의 특성상 과거 이력 및 현재 상태의 검색을 지원해야 한다. 따라서 이를 위해 새로운 데이터 모델과 효율적인 질의 처리를 위한 색인 구조가 필요하다.

본 논문은 태그 객체를 위한 데이터 모델 및 질의를 제안하고 효율적인 질의처리를 위한 색인 구조를 제시한다. 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 소개하며, 3장에서는 대상환경과 문제를 정의한다. 4장에서는 데이터 모델과 질의를 제안하고, 5장에서는 태그 객체를 위한 색인 구조 및 삽입정책과 분할 정책에 대해서 기술한다. 6장에서는 실험을 통해서 기존 색인과의 성능을 비교하고 마지막으로 7장에서는 결론 및 향후 연구를 기술한다.

2. 관련연구

공간 데이터를 위한 색인인 R-tree[1]는 단말 노드에 데이터를 가지는 B-tree와 유사한 높이-균형 트리이다. 각 노드는 디스크 페이지에 대응되고 색인 구조는 질의에 있어서 최소한의 노드만을 검색하도록 설계된다. 데이터 삽입 연산은 루트에서부터 최소 영역 확장을 가지는 엔트리의 하위 노드를 선택함으로써 단말 노드를 찾고 데이터를 저장한다.

R*-tree[2]는 재삽입 정책으로 노드 겹침을 최소화하여 [1]보다 향상된 검색 성능을 제공한다. 노드 오버플로우가 발생할 경우 재삽입을 먼저 수행하고, 그 노드에서 다시 오버플로우가 발생하면 분할하는 기법을 사용한다. 그리고 데이터 삽입 시 최소 영역 확장과 최소 겹침 정책을 사용하여 데이터가 삽입될 단말 노드를 선택한다. 그러나 최소 겹침 알고리즘은 좀더 많은 CPU 연산을 요구하고 공간 데이터에서 높은 검색 성능을 보이지 않기 때문에 단말 노드를 엔트리로 가지는 비단말 노드에서만 최소 겹침 알고리즘을 사용한다.

이동 객체의 궤적을 위한 3DR-tree[3]는 1차원 시간과 2차원 공간을 따로 분리하여 저장하지 않고 하나의 3차원 색인에 저장한다. 삽입 정책과 분할 정책은 기존 R-tree의 삽입 및 분할 정책을 그대로 사용하기 때문에 시간에 따라 색인 값이 변하는 연대기적 데이터에는 공간활용도에 있어서 비효율적이다.

3. 대상환경 및 문제 정의

3.1 대상 환경

RFID 시스템은 태그(tag)와 판독기(reader)로 구성된다[4]. 태그는 고유 식별자(*tid*)를 가지며, 물류 관리나 위치 추적과 같은 응용에서 제품이나 컨테이너와 같은 이동성이 있는 객체에 장착된다. 판독기는 인식 영역(*loc*)에 존재하는 태그 객체를 인식하고 위치 데이터를 서버로 전송한다. 예를 들어서 식별자 *tid*를 가지는 태그 객체가 판독기의 인식 영역(*loc*)으로 이동할 때, 시간 *t*에서 인식 영역 안으로 들어갈 경우 enter 이벤트가 발생하고 (tid, loc, t_{enter})가 서버로 전송된다. 반대로 객체가 인식 영역을 벗어날 경우 leave 이벤트가 발생하며 (tid, loc, t_{leave})가 서버로 전송된다. 이때 두 이벤트의 *tid*와 *loc*는 동일하며 t_{leave}는 t_{enter}보다 항상 크다. 이렇게 수집된 데이터는 하나의 다차원 색인에 저장되고 검색된다.

3.2 문제 정의

태그 객체를 위한 색인 구조로 기존의 색인 방법을 사용하면 다음과 같이 두 가지 문제점이 있다. 첫째, 기존 시공간 데이터를 위한 색인들은 객체 식별자를 색인 값으로 사용하지 않고 속성으로서 저장하기 때문에 객체 식별자를 인수로 가지는 find질의를 효율적으로 수행할 수 없다. 예를 들어서, "22번 컨테이너가 9시 있었던 위치는?"과 같은 질의를 처리하기 위해서는 주어진 시간에 포함되는 모

*본 연구는 한국학술진흥재단 2004년도 지방대학육성지원사업의 연구(D00319) 지원으로 수행하였음

든 객체를 탐색하고 객체 식별자로 여과해야 하므로 탐색 범위가 넓어 비효율적이다. 따라서 find 질의를 효율적으로 처리하기 위해서는 객체 식별자를 색인의 도메인으로 유지하고 있어야 한다.

둘째, 기존 색인은 두 점을 선분으로 연결한 과거 데이터만 저장하기 때문에 현재 데이터와 관련된 질의를 처리할 수 없는 문제점이 있다. 예를 들어서 enter와 leave 이벤트가 발생한 태그 객체의 데이터(두 이벤트를 연결한 선분)는 색인에 저장되지만 enter 이벤트만 발생한 태그 객체의 데이터(enter 이벤트에 해당하는 점)는 색인에 저장되지 않는다. 이런 경우 "9번 톨게이트에 현재 위치한 컨테이너는?"과 같이 현재 위치와 관련된 질의를 처리할 수 없는 문제점이 있다.

본 논문에서는 이러한 문제점들을 해결하기 위해서 태그 객체의 데이터 모델을 정의하고 새로운 색인 구조를 제시한다. 그리고 태그 객체와 관련된 질의를 효율적으로 처리하기 위해서 새로운 삽입 정책과 분할 정책을 제시한다.

4. 질의와 데이터 모델

표 1은 RFID 시스템에서 사용되는 질의 종류를 보여준다. find 질의는 객체 식별자와 시간이 주어질 때, 객체의 위치를 찾는 질의이고 look 질의는 공간 영역과 시간이 주어질 때, 주어진 질의 영역에 위치하는 객체 식별자를 찾는 질의이다. with는 주어진 객체 식별자와 동일한 위치를 가지는 다른 태그 객체를 찾는 질의이고 find와 look을 이용해서 처리할 수 있다. 예를 들어서, find를 수행하여 객체의 위치를 찾고 이러한 위치 값을 이용하여 look 질의를 수행하면 동일 위치를 가지는 해당 태그 객체의 식별자를 얻을 수 있다. history는 특정 태그 객체가 과거에 머물렀던 모든 판독기의 위치를 찾는 질의로서 find를 이용하여 처리할 수 있다.

표 1: 질의 종류

질의 종류	리턴 값	질의 예
find(*tid*, *time*)	*loc* set	*tid2*가 2시에 있었던 위치를 찾아라
look(*loc*, *time*)	*tid* set	A지역에서 2시에 있었던 제품을 찾아라
with(*tid*, *time*)	*tid* set	*tid2*와 2시에 같은 위치에 있었던 다른 제품을 찾아라
history(*tid*)	*loc* list	제품 *tid2*의 전체 이동 경로를 찾아라

본 논문에서는 태그 객체와 관련된 기본적인 질의인 find와 look을 처리하기 위해서, 태그 객체를 간격(interval) 데이터로 정의한다. 이러한 데이터는 객체 식별자(*tid*), 판독기의 위치(*loc*), 시간 간격(*time interval*)으로 구성되고 하나의 다차원 색인에 저장된다. 이때 find 질의를 효율적으로 처리하기 위해서 위치와 시간뿐만 아니라 객체 식별자도 색인 도메인에 포함시킨다.

색인에 저장되는 데이터는 (*tid*, *loc*, t_{enter}, t_{leave})로 표현되며 시간 간격 [t_{enter}, t_{leave}] 동안 객체 *tid*가 판독기의 위치 *loc*에 존재했음을 의미한다. *tid*가 판독기의 위치 *loc*에서 enter를 발생시키면 새로운 데이터가 색인에 삽입된다. 이때 인식 영역을 벗어나는 시간(t_{leave})을 알 수 없기 때문에 시간 간격에서 t_{leave}는 *now*로 나타낼 수 있으며, 현재까지 인식 영역 안에 존재함을 의미한다. 반면 객체 *tid*가 *loc*에서 leave 이벤트를 발생시킬 때는 기존 데이터에서 *now*의 값을 t_{leave}로 갱신하는 연산이 수행된다. 구현에서 *now*의 값은 시간 도메인에서 가장 큰 값을 사용한다.

now 값을 가지는 현재 위치 데이터는 갱신 연산이 수행될 때까지 시간이 지남에 따라 계속 성장한다(갱신 연산이 수행된 데이터는 더 이상 성장하지 않는다). 이 논문에서는 두 데이터를 구별하기 위해서 *now* 값을 포함하는 간격 데이터를 성장 간격(growing interval, GI)이라 하고 갱신 연산이 수행된 데이터를 고정 간격(fixed interval, FI)이라 정의한다.

5. 태그 객체를 위한 색인구조

5.1 색인 구조

본 논문에서 제안하는 색인 구조의 비단말 노드는 기존 R-Tree와 동일한 <*cp*, *rect*> 형태의 엔트리를 가진다. *cp*는 트리에서 자식 노드의 주소를 가리키고 *rect*는 자식 노드의 모든 엔트리를 포함하는 최소 경계 박스(Minimum Bounding Box, MBB)를 의미한다. 단말 노드는 <*I*> 형태의 엔트리를 가진다. *I*는 <*tid*, *loc*, t_{enter}, t_{leave}> 형태이기 때문에 *tid*를 단말 노드의 엔트리에는 따로 저장할 필요가 없다. *loc*는 2차원일 경우 (x, y)의 공간 좌표를 가지고, 3차원일 경우 (x, y, z)를 가진다.

색인을 변경하는 주요 연산은 enter 이벤트가 발생할 때 수행되는 삽입 연산이며 이것은 노드의 분할을 일으키고 색인의 크기를 증가시킨다. leave 이벤트가 발생할 경우, 기존 데이터를 수정하는 갱신 연산이 수행되며, 색인의 크기는 변경되지 않는다. 대신에 단말 노드의 엔트리에서 t_{leave} 값이 큰 값에서 작은 값으로 수정되기 때문에 필요한 경우 단말 노드에서 루트 노드까지 노드의 MBB를 조절한다

5.1 데이터 삽입

삽입 알고리즘은 새로운 데이터 GI가 들어갈 단말 노드를 찾는 ChooseSubtree와 삽입 후의 노드 MBB를 조절하고 분할을 상위 노드로 전파하는 AdjustTree 알고리즘으로 구성된다[1]. R-Tree에서 삽입은 최소 영역 확장을 가지는 엔트리를 선택하여 단말 노드를 찾아 내려 간다. 그러나 최소 영역 확장 정책은 간격 데이터에서 노드 겹침을 증가 시킬 수 있는데, 이것은 노드가 GI와 FI를 동시에 가질 수 있기 때문이다. 그림 1은 새로운 데이터 GI_{NEW}가 삽입될 때, R1보다는 R2에서 더 작은 영역 확장을 가질 경우 R1과 R2에서 노드 겹침이 발생함을 보여준다. 이것을 해결하기 위해서는 최소한의 노드 겹침을 일으키는 엔트리를 선택해야 한다. 최소한의 노드 겹침을 가지는 엔트리를 선택하는 알고리즘은 [2]에 있기 때문에 본 논문에서는 삽입 알고리즘을 따로 기술하지 않는다.

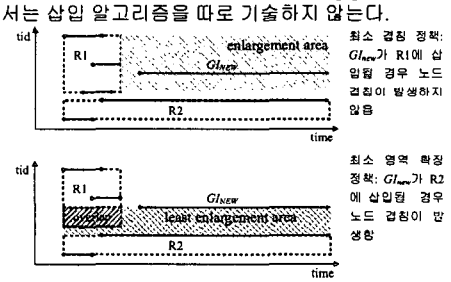

그림 1: 최소 영역 확장 정책으로 인한 노드 겹침

5.2 데이터 갱신 알고리즘

이전 데이터 GI를 FI로 바꾸는 갱신 알고리즘 UpdateData는 이전 GI를 찾는 FindLeaf와 필요 시에 노드 MBB를 조절하는 AdaptTree 알고리즘으로 구성된다.

Algorithm UpdateData(FI I_{NEW})

```
UD1.  GI I_OLD := GI(I_NEW.tid, I_NEW.loc, I_NEW.lt, now);
UD2.  Leaf l := FindLeaf(I_OLD);
UD3.  IF l = NULL
        return FALSE;
UD4.  ELSE
        find entry e in l that contains I_OLD;
        update e.I into I_NEW;
        Node p := ParentNode(l);
        IF(p != NULL) AdaptTree(p,l);
        return TRUE
```

Algorithm AdaptTree(Node *p*, Node *l*)

```
AT1.  find entry e in p that contains l.mbb;
AT2.  IF e.mbb != l.mbb
        e.mbb := l.mbb;
        adapt p.mbb so that it tightly encloses all entry
        rectangles in p;
        Node pp := ParentNode(p);
        IF(pp != NULL) AdaptTree(pp, p);
AT3.  ELSE stop;
```

알고리즘 1: UpdateData

5.3 단말 노드 분할 알고리즘

GI와 FI는 색인의 단말 노드에 동시에 존재할 수 있으므로 노드

의 영역을 크게 만드는 GI보다는 FI를 많이 가짐으로써 이웃 노드와의 겹침을 최소화해야 한다. 분할 시에 단말 노드에 있는 *tid* 수는 GI의 수와 같거나 작기 때문에 *tid* 수가 작으면 FI 수가 높아진다. 본 논문에서는 단말 노드에 존재하는 *tid* 수를 작게 만들기 위해서 임의의 *p*개의 *tid*를 가질 때까지 *tid* 축에서 분할하는 정책을 사용한다. 여기서 *p/N(N:* 노드의 최대 엔트리 수*)*를 *tid* 분할 인자(TSF)라 정의한다.

단말 노드에서 분할이 일어나는 과정은 다음과 같다. 단말 노드에서 오버플로우가 발생할 경우 노드에 존재하는 *tid* 수가 *p*보다 크면 *tid* 축에서 분할이 일어난다. 다음 분할 시에 *tid* 수가 *p*보다 크면 계속해서 *tid* 축에서 분할이 일어나고 그렇지 않을 경우에는 시공간 분할을 수행한다. 시공간 분할은 기존 R*-tree의 분할 정책과 동일하지만 *tid*값은 고려하지 않고 시간과 공간으로 형성된 MBB만 고려한다. 시공간 분할 후에 *tid* 축으로 분할 하지 않으면 반드시 시간 축 분할을 수행하며, 시간 축 분할은 GI와 FI를 서로 다른 노드에 저장한다.

알고리즘 2는 *N*+1개의 데이터를 두 그룹으로 분할하는 SplitLeafNode 알고리즘을 기술한다.

Algorithm SplitLeafNode(GI I_{NEW}, GROUP *g1*, GROUP *g2*)	
SLN1.	calculate *tidNum* of *N* entries in leaf *l* and I_{NEW};

SLN2.	IF *tidNum* > *p*
	invoke RFSplitTID(I_{NEW}, *g1*, *g2*);
	splitedAixs := TID;

SLN3.	ELSE IF *tidNum* <= *p* AND *splitedAixs* = TID OR *splitedAixs* = TIME
	invoke RFSplitSpatioTemporal(I_{NEW}, *g1*, *g2*);
	splitedAixs := SPATIOTEMPORAL;

SLN4.	ELSE IF *tidNum* <= *p* AND *splitedAixs* = SPATIOTEMPORAL
	invoke RFSplitTime(I_{NEW}, *g1*, *g2*);
	splitedAixs := TIME;

Algorithm RFSplitTID(GI I_{NEW}, GROUP *g1*, GROUP *g2*)	
STI1.	sort *N* +1 entries in increasing order of *tid* value;
STI2.	make *n* lists each list includes same *tid* value;
STI3.	insert first *n*/2 lists to *g1* and insert the remaining lists to *g2*;
Algorithm RFSplitSpatioTemporal(GI I_{NEW}, GROUP *g1*, GROUP *g2*)	
	/*RFSplitSpatioTemporal is similar to R*-tree's split algorithm except that tid domain is not considered*/
SST1.	invoke ChooseSplitAixs;
SST2.	invoke ChooseSplitIndex;
SST3.	distribute *N*+1 entries into g1 and g2;
Algorithm RFSplitTime(GI I_{NEW}, GROUP *g1*, GROUP *g2*)	
STE1.	IF each entry *e* in *N*+1 entries is GI insert *e* to g1;
STE2.	ELSE insert *e* to g2;

알고리즘 2: SplitLeafNode

6. 성능평가

본 논문에서 제안한 단말 노드 분할 정책이 다른 기존의 분할 정책보다 우수함을 보여주기 위해서 R-tree의 Quadratic Split과 R*-tree의 재삽입 및 분할 정책과 비교한다. 실험에 사용된 모든 색인은 간격 데이터를 저장한다. 실험 데이터는 GSTD 알고리즘[5]과 유사한 태그 객체의 간격 데이터를 생성하는 프로그램을 개발하여 사용하였다. 그리고 실험에 사용된 데이터 수는 1000개 객체가 각각 50,000번의 enter와 leave를 발생하는 100,000개의 데이터를 사용하였다.

6.1 삽입 성능

그림 2는 3가지 종류의 색인에서 삽입 성능의 차이를 보여준다. 그림에서 RF_ 다음의 접미사는 *tid* 분할 인자(TSF)를 나타낸다. 본 논문에서 제안하는 색인이 TSF 0.5에서 기존 R-tree와 R*-tree보다 각 17%와 39%의 성능 향상이 있었다. 삽입 성능에서 R*-tree가 낮은 이유는 노드에서 오버플로우가 발생하면 재삽입으로 인해 디스크 접근 수가 증가하기 때문이다.

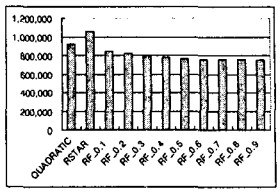

그림 2: 100,000개의 데이터에서 삽입 성능 비교.

6.2 질의 성능

그림 3은 find 질의의 디스크 접근 횟수를 보여준다. TSF가 0.5 일 때 이 논문에서 제안한 색인이 기존 R-tree와 R*-tree보다 평균 80%와 19%의 성능 향상을 보였다. 그림 4는 look 질의의 디스크 접근 횟수를 보여주는데, TSF가 0.5 일 때 가장 좋은 성능을 보였으며 R-tree보다 평균 65%, R*-tree보다 평균 34%의 성능 향상을 보였다.

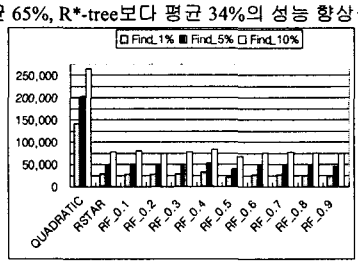

그림 3: 100,000개 데이터에서 find 질의 성능 비교

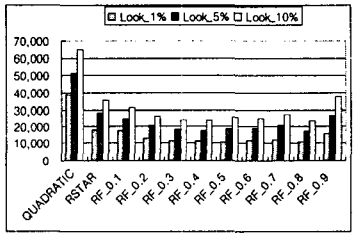

그림 4: 100,000개 데이터에서 look 질의 성능 비교

7. 결론 및 향후 연구

이 논문에서는 태그 객체의 다차원 대용량 데이터에서 질의 처리를 위한 데이터 모델과 색인 구조를 제안하였다. 태그 객체는 간격 데이터로 표현 했으며 태그 *tid*를 새로운 색인 도메인으로 시공간 도메인에 포함시켰다. 그리고 이 논문에서 제안한 새로운 분할 정책이 기존 분할정책보다 더 우수함을 실험을 통해서 증명하였다.

향후 연구로서 노드 겹침을 일으키는 긴 FI 데이터를 효율적으로 저장하는 색인 구조에 대한 연구가 필요하다.

참고문헌

[1] A. Guttman, "R-trees: A dynamic index structre for spatial searching", ACM SIGMOD Conference, pp.47-54, 1984.

[2] N. Beckmann and H. P. Kriegel, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", In Proc. ACM SIGMOD, pp.332-331, 1990

[3] Y. Theodoridis, M. Vazirgiannis and T. Sellis, "Spatio-Temporal Indexing for Large Multimedia Applications", In International Conf, on Multimedia Computing and System, pp.441-448, 1996

[4] K. Romer, T. Schoch, F. Mattern and T. Dubendorfer, "Smart identification frameworks for ubiquitous computing applications" Pervasive Computing and Communications. Proceedings of the First IEEE International Conference o, p256-262, 2003.

[5] Y. Theodoridis, J. R. Silva and M. A. Nascimento, "On the Generation of Spatiotemporal Datasets", SSD, Hong Kong, LNCS 1651 Springer, pp.147-164, 1999.