RFID 스트리밍 데이타의 연속질의를 위한 영역 스태빙 기법

(Range Stabbing Technique for Continuous Queries on RFID Streaming Data)

박 재 관 † 홍 봉 희 †† 이 기 한 †††
(Jaekwan Park) (Bonghee Hong) (Kihan Lee)

요 약 RFID 표준 개발을 주도하고 있는 EPCglobal은 RFID 미들웨어에 관한 표준 인터페이스로써 ECSpec(Event Cycle Specification)과 ECReports(Event Cycle Reports)를 제시하였다. ECSpec은 애플리케이션이 원하는 태그 데이타에 대한 명세로써 일정 시간 동안 태그 이벤트에 대한 여과 및 수집을 반복적으로 처리하기 위한 연속질의(Continuous Query, CQ)이며 ECReports는 처리 결과에 대한 명세이다. 따라서 연속 질의 수행에 적합하도록 설계된 질의 색인(Query Index) 기법을 적용하여 ECSpec을 질의 색인의 데이타로써, 태그 이벤트를 질의 색인의 질의로써 수행하면 효율적이다. 하지만 RFID 물류환경에서는 유사한 또는 동일 상품군이 대량으로 이동하게 되고, 이때 상품에 부착된 태그가 RFID 리더에서 인식될 때 발생하는 이벤트가 짧은 기간 동안 다수 발생하게 되는데, 이를 개별적으로 처리하는 것은 비효율적이다.

이 논문에서는 ECSpec에서 지정하는 보고주기 기간 동안 태그 이벤트에 의한 질의를 수집하여 그룹을 구성하고 질의 색인에 영역 질의로써 수행하여 유사 검색 과정을 제거하는 기법을 제안한다. 이러한 질의 그룹 처리 기법을 위해, 일정기간 연속된 태그 이벤트를 효과적으로 수집하기 위한 큐의 구성 방법과 수집된 태그 이벤트로부터 영역 질의를 생성하기 위한 방법을 제안한다. 또한 실험을 통해 이러한 기법의 효율성을 확인한다.

키워드 : 미들웨어, 연속 질의, 전자태그, 스트리밍 데이타

Abstract The EPCglobal leading the development in RFID standards proposed Event Cycle Specification (ECSpec) and Event Cycle Reports (ECReports) for the standard about RFID middleware interface. ECSpec is a specification for filtering and collecting RFID tag data and is treated as a Continuous Query (CQ) processed during fixed time intervals repeatedly. ECReport is a specification for describing the results after ECSpec is processed. Thus, it is efficient to apply Query Indexing technique designed for the continuous query processing. This query index processes ECSpecs as data and tag events as queries for efficiency. In logistics environment, the similar or same products are transferred together. Also, when RFID tags attached to the products are acquired, the acquisition events occur massively for the short period. For these properties, it is inefficient to process the massive events one by one.

In this paper, we propose a technique reducing similar search process by considering tag events which are collected by the report period in ECSpec, as a range query. For this group processing, we suggest a queuing method for collecting tag events efficiently and a structure for generating range queries in the queues. The experiments show that performance is enhanced by the proposed methods.

Key words : Middleware, Continuous Query, RFID, Streaming Data

· 이 논문은 2008년 교육과학기술부로부터 지원받아 수행된 연구임(지역거점연구단육성사업/차세대물류IT기술연구사업단)

† 정 회 원 : LG전자기술원 선임연구원
jkpark183@gmail.com
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
vvbhhong@gmail.com
††† 정 회 원 : 삼성전자 연구원
kihan0813.yi@samsung.net

논문접수 : 2008년 10월 21일
심사완료 : 2009년 1월 6일

정보과학회논문지: 데이타베이스 제36권 제2호(2009.4)

1. 서 론

객체 자동 식별 및 제품의 이력 추적을 위한 기술로써 RFID시스템[1]이 각광 받고 있다. RFID 시스템은 개별 제품의 식별을 위해 부착되는 태그, 태그의 정보를 인식하는 판독기 그리고 태그 이벤트 데이타 및 응용 시스템의 질의를 처리하는 미들웨어로 구성된다. 전세계의 물류 환경을 통합하기 위한 표준을 주도하고 있는 EPCglobal은 RFID를 이용한 글로벌 네트워크 구축을 위해 미들웨어의 표준 인터페이스인 ALE(Application Level Event) 명세를 제안하였다. ALE 명세는 애플리케이션이 미들웨어로부터 태그 정보를 얻기 위한 표준 인터페이스인 ECSpec(Event Cycle Specification)과 ECReports(Event Cycle Reports)를 명시하고 있다. ECSpec은 애플리케이션이 원하는 태그 데이타에 대한 명세로써 일정 시간 동안 판독기에서 인식되는 태그 이벤트에 대해 여과 및 수집을 반복적으로 처리하기 위한 연속질의(Continuous Query, CQ)[2]이며 ECReports는 ECSpec의 처리 결과에 대한 명세이다[3].

스트리밍 환경에서 기존의 데이타베이스 관리 시스템에서 사용하는 데이타 색인 기법을 사용하면 계속적인 색인 갱신이 필요하고 색인이 갱신될 때마다 주어진 모든 연속질의를 재수행 해야 하기 때문에 매우 비효율적이다. 따라서 이러한 스트리밍 환경에서는 연속질의를 데이타로써 저장하고 색인을 구축하여 끊임없이 발생하는 스트리밍 데이타를 질의로써 처리하는 질의 색인 기법을 사용하면 효율적이다[4]. 그러므로 RFID 미들웨어에서는 태그가 판독기에 의해 인식될 때 발생하는 스트리밍 데이타, 즉 태그 이벤트와 연속질의인 ECSpec에 대해 질의 색인 기법을 적용하여 ECSpec을 질의색인의 데이타로써, 태그 이벤트를 질의 색인의 질의로써 수행하면 효율적이다[5,6].

RFID 물류환경에서 태그를 부착한 제품들은 개별적으로 이동하는 것이 아니라 유사 제품끼리 그룹을 지어 이동하는 특징이 있다. 따라서 기존의 스트리밍 데이타와는 달리 태그 이벤트는 짧은 기간 동안 대량으로 발생하며 그 이벤트들은 유사한 식별자를 가진다. 기존의 RFID 질의 색인에서는 태그 이벤트를 질의로써 처리하지만 태그 이벤트의 이러한 특징을 고려하지 않는다. 즉, 질의의 결과가 동일하거나 유사한 다수의 질의를 개별적으로 처리함으로써 색인에 대해서 동일 또는 유사 검색과정을 반복적으로 수행하는 문제점이 있다. 기존 연구에서는 이러한 태그 이벤트들이 발생하는 특징을 고려하지 않기 때문에 유사한 태그 이벤트들이 다수 발생하는 물류 환경에서는 효율적으로 동작하지 않는다.

이러한 문제를 해결하기 위해 본 논문에서는 ECSpec에서 지정하는 보고주기 기간 동안 태그 이벤트에 의한 질의를 수집하여 그룹으로 구성하고 질의 색인에 영역 질의로써 수행한다. 이러한 질의 그룹 처리 기법은 유사 검색 과정을 제거하므로 RFID 미들웨어의 성능을 향상시킨다. 이를 위해 본 논문에서는 태그 이벤트를 효과적으로 수집하기 위한 큐의 구성 방안을 제안하고 수집된 태그 이벤트로부터 질의 그룹으로 구성하기 위한 자료구조를 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 기술하고, 3장에서는 대상환경과 문제정의를 기술한다. 4장에서는 연속된 태그 이벤트 질의에 대한 효율적인 처리 기법을 제안한다. 5장에서는 제안한 기법에 대한 성능 평가 및 분석 결과를 기술한다. 마지막으로, 6장에서는 결론 및 향후 연구를 기술한다.

2. 관련 연구

이 장에서는 본 논문과 관련하여 스트리밍 데이타에 대해서 연속질의를 효율적으로 처리하기 위해 제안된 기존의 질의색인 연구에 대해 기술한다. 질의 색인에 대한 연구는 크게 이동체 환경, 센서 네트워크 환경 그리고 RFID 환경으로 나눌 수 있다.

이동체 데이타베이스 환경에서는 이동체의 위치 스트리밍 데이타에 대한 연속 질의를 처리하기 위한 질의 색인 기법이 연구되었다. CQI[4] 색인 기법은 영역 질의를 그리드 셀로 분할하고 분할에 사용된 셀의 리스트 구조에 질의의 식별자를 저장하는 방법이다. 즉, 이 리스트는 이동체가 위치를 보고할 때 해당 질의를 찾기 위한 대상이 된다. VCR[7] 색인 기법은 영역 질의를 가변 크기의 가상 구조(Virtual Constructs, VCs)를 이용하여 나누고 질의의 식별자는 각 가상 구조의 저장고간에 삽입된다. 따라서, 탐색은 이동체가 보고한 위치를 포함하는 가상 구조를 찾는 과정으로 구성된다.

센서 네트워크 환경에서 대표적인 질의색인으로는 NiagaraCQ[8]나 TelegraphCQ[9]와 같은 센서 데이타 스트림 시스템에서 채택하고 있는 IBS-tree[10]를 들 수 있다. IBS-tree는 1차원 균형 이진 트리이며, 질의 속성의 개수만큼 트리가 구축된다. 이 색인은 연속 질의에 명시되는 속성의 간격을 저장하고 센서 노드에서 감지 이벤트가 발생할 때 감지된 값을 만족하는 간격 데이타를 이진 탐색하여 결과셋을 도출한다.

RFID 환경을 고려한 대표적인 색인으로는 TLC-Index[11]를 들 수 있다. TLC-Index는 연속질의인 ECSpec을 2차원 공간 상에 2D Interval로써 표현하여 삽입 및 점 검색을 효율적으로 지원하기 위한 색인 구조이다. ECSpec의 Predicate을 2D Interval로써 표현되며 매우 긴 길이를 가지는 Long Interval이 된다. TLC-Index는

이러한 문제점을 해결하기 위하여 그리드 방식의 큰 크기를 가지는 셀 분할 구조와 선분 모양의 가상 분할 구조를 병행하여 사용한다. TLC-Index는 셀 분할 구조와 가상 분할 구조의 장점을 살려 2D Interval을 다양한 크기의 레벨을 가지는 셀 분할 구조로 분할 삽입함으로써 저장 공간의 소모를 줄이고 삽입 성능 및 검색 성능을 향상시킨다.

3. 대상환경 및 문제정의

3.1 대상환경

RFID 응용 물류 환경에서는 태그를 부착한 제품이 생산지에서 소비지로 이동하며 각 지점에서 설치된 판독기를 통해 제품에 부착된 태그가 인식된다. 이렇게 인식된 태그 이벤트 정보들은 RFID 미들웨어로 전송되고 수집된다. 애플리케이션은 미들웨어로부터 원하는 정보를 획득하기 위하여 ECSpec을 등록하고 그 결과를 지속적으로 받는다. 이때, ECSpec에 포함된 논리적 리더명과 태그의 패턴에 대한 여과정보는 판독기 식별자 도메인(Reader Identification Domain, RID)과 태그 식별자 도메인(Tag Identification Domain, TID)으로 이루어진 2차원 색인 공간 상에 질의 데이타로써 저장된다. 특정 판독기에서 읽힌 태그 이벤트의 정보는 2차원 공간(RID, TID)에서 점으로 표현된다. 예를 들어 판독기(ridi)가 태그(tidj)를 인식했다면 태그 이벤트 정보는 (ridi, tidj)가 된다. 이렇게 표현된 태그 이벤트 정보는 질의 색인에 Stabbing 질의로써 수행된다[5,6].

본 논문에서는 질의 색인을 위하여 질의 데이타와 stabbing 질의를 다음과 같이 정의한다.

정의 1. 질의 데이타 : 질의 색인을 구성하는 데이타로써 ECSpec의 논리적 리더명과 태그 패턴에 대한 여과정보를 변환하여 2차원 공간(RID, TID)에 표현한 것

정의 2. Stabbing 질의 : 판독기에서 감지된 태그 이벤트(ridi, tidj)를 질의 색인에서 점질의로써 수행하는 것

그림 1은 질의가 등록되는 과정과 태그 이벤트에 의한 stabbing 질의의 수행 과정을 보여준다. 먼저, 애플리케이션이 등록한 ECSpec이 변환되어 질의 색인에 질의 데이타 Q5로서 등록된다. 다음으로 판독기에서 발생하는 태그 이벤트는 stabbing 질의 (2, 4)로써 질의 색인에 수행된다. 질의 수행에 의해 질의 데이타 Q5가 탐색되고 그 결과로 태그 이벤트 정보가 Q5를 등록한 애플리케이션에게 전달된다.

3.2 문제정의

RFID 태그 이벤트는 센서 네트워크의 데이타와 유사하게 스트리밍 데이타 특성을 가진다. 즉, 곳곳에 배치된 판독기로부터 끊임없이 연속적으로 태그 이벤트가 발생된다. 하지만, RFID 태그 이벤트는 태그를 부착한 제품의 이동 특성에 의해 센서 데이타와 다른 차이점이 있다. 태그를 부착한 제품은 개별적으로 움직이는 것이 아니라 비슷한 제품 군을 이루어 생산지로부터 소비지로까지 대량으로 이동되는 특성을 가진다. 따라서 이러한 특성에 따라 판독기의 인식에 의해 발생하는 태그 이벤트는 대량으로 발생하며, 질의 색인으로의 stabbing 질의 역시 짧은 시간 동안 매우 많이 발생하게 된다. 이러한 태그 이벤트들의 태그 식별자들은 태그 데이타 표준 명세

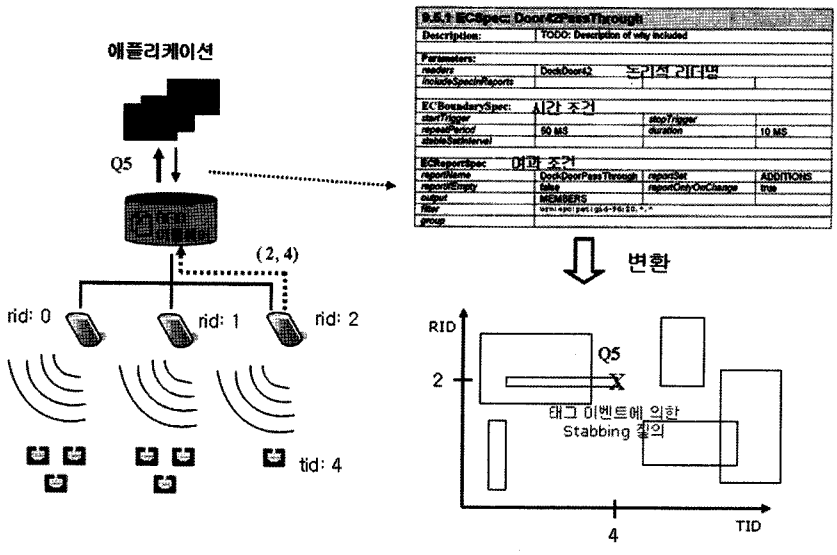

그림 1 RFID 미들웨어에서 질의 색인 수행 과정 예제

(Tag Data Standard, TDS)[12]에 따르면 근소한 차이를 가진다. 따라서, RFID 리더에서 태그를 인식할 때 발생하는 stabbing 질의 역시 유사한 값들을 가지게 된다.

기존의 RFID 미들웨어에서의 질의처리는 stabbing 질의를 개별적으로 처리하였다. 즉, 발생하는 모든 태그 이벤트에 대해서 개별적으로 stabbing 질의로 처리하고 그 결과를 반환하는 과정을 반복적으로 수행하였다. 따라서 제품의 이동 특성에 따른 RFID 스트리밍 데이타를 개별 질의로 처리할 경우 질의 색인에서 동일 또는 유사 검색과정을 반복적으로 수행하는 문제점을 수반한다.

그림 2는 개별 stabbing 질의로 처리 시 동일 검색 과정을 반복적으로 수행하는 것을 보여준다. 즉, 각 질의는 최소 경계 사각형(Minimum Bounding Rectangle, MBR) R1의 데이타 Q2, Q3, Q4를 계속적으로 탐색하는 동일 검색 과정을 반복하게 된다. 이러한 단점을 보완하기 위해서, 본 논문에서는 태그 이벤트의 대량 발생 및 유사성을 고려하여 연속된 stabbing 질의를 개별적으로 처리하지 않고 질의그룹을 구성하여 영역 질의로 수행함으로써 유사 질의에 의한 반복적 탐색을 방지하면서 질의 수행 횟수를 감소시키기 위한 기법을 제안한다.

4. 연속된 stabbing 질의의 효율적 처리 기법

4.1 기본 아이디어

RFID 연속 질의는 보고주기 기간을 가진다. 즉, stabbing 질의 수행 결과를 즉시 보고하지 않고 보고주기 기간 동안 수집한다. 그리고 보고주기 기간이 만료될 때 수집된 결과 집합을 애플리케이션에게 전달한다. 이러한 질의의 특성은 일정 기간 동안 질의 처리를 지연하는 것을 허용한다. 본 논문에서는 이러한 특성을 이용하여 대량으로 발생하는 stabbing 질의를 즉시 수행하지 않고 보고주기 기간 동안 수집 후 영역질의로 처리하는 방법을 제안한다. 그림 3처럼 다수의 개별 stabbing 질의를 영역 질의로 구성해 처리함으로써 질의 수행 횟수를 줄인다.

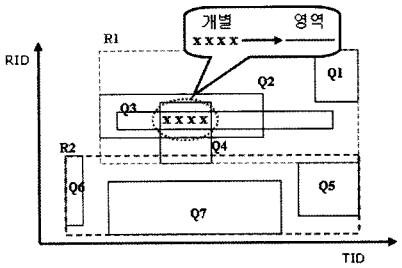

그림 3 개별 stabbing 질의를 영역 질의로 처리

본 논문에서는 RFID 태그 이벤트에 의한 stabbing 질의의 질의 그룹을 *sequence*라고 정의한다.

정의 3. Sequence S : 각 논리적 리더 식별자(Reader IDentification, RID)에 대해서 주어진 특정 시간 동안 연속된(connected) 태그 식별자(Tag IDentification, TID) 값들의 집합(Set)으로써 $\{rid_i, (tid_1, \cdots, tid_{n-1}, tid_n)\}$ (단, $tid_n - tid_{n-1} = 1$)로 표현된다. 예를 들어 "Gate A"(RID = 1)에서 "0.0.1" ~ "0.0.5"의 연속되는 태그 데이타가 일정기간 동안 수집되었다면 질의 그룹인 *sequence*는 {1, (1, 2, ···, 4, 5)}가 된다.

Stabbing 질의는 TID축에서 유사한 값을 가지지만 발생하는 순서는 비정렬 및 불연속적이다. 이러한 stabbing 질의들로부터 sequence를 구성하기 위해서는 먼저 stabbing 질의들은 sequence로 구성될 수 있는 동일 RID 별로 분류 및 수집을 해야 한다. 동일 RID를 가진 stabbing 질의들은 동일한 보고 주기를 따르기 때문이다. 다음으로, 수집된 질의 집합으로부터 sequence를 구성하는 과정이 필요하다. 따라서 본 논문에서는 stabbing 질의의 분류 및 수집을 위한 큐의 구성방안과 분류된 질의에서 sequence를 효율적으로 구성하기 위한 자료구조를 제안한다.

4.2 Stabbing 질의 분류를 위한 큐의 구성

판독기(Reader, 리더)는 물리적 리더와 논리적 리더로 구별되며, 정의는 다음과 같다.

EPC Code	Reader	Timestamp
10.10.4	Gate A	12:00:00
10.10.2	Gate A	12:00:00
10.10.5	Gate A	12:00:01
10.10.1	Gate A	12:00:02
...	...	...

다수 태그 이벤트를
개별 질의로 반복 수행

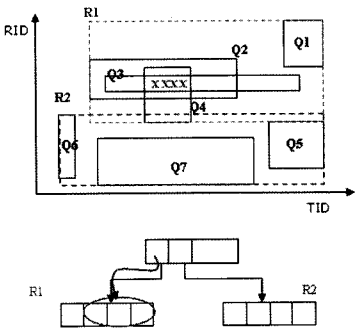

그림 2 동일 또는 유사 검색 과정을 반복 수행하는 문제

정의 4. 물리적 리더 : 실제 물리적 장치인 판독기를 일컫는 것으로 태그 이벤트의 원천이다.

정의 5. 논리적 리더 : 물리적 리더를 논리적 관점에서 분류한 것으로 하나 이상의 물리적 리더로 구성된다. 각 논리적 리더는 식별자 $\{rid_i \mid rid_i \in N\}$를 가진다.

질의 데이타인 ECSpec은 물리적 리더가 아닌, 논리적 리더를 명시한다[3]. 하지만 stabbing 질의로 수행되는 태그 이벤트는 물리적 리더 단위로 발생한다. 따라서 물리적 리더 별로 발생하는 stabbing 질의를 논리적 리더 단위로의 맵핑 과정이 필요하다. 이를 위해 본 논문에서는 맵핑 과정을 수행하기 위해 현재 구성된 *논리적 리더 별로 큐(Logical Reader Queue, LRQ)*를 두는 방법을 제안한다. 그림 4와 같이 논리적 리더 구성에 따라 물리적 리더를 직접 큐에 연결하게 되면 분류를 위한 추가 비용 없이 stabbing 질의의 분류가 가능하다. 이렇게 분류된 stabbing 질의들은 sequence 구성을 위한 자료구조로 전달된다.

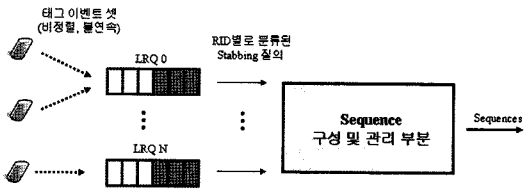

그림 4 Stabbing 질의 분류를 위한 논리적 리더 큐

4.3 Sequence 추출 및 관리 구조 설계

이 장에서는 논리적 리더 별로 분류된 stabbing 질의 집합에서 sequence를 구성하기 위한 *SequenceSet* 구조를 제안한다.

4.3.1 Sequence 추출을 위한 자료 구조

SequenceSet은 sequence를 효율적으로 구성하기 위한 자료 구조이다. 자료구조인 Set 개념을 확장한 것으로 비정렬, 불연속 stabbing 질의 집합에서 sequence를 빠르게 구성하기 위한 구조를 가진다.

4.3.1.1 SequenceSet의 구조

SequenceSet은 stabbing 질의 집합에서 구성된 sequence의 정보를 표현하는 Sequence 헤더(Sequence Header) 구조와 각각의 sequence에 대한 구성요소인 stabbing 질의에 대한 리스트인 원본 데이타 리스트(Original Data List, ODL)로 구성된다. 그림 5는 SequenceSet에 대한 구조를 보여주고 있다.

Sequence Header는 하나의 sequence를 표현하는 Sequence 경계 표현(Sequence Boundary Representation, SBR) 항목의 리스트 구조이다. SBR은 Sequence의 하한 값(lower value)과 상한 값(upper value) 그리고 원본 데이타 리스트에 대한 링크로 구성된다. Sequence Header는 두 가지 성질을 만족한다. 첫째, SBR_{n-1}과 SBR_n은 2 이상의 차이를 가진다. 둘째, SBR들은 하한 값을 기준으로 오름차순으로 정렬된다.

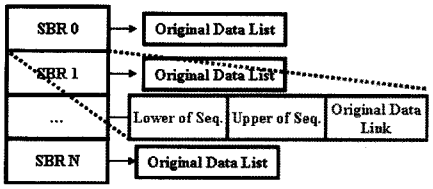

그림 5 SequenceSet의 구조

원본 데이타 리스트는 삽입된 stabbing 질의 정보 중 sequence를 구성하는데 필요한 태그 식별자 이외에 논리적 리더 식별자 및 실제 판독기에서 감지된 시간 값에 대한 정보를 유지하기 위해 필요하다.

그림 6은 5개의 sequence가 구성되어 있는 SequenceSet의 예제를 보여준다.

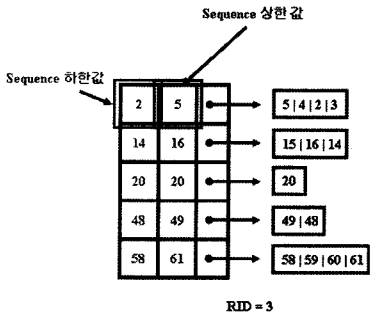

그림 6 SequenceSet 예제

4.3.1.2 단순 삽입 방법(Simple Insertion)

SequenceSet에 새로운 stabbing 질의의 삽입은 세 단계의 과정을 거쳐 이루어진다. 먼저 삽입된 stabbing 질의가 들어갈 위치를 찾는데 이것은 sequence들이 오름차순으로 정렬을 유지하기 때문에 이진 검색으로 삽입 대상 sequence를 선택한다. 다음으로 선택된 sequence에 대한 조정단계가 수행된다. 그림 7에서 보이는 것처럼 삽입되는 stabbing 질의가 특정 sequence의 하한 값이나 상한 값과 1의 차이가 있을 때는 해당 sequence의 확장(**Extend**)이 된다. 또한 그림 8과 같이 특정 시퀀스에 포함되지 않으면서 가장 값이 유사한 sequence의 하한 값이나 상한 값과 2이상의 차이가 있을 경우에는 새로운 sequence가 생성(**Generate**)된다. 마지막으로 그림 9와 같이 삽입된 stabbing 질의가 두 sequence를 이어줄 경우 하나의 sequence로 병합(**Union**)된다. 조정 단계가 끝난 후 해당하는 sequence

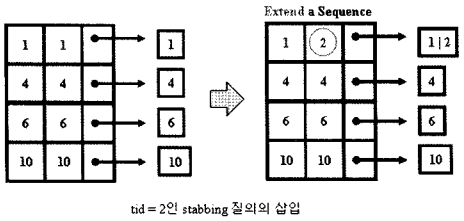

그림 7 삽입에 의해 sequence가 확장되는 경우

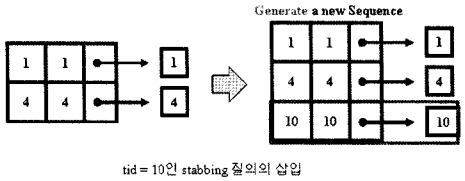

그림 8 삽입에 의해 sequence가 생성되는 경우

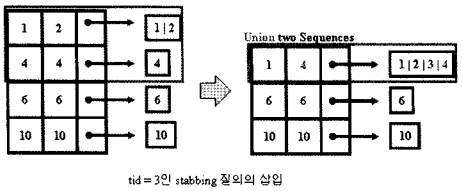

그림 9 삽입에 의해 sequence가 병합되는 경우

에 대한 원본 데이타 리스트의 마지막에 stabbing 질의가 삽입된다.

알고리즘 1은 SequenceSet에 대한 단순 삽입 알고리즘이다. 알고리즘 2, 3, 4는 알고리즘 1에서 각각 생성(Generate), 확장(Extend) 및 병합(Union)을 처리하기 위한 알고리즘이다.

Algorithm SimpleInsertion(Integer *epc*)

SBR $entry_i$ = find largest i_{th} entry if $entry_i$.lower < *epc*
IF $entry_i$.upper ≥ *epc*
    THEN $entry_i$.**add**(*epc*)
ELSE
    IF *epc* - $entry_i$.upper > 1 and $entry_{i+1}$.lower – *epc* > 1
        THEN SBR newEntry = SequenceHeader.**Generate**(*epc*, i)
            newEntry.**insert**(*epc*)
    ELSEIF *epc* – $entry_i$.upper ≤ 1 and $entry_{i+1}$.lower – *epc* > 1
        THEN SequenceHeader.**Extend**(*epc*, i , **UPPER**)
            $entry_i$.**insert**(*epc*)
    ELSEIF *epc* – $entry_i$.upper > 1 and $entry_{i+1}$.lower – *epc* ≤ 1
        THEN SequenceHeader.**Extend**(*epc* , i+1, **LOWER**)
            $entry_{i+1}$.**insert**(*epc*)
    ELSE
        THEN SBR mergedEntry = SequenceHeader.**Union**(i, i+1)
            mergedEntry.**insert**(*epc*)
    ENDIF
ENDIF

알고리즘 1 단순삽입(SimpleInsertion)

Algorithm Generate(Integer *epc*, Integer *i*)

Make new SBR entry e
Assign epc to both e.lower and e.upper
Insert e into next position of i_{th} entry in SequenceHeader
Return entry e

알고리즘 2 생성(Generate)

Algorithm Extend(Integer *epc*, Integer *i*, Boundary *b*)

SBR s = getEntry(*i*)
IF *b* = **UPPER**
    THEN assign *epc* to s.upper
ELSEIF *b* = **LOWER**
    THEN assign *epc* to s.lower
Update s into SequenceHeader

알고리즘 3 확장(Extend)

Algorithm Union(Integer *i*, Integer *i+1*)

SBR mergerEntry = getEntry(*i*)
SBR mergeeEntry = getEntry(*i+1*)
Assign mergeeEntry.upper to mergerEntery.upper
Add all original data of mergeeEntry to mergerEntry
Delete mergeeEntery from SequenceHeader
Update mergerEntry into SequenceHeader

알고리즘 4 병합(Union)

4.3.1.3 개선된 삽입 방법(Advanced Insertion)

그림 10은 비연속적인 값을 가지는 stabbing 질의 집합이 삽입된 경우이다. 단순 삽입 방법을 사용할 경우, 만약 모든 삽입되는 stabbing 질의간 간격이 2인 경우 삽입된 stabbing 질의 개수만큼의 sequence가 생성된다. 이 경우, sequence에 의해 개선되는 효과에 비해

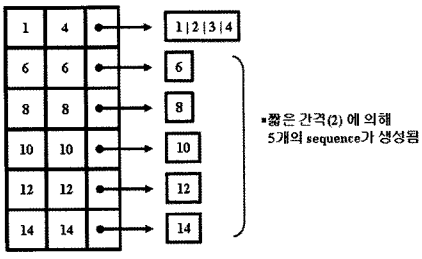

그림 10 다수의 sequence가 생기는 경우

sequence 생성에 필요한 추가 비용이 커지는 한계가 있다. 따라서 이러한 경우를 효과적으로 처리하기 위한 방안이 필요하다. 이 논문에서는 stabbing 질의들이 비연속적인 경우에 다수의 sequence가 생성되는 것을 방지하기 위해 개선된 삽입 방법을 제안한다.

개선된 삽입 방법은 sequence 구성 시 최대 허용 간격 개념을 도입하여 틈을 가진 *Pseudo-Sequence*를 생성한다.

정의 6. 최대 허용 간격(Maximum Gap, MaxGap) : sequence를 구성하는 stabbing 질의 간에 가질 수 있는 최대 간격

정의 7. Pseudo-Sequence : 최대 허용 간격 이하의 틈을 가진 sequence로써 $\{rid_i, (tid_j, \cdots, tid_{n-1}, tid_n)\}$ (단, $tid_n - tid_{n-1} \leq MaxGap$)로 표현된다.

최대 허용 간격 개념을 도입 시 삽입 알고리즘에서 확장(Extend)과 병합(Union)에 대한 수행 조건은 그림 11과 같이 변경된다.

| Insert Data - Upper(S_i) | ≤ MaxGap and | Insert Data – Lower(S_{i+1}) | > MaxGap
or
| Insert Data - Upper(S_i) | > MaxGap and | Insert Data – Lower(S_{i+1}) | ≤ MaxGap

a) Advanced Insertion을 위한 확장조건

| Insert Data - Upper(S_i) | ≤ MaxGap and | Insert Data - Lower(S_{i+1}) | ≤ MaxGap

b) Advanced Insertion을 위한 병합조건

그림 11 Advanced Insertion에서의 Extend와 Union의 수행 조건

그림 12와 그림 13은 개선된 삽입 방법에서 확장과 병합에 대한 예제로써, 최대 허용 간격에 의해 생성되는 sequence의 수가 적어지는 것을 알 수 있다.

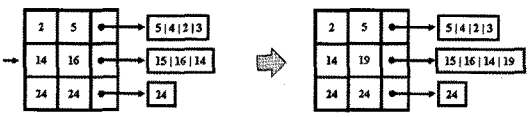

그림 12 MaxGap = 3, tid = 19인 stabbing 질의의 삽입

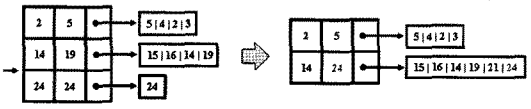

그림 13 MaxGap = 3, tid = 21인 stabbing 질의의 삽입

개선된 삽입 방법은 최대 허용 간격의 개념을 도입함으로써 Pseudo-Sequence를 생성하기 때문에 단순 삽입 방법보다 적은 수의 sequence를 생성하여 질의 수행 횟수를 감소시킨다. 하지만 질의 처리 시 sequence의 하한 값과 상한 값으로 이루어진 영역 질의를 수행하면 Pseudo-Sequence가 가진 틈에 의해 불필요한 질의 결과가 생긴다. 즉, 최대 허용 간격의 크기가 클수록 생성되는 sequence의 개수가 감소하지만 그에 따른 불필요한 질의 결과의 수가 증가한다. 따라서 이러한 절충 조건(trade-off)때문에 최대 허용 간격을 위한 적절한 크기 설정이 필요하다. 이를 위해 다양한 실험을 수행하여 최대 허용 간격 값을 결정한다. 표 1은 단순 삽입 방법(Simple Insertion)과 개선된 삽입 방법(Advanced Insertion)에 대한 비교 분석을 보여준다.

4.3.1.4 Sequence 추출 방법(Sequence Retrieving)

Sequence 추출을 위한 알고리즘은 Sequence Header에 구성된 sequence 정보들을 순차적으로 읽어가며 질의 색인에 대한 영역 질의로써 추출하는 과정으로 구성된다. 그림 14는 두 개의 SequenceSet에서 sequence를 추출하는 과정을 보여주는데 sequence는 하한 값과 상한 값으로 표현되는 영역 질의로써 질의 색인에서 수행된다.

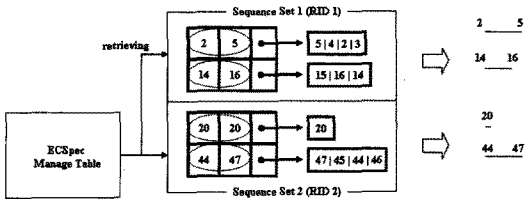

그림 14 SequenceSet의 추출 과정에 대한 예제

5. 성능 평가

이 장에서는 본 논문에서 제안하는 sequence에 의한 영역질의 성능을 평가하기 위하여 기존의 개별 stabbing 질의 방법과 비교한다. 성능 평가를 위해 질의 색인으로는 대표적인 다차원 색인인 R-tree[13]를 사용하였으며, 노드 용량(Node Capacity)을 1024로 주었다. 메인 메모리 환경에서 트리 계열 색인의 비용은 노드의 접근횟수와 노드를 처리하는 비용으로써 구할 수 있다[14]. 따라서 질의 처리 성능을 비교하기 위해 이 두가지 요소를 사용하였다. 실험의 컴퓨팅 환경은 Pentium

표 1 단순 삽입 방법과 개선된 삽입 방법의 비교 분석

삽입 방법	Simple Insertion	Advanced Insertion
생성 질의	√ Sequence	√ Pseudo-Sequence
장점	√ 정제 단계가 필요 없음	√ 생성되는 sequence의 수가 적음(최대 허용 간격 의존)
단점	√ 생성되는 sequence의 수가 많음	√ 정제 단계가 필요

IV 2.66Ghz 개인용컴퓨터, 1GB RAM, Windows XP의 운영체제에서 구현되었으며 Java언어를 사용하여 플랫폼에 독립적으로 동작할 수 있도록 하였다.

5.1 실험 환경

그림 15는 각각 질의 데이타와 Stabbing 질의 데이타에 대한 예제이다. 질의 데이타는 논리적 리더 식별자(RID), 여과 조건인 태그 패턴 및 시간 조건인 보고 주기 값으로 구성된다. 여기서 태그 패턴은 회사, 제품 및 아이템으로 구성되며 각각은 상수, 영역, 전체(*)를 값으로 가질 수 있다. Stabbing 질의 데이타는 논리적 리더 식별자와 태그 패턴 데이타로 구성된다.

9887-9979,11.11-18.619,91	8111,3.19.800
5669-5717,9.11.668-913,48	8588,8.13.695
836-907,14.27.*,19	97,6.12.641
2448-2540,11.19-19.*,11	5540,1.21.663
9642-9722,0.16.*,77	4502,13.19.396
1564-1625,15.3.858,84	5473,18.24.82
7546-7553,2.19.864-993,32	2503,13.4.318
7840-7866,9.19.*,54	6112,8.2.381
9847-9882,14.23-31.712,17	7231,9.16.759
9817-9841,10.*.1004,30	5675,4.2.993
4904-4937,1.16.754-839,80	9480,9.13.840
9489-9459,15.27-28.*,89	2772,11.21.187
6684-6688,7.13-26.566-902,63	6612,12.8.132
9517-9615,1.9-12.*,38	9830,14.18.103
4078-4103,14.28.119,66	8735,13.30.166
1746-1801,6.2-25.494-629,51	1048,13.3.734
6437-6526,8.28.789,57	1146,5.25.351
694-776,2.26.123-987,77	2443,0.3.457
6266-6363,1.0-0.*,30	5610,9.10.625
4149-4183,0.25-25.662-860,28	9458,13.5.791

그림 15 질의 데이타 및 Stabbing 질의 예제

실험에서 사용된 질의 데이타 집합의 종류는 표 2, stabbing 질의의 집합의 종류는 표 3과 같다.

표 2 질의 데이타 집합의 종류

종류	RID 범위	질의 데이타 수
QU1	[1, 100]	100
QU2	[1, 100]	500
QU3	[1, 100]	1000
QU4	[1, 100]	5000
QU5	[1, 100]	10000
QU6	[1, 100]	20000
QU7	[1, 100]	50000
QU8	[1, 100]	100000

표 3 Stabbing 질의 데이타 집합의 종류

종류	분포	RID 범위	질의 데이타 수
DU1	Uniform	[1, 100]	100000
DG1	Gaussian	[1, 100]	100000
DS1	Skewed	[1, 100]	100
DS2	Skewed	[1, 100]	500
DS3	Skewed	[1, 100]	1000
DS4	Skewed	[1, 100]	5000
DS5	Skewed	[1, 100]	10000
DS6	Skewed	[1, 100]	20000
DS6	Skewed	[1, 100]	50000

5.2 삽입 성능 비교

그림 16은 기존 연구에서 stabbing 질의를 개별적으로 처리하는 경우와 Sequence로 처리하는 경우에 따른 질의 색인의 구축 비용에 대한 실험 결과를 보여준다. 데이타의 분포는 균등 분포를 사용하였으며 데이타 개수를 증가시키면서 실험을 수행 하였다.

실험결과는 Sequence를 사용하는 방법이 근소하게 많은 시간 비용이 소모되었다. 그리고 삽입되는 데이타의 수가 커질수록 구축 비용의 차이가 커지는 것을 알 수 있다. 그러나 Sequence Set은 구축된 뒤에는 추가되거나 변경 및 삭제 빈도가 적기 때문에 처리 시간 비용이 상대적으로 급격하게 증가하지는 않는 것을 알 수 있다.

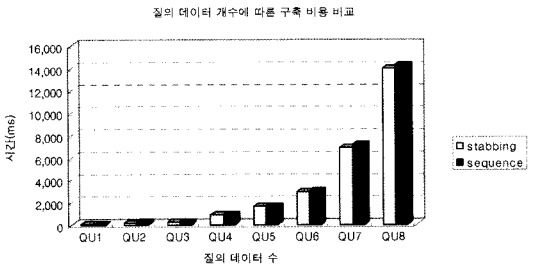

그림 16 질의 데이타 개수에 따른 구축 비용

5.3 최대 허용 간격에 따른 실험

최대 허용 간격의 크기를 다양하게 설정하여 최대 허용 간격의 최적 값을 구하는 실험을 수행하였다. 실험에 사용된 데이타는 QU8이며, stabbing 질의는 분포별로 DU1, DG1, DS8을 사용하였다. 비교 요소로는 총 질의 횟수, 질의 색인의 노드 접근 수 그리고 총 질의 처리 소모 시간을 사용하였다. 그림 17과 그림 18은 최대 허용 간격에 따른 총 질의 횟수 및 색인의 노드 접근 수를 보여준다. 최대 허용 간격이 커질수록 모든 분포에서 질의 횟수가 줄어들고 노드 접근 횟수도 줄어듬을 알 수 있다. 또한 질의 횟수가 적을수록 노드 접근 수 또한 적어지는 것을 알 수 있다.

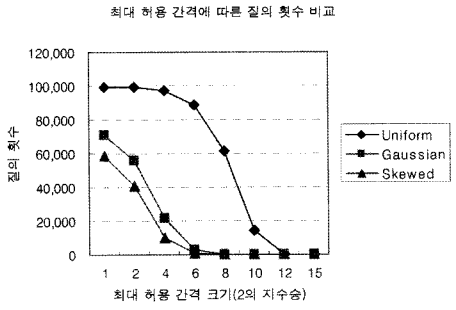

그림 17 최대 허용 간격 크기에 따른 질의 횟수

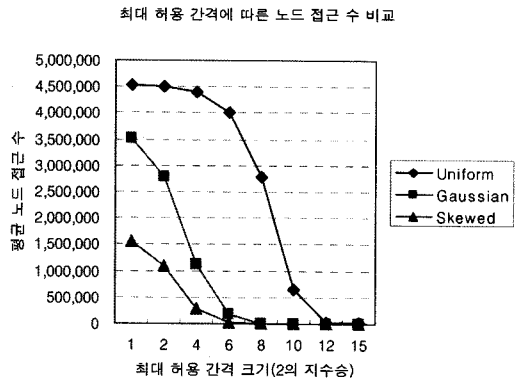

그림 18 최대 허용 간격 크기에 따른 노드 접근 수

그림 19는 최대 허용 간격에 따른 총 질의 처리 시간을 보여준다. 최대 허용간격이 균등 분포에서는 2^{10}, 가우시안 분포와 사향 분포에서는 2^4일 때 가장 좋은 성능을 보인다. 즉, stabbing 질의의 분포에 따라서 최대 허용 간격의 최적 값이 다른 것을 알 수 있다. 또한 최대 허용 간격이 이보다 더 커지면 질의 횟수나 노드 접근 수가 적어지는 것에 비해서 총 질의 소요 시간 비용은 급격하게 증가하는 것을 알 수 있다. 최대 허용 간격이 일정 크기 이상 일 경우에는 큰 틈을 가지는 Pseudo-Sequence가 생성되기 때문에 노드 접근 수는 적으나 그에 따른 정제 단계의 비용이 매우 커지기 때문이다.

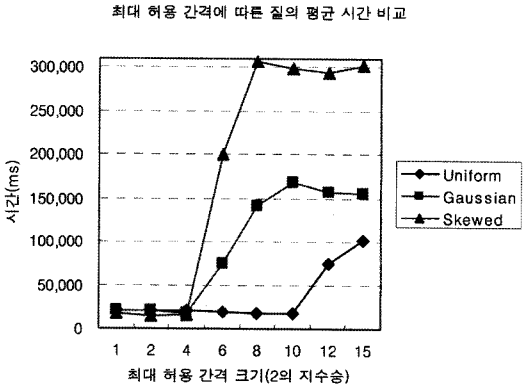

그림 19 최대 허용 간격 크기에 따른 질의 처리 시간

5.4 SequenceSet의 삽입 방법에 따른 성능 비교

그림 20, 그림 21, 그림 22는 SequenceSet의 삽입 방법인 단순 삽입 방법과 개선된 삽입 방법에 따른 질의 횟수, 질의 처리 소요 시간 및 노드 접근 수를 각각 보여준다. 질의 데이타는 QU8을 사용하였으며 논문의 대상환경과 유사한 환경에서의 실험을 위해 stabbing 질의 분포로써 사향 분포의 stabbing 질의 집합을 사용하였다. 그리고 개선된 삽입 방법에서는 사향 분포에서의 최적의 최대 허용 간격 값인 24를 사용하였다.

그림 20, 그림 21 그리고 그림 22에서 보듯이 stabbing 질의 셋이 커질수록 색인에서의 질의 수행 횟수, 총 질의 처리 소요 시간 및 노드 접근 횟수 등의 모든 부분에서 개선된 삽입 방법이 좋은 결과를 보였다.

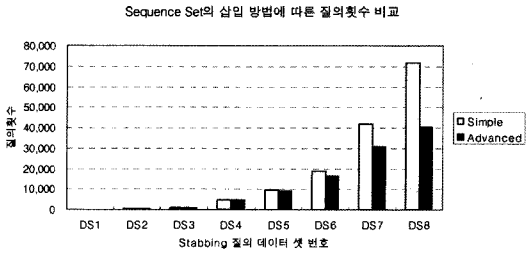

그림 20 SequenceSet의 삽입 방법에 따른 질의 횟수

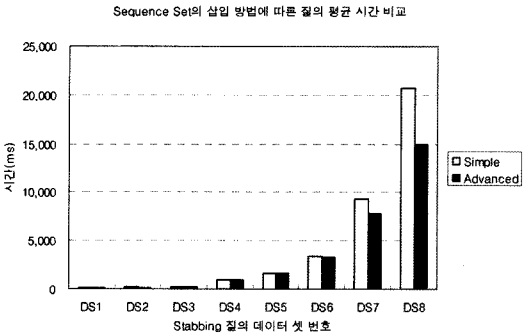

그림 21 SequenceSet의 삽입 방법에 따른 질의 처리 시간

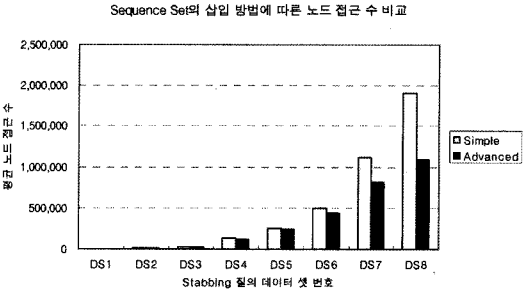

그림 22 SequenceSet의 삽입 방법에 따른 노드 접근 수

5.5 질의 처리 기법에 따른 성능 비교

그림 23, 그림 24, 그림 25는 기존의 개별 stabbing 질의 처리 방법과 본 논문에서 제안한 Sequence를 이용한 질의 처리 방법에 대해서 질의 색인에서의 질의 수행 횟수, 질의 처리 소요 시간 및 노드 접근 수를 각각 보여준다. 데이타는 QU8을 사용하였으며 stabbing

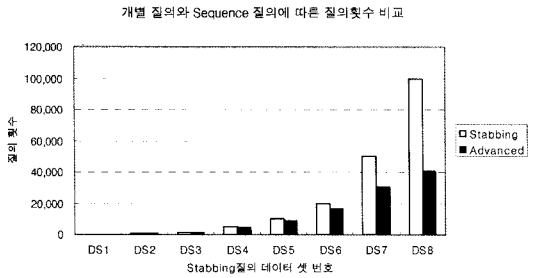

그림 23 개별 질의 및 sequence 질의에 따른 질의 횟수

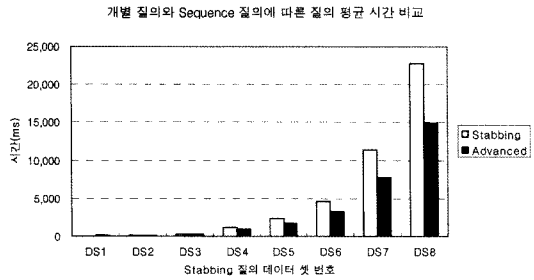

그림 24 개별 질의 및 sequence 질의에 따른 질의 처리 시간

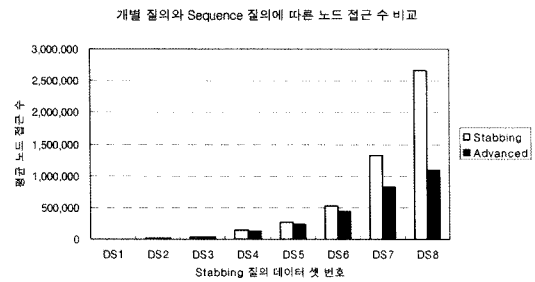

그림 25 개별 질의 및 sequence 질의에 따른 노드 접근 수

질의 분포는 사향 분포를 사용하였다. Stabbing 질의 수가 증가할수록 질의 횟수, 노드 접근 수 및 질의 처리 소요 비용에서 개별적인 stabbing 질의 처리 성능보다 향상된 결과를 보였다.

6. 결론 및 향후 연구

RFID 물류 환경에서 태그를 부착한 제품들은 개별적으로 이동하는 것이 아니라 유사 제품끼리 다수의 그룹을 지어 이동하는 특징이 있다. 이로 인해 태그가 판독기에 인식될 때 발생하는 태그 이벤트는 짧은 기간 동안 대량으로 발생하며 그 식별자 또한 유사한 값들을 가진다. 기존의 연구에서는 태그 이벤트를 stabbing 질의로써 처리할 때 이러한 특징을 고려하지 않고 개별적으로 처리함으로써 질의 처리 시 동일 또는 유사 검색 과정을 반복하는 한계가 있다.

이 논문에서는 반복 검색 문제를 해소하기 위해 연속적으로 발생하는 태그 이벤트를 질의 집합인 sequence로 구성하여 영역 질의로 처리하는 방안을 제안하였다. 이를 위해 stabbing 질의들을 분류 및 수집하기 위한 논리적 리더 큐의 구성과 질의 집합에서 sequence를 구성하기 위한 자료구조로써 SequenceSet을 제안하였다. SequenceSet에서는 비정렬 및 불연속적인 stabbing 질의 집합에서 효율적으로 sequence를 구성하기 위한 알고리즘을 제시하였다. 그리고, 제안한 기법에 대해 다양한 조건에서의 실험을 수행하였으며 기존의 stabbing 질의 처리 기법에 비해 질의를 큐에 유지하기 위한 삽입 시간 비용은 다소 증가하였으나 이러한 stabbing 질의를 처리할 때의 성능은 월등히 향상되었다. 이러한 처리 방법이 유효한 이유는, ECSpec이 일정기간 질의 결과(태그 이벤트 정보)를 모으는 형태의 연속 질의이고 각 태그 이벤트에는 태그 식별자, 인식된 시간 정보가 포함되기 때문에 개별 처리 방법과 영역 질의 처리 방법에 대한 결과셋의 구성이 동일하기 때문이다. 제안된 기법은 RFID 태그와 같이, 식별자가 부여되는 자동 인식 장비로 구축되는 물류환경에서 효과적으로 적용될 수 있다.

참 고 문 헌

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

[2] S. R. Madden, M. A. Shah, J. M. Hellerstein and V. Raman, "Continuously adaptive continuous queries over streams," *ACM SIGMOD*, pp. 49-60, 2002.

[3] K. Traub, S. Bent, T. Osinski, S. N. Peretz, S. Rehling, S. Rosenthal and B. Tracey, "The Application Level Event(ALE) Specification, Version1.0," *EPCglobal*, 2005.

[4] Dmitri V. Kalashnikov, Sunil Prabhakar, Susanne E. Hambrusch, Walid G. Aref, "Efficient Evaluation of Continuous Range Queries on Moving Objects," *DEXA 2002*, pp. 731-740, 2002.

[5] 박재관, 홍봉희, 반재훈, "RFID 스트리밍 데이타 처리를 위한 연속 질의의 변환 기법", *정보처리학회 논문지*, pp. 273-284, 2007.

[6] 석수욱, 박재관, 홍봉희, "RFID 미들웨어에서 연속질의 처리를 위한 질의 색인 기법", *한국정보과학회 한국컴퓨터종합학술대회 2005 논문집(B)*, pp. 28-30, 2005.

[7] Kun-Lung Wu, Shyh-Kwei Chen and Philip S. Yu, "VCR indexing for fast event matching for highly-overlapping range predicates," *SAC 2004*, pp. 740-747, 2004.

[8] J. Chen et al, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases," ACM SIGMOD, pp. 379-390, 2000.
[9] S. Chandrasekaran et al, "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," *In Proc. First Biennial Conf. on Innovative Data Systems Research*, pp. 269-280, 2003.
[10] E. N. Hanson et al, "A Predicate Matching Algorithm for Database Rule Systems," *ACM SIGMOD*, pp. 271-280, 1990.
[11] 석수욱, 박재관, 홍봉희, "RFID 태그 데이타의 연속질의 처리를 위한 질의 색인", *한국정보과학회 논문지*, pp. 166-178, 2007.
[12] "EPC Generation 1 Tag Data Standards Version 1.1 Rev.1.2.7," *EPCglobal*, 2005.
[13] A. Guttman, "R-Tree: A dynamic index structure for spatial searching," *In Proc. of the 1984 ACM SIGMOD Intl. Conf. on Management of Data*, pp. 47-57, 1984.
[14] Kyounghwan An and Bonghee Hong, "Growing Node Policies of a Main Memory Index Structure for Moving Objects Databases," *15th International Conference, DEXA 2004*, pp. 834-843, 2004.

박 재 관

1999년 부산대학교 컴퓨터공학과 졸업(학사). 2001년 부산대학교 대학원 컴퓨터공학과 졸업(공학석사). 2008년 부산대학교 대학원 컴퓨터공학과(공학박사). 2008년~현재 LG전자기술원 선임연구원. 관심분야는 모바일 임베디드 DBMS, 유비쿼터스 시스템, 공간 데이타베이스

홍 봉 희

1982년 서울대학교 전자계산기공학과 졸업(학사). 1984년 서울대학교 대학원 전자계산기공학과 졸업(공학석사). 1988년 서울대학교 대학원 전자계산기공학과 졸업(공학박사). 1987년~현재 부산대학교 컴퓨터공학과 교수. 관심분야는 공간 데이타베이스, RFID 시스템, RFID 데이타베이스

이 기 한

2005년 부산대학교 컴퓨터공학과 졸업(학사). 2007년 부산대학교 대학원 컴퓨터공학과 졸업(공학석사). 2007년~현재 삼성전자 연구원. 관심분야는 데이타베이스, 이동체 데이타베이스, RFID 시스템