제 26 회 한국정보처리학회 추계학술발표대회 논문집 제 13 권 제 2 호 (2006. 11)

스트림 데이터 처리를 위한 질의 색인 기법

이동규*, 정재두**, 이양구*, 정영진*, 류근호*
*충북대학교 데이터베이스 / 바이오인포매틱스 연구실
**육군본부
e-mail : {dglee*, leeyangkoo*, yjjeong*, khryu*}@dblab.chungbuk.ac.kr
**chungjaedu@hanmail.net

Query Indexing Technique for Processing Stream Data

Dong Gyu Lee*, Jae Du Chung**, Yang Koo Lee*, Young Jin Jung*, Keun Ho Ryu*
*Database / Bioinformatic Laboratory, Chungbuk National University
**ROK

요 약

센서 네트워크 환경에서 스트림 데이터를 모니터링 하기 위해서는 스트림 데이터에 대한 연속적인 질의들을 효과적으로 처리하는 것이 필요하다. 이러한 연속적인 질의를 빠르게 검색하고 처리하기 위하여 낮은 저장 비용과 빠른 탐색 성능을 가진 질의 색인 기법이 많이 활용되고 있다. 기존 연구들은 사전에 삽입될 Interval 을 알고 트리를 구성하므로 동적인 삽입, 삭제가 불가능하거나 삽입된 Interval 수와 Interval 의 범위에 따라 높은 저장 비용이나 상대적으로 느린 탐색 속도를 보인다. 따라서 이 논문에서는 연속적인 질의 처리를 효율적으로 하기 위하여 Hashed Multiple Lists 를 제안한다. 제안된 기법은 빠른 선형 탐색 성능과 낮은 저장 비용을 요구하며 삽입, 삭제가 용이하고 다양한 범위를 표현할 수 있는 장점이 있다. 제안된 색인 기법은 센서 네트워크를 응용한 시스템과 상황 인식 시스템 등에서 연속적인 질의를 처리하는데 활용할 수 있다.

1. 서론

센서 네트워크 및 모바일 컴퓨팅 기술의 발달로 실생활에 센서를 활용한 ad-hoc 네트워크를 구성하기 위해 다양한 연구가 계속되고 있다[1, 2]. 이러한 네트워크를 기반으로 지능적인 유비쿼터스 서비스를 제공하기 위해서는 먼저, 센서로부터 입력되는 데이터를 조합하고 질의하는 것이 필요하다. 그러나 정적인 데이터를 처리 하기 위한 기존의 언어 및 연산자를 활용하여 센서 네트워크에서 발생하는 연속적인 스트림 데이터를 처리하는 것은 적절하지 못하기 때문에, 센서 데이터의 특성에 알맞은 시스템 및 처리 기법에 대한 설계가 필요하다[3, 4, 5].

센서 데이터와 기존 데이터베이스의 데이터 사이는 두 가지의 큰 차이점이 있다[1]. 첫째, 스트림 데이터는 실시간으로 처리 되어야 한다. 교통사고, 네트워크 침입 등과 같이 즉시 응답해야 할 필요가 있는 실시간 이벤트를 처리해야 하며, 스트림 데이터를 디스크에 저장하기에는 매우 높은 저장 비용이 요구된다. 둘째, 센서들은 낮은 성능의 프로세서와 제한된 전력 자원을 갖기 때문에, 질의 엔진은 가능한 소모를 줄이는 것이 필요하다.

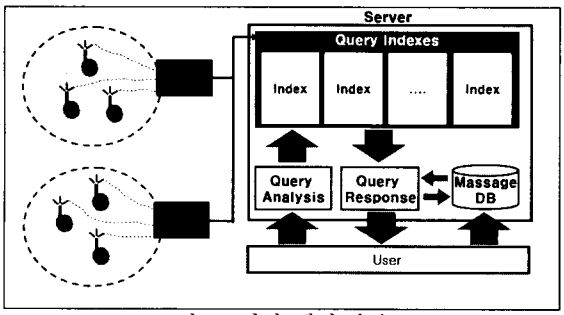
이러한 스트림 데이터를 모니터링하고 효율적으로 처리하기 위해서는 필요한 만큼 많은 연속적인 질의가 만들어 질 수 있고 입력되는 스트림 데이터에 대해 각 데이터 아이템을 연속적으로 처리 해야 한다[3]. 연속적인 질의를 빠르게 평가하기 위해서 다양한 질의 색인 기법이 사용되고 있다.

질의 처리를 위한 데이터 값의 범위를 저장하는 질의 색인은 스트림 데이터가 시스템에 입력될 때마다 질의 색인에 저장된 데이터의 범위를 확인하고, 전송된 데이터가 특정 질의 범위를 만족하면, 그 범위에 해당하는 질의 식별자를 반환하여, 질의가 처리되도록 한다. 좋은 질의 색인은 연속적인 스트림 데이터를 처리하기 위하여 낮은 저장 비용과 빠른

본 연구는 산업자원부 · 한국산업기술평가원 지정 청주대학교 정보통신연구센터의 지원에 의한 것입니다.

탐색 성능을 가져야 한다. 또한, 많은 질의를 메모리에 저장할 수 있어야 한다.

(그림 1)의 질의 색인을 응용 구조이다. 서버에는 질의 색인, 질의 분석기, 질의 응답기, 그리고 메시지 DB 로 구성된다. 사용자는 질의를 질의 분석기로 보내면 분석기는 질의의 조건 절을 확인하고 각 속성에 대한 Interval 을 만든다. 만들어진 Interval 은 속성에 따라 질의 색인을 구성하고 In-Situ 센서 네트워크로부터 입력되는 스트림 데이터를 받아 해당하는 Interval 을 검색하여 질의 식별자를 반환한다. 질의 응답기는 반환한 속성별 식별자를 가지고 일치하는 메시지를 메시지 DB 에서 검색하여 메시지를 통해 사용자에게 통보해주거나 Actuator 를 동작시켜 상황에 대처한다. 이와 같은 응용은 효율적인 스트림 데이터 처리나 상황 인식 시스템에서 응용이 가능하며 질의 색인으로 인해 많은 질의와 빠른 스트림 데이터를 동시에 처리 해줄 수 있는 장점이 있다.



(그림 1) 질의 색인 응용 구조

이와 같은 응용 구조를 통해 질의 색인이 스트림 환경에서 효율적인 연속적인 질의들을 처리하는 것을 확인할 수 있다. 만약 질의 색인이 없다면, 연속적인 질의를 하나씩 처리해야 하거나 또는 데이터베이스에 저장한 후에 처리해야 하므로 저장 및 검색 비용이 많이 소요된다. 따라서 스트림 환경에서 연속적인 질의를 효율적으로 처리하기 위해서는 질의 색인이 필요하다.

따라서 이 논문에서는 연속적인 질의를 효율적으로 평가하기 위해서 Hashed Multiple Lists 를 제안한다. 제안된 기법은 빠른 탐색 속도를 보이며 다양한 Interval 삽입이 가능하여 스트림 데이터 환경에서 질의 처리 시스템이나 센서 네트워크를 이용한 상황인식 시스템에도 적용 가능하다.

2. 관련 연구

질의 색인에 대한 기존 연구로써 하나의 상수 값을 포함하는 Interval 들을 찾는 질의인 Stabbing query[3, 6]을 하는 색인들과 스트림 환경에서 빠른 탐색성능을 보이는 CEI-based query indexing[3]을 설명한다. Segment trees[6], Interval trees[6], Interval binary search trees(IBS-tree)[7], Interval skip lists(IS-lists)[8]와 같은 기존 연구들이 있다. 이러한 기존 연구들은 일반적으로 스트림 데이터 환경에 맞게 설계된 기법이 아니므로 스트림 데이터 환경에는 안정적이지 않다.

Segment tree 와 Interval tree 는 사전에 모든 interval 이 트리를 구성하고 있어야 하므로 동적인 삽입, 삭제가 어렵다[6].

IBS-tree 와 IS-lists 는 메인 메모리 기반 질의 색인으로 설계되었다. 두 기법은 많이 겹치는 Interval 을 다루는 첫 번째 동적인 기법이고 두 기법은 유사한 원리를 가졌지만 IS-lists 가 구현이 더 간단하므로 더 좋은 성능을 가졌다[8]. 또한, 질의 범위를 다양하게 표현할 수 있으며, 규칙에 따라 적절한 사람에게 메시지를 전달하는 상황 정보 필터에도 적용이 가능하다[9]. 삽입된 Interval 수가 많고 검색하고자 하는 값이 헤더로부터 멀수록 탐색시간은 증가한다. 긴 Interval 을 삽입 시, 삽입비용도 크다. IBS-tree 와 IS-lists 는 n 을 Interval 의 전체 수라고 하면, $O(\log(n))$의 탐색 시간과 $O(n\log(n))$의 저장 비용을 요구한다.

가장 최근에 스트림 데이터를 효율적으로 처리하기 위해 제안된 CEI-based query indexing 이 있다. 이 기법도 Stabbing query 와 같은 수행을 하며 이전의 기존 연구와는 다르게 스트림 환경에 적합한 빠른 검색 성능, 낮은 저장 비용, 동적인 삽입과 삭제가 가능하다. 하지만 전체 Interval 의 수가 10 만개 이하일 때 저장 비용이 IS-lists 보다 크며 긴 Interval 을 삽입 시 비용 또한 크다. 이렇게 긴 Interval 삽입이 많은 경우 ID 리스트에 같은 질의 ID 가 중복 삽입되므로 검색 성능은 저하된다.

따라서 기존 연구들의 탐색 비용은 삽입된 Interval 의 수와 길이에 비례하여 증가하고 Interval 의 범위가 크고 많은 Interval 을 처리하는데 효율적이지 못한 문제점을 가지고 있다.

3. Hashed Multiple Lists

연속적인 질의를 색인화하고 간단한 해쉬 테이블을 사용하여 스트림 데이터를 빠르게 처리할 수 있는 Hashed Multiple Lists(HMLists)는 Interval Skip Lists(IS-lists)을 확장한 구조이다. IS-lists 와 같은 구조 상에서 Interval 수와 길이가 증가하면 탐색 성능이 저하되는 것을 막기 위해 간단한 해쉬 테이블을 사용하였고 리스트의 높이에 따라 노드의 레벨을 결정하므로 노드 접근 수를 줄여 탐색 성능을 향상 시켰다. 이 장에서는 제안된 질의 색인이 다루고자 하는 질의 유형과 변환된 Interval 을 알아보고 구성 과정과 탐색 과정을 설명하겠다. (그림 2)는 연속적인 질의와 질의의 조건 절을 변환하여 만든 Interval 을 보여준다. 이 논문에서는 Interval 형태의 하나의 조건을 가지는 (그림 2)와 같은 간단한 질의에 초점을 맞추고 있다. 그러나 제안한 질의 색인은 (그림 1)의 질의 색인 응용 구조와 같이 복수의 속성 상에서 Interval 형태의 다양한 조건들을 가지는 복잡한 질의에도 적용할 수 있다.

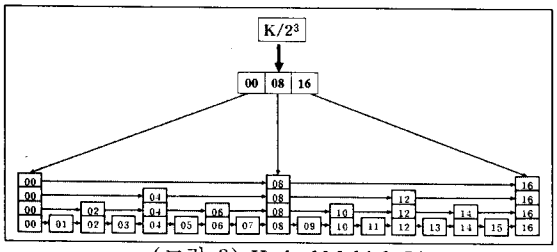
Query 1:
SELECT QID ***FROM*** IDList AS L
WHERE L.Temperature>=50 ***AND*** L.Temperature <200

Interval T1: [50, 200)

(그림 2) 연속적인 질의와 Interval 의 유형

속성은 정수 타입 또는 실수 타입이 될 수 있고 질의 Interval 은 정수 타입의 두 개의 끝점으로 표현된다. 즉, Interval 들은 s 와 e 는 정수이고 s < e 인 조건 하에 [s, e], [s, e), (s, e], 그리고 (s, e)와 같이 다양하게 표현될 수 있다.

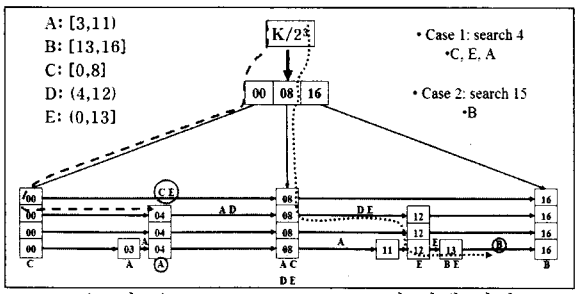
(그림 2)의 Query 1 과 같은 질의를 (그림 1)에서 설명한 질의 분석기를 통해서 Interval T1 을 생성한다. 이렇게 생성된 Interval 은 (그림 3)과 같이 질의 색인을 구성할 수 있으며 (그림 4)과 같이 Stabbing query 을 할 수 있다.



(그림 3) Hashed Multiple Lists

(그림 3)는 이진 트리 형태로 노드를 생성한 HMLists 의 구조를 나타낸다. Interval 삽입 시에는 최소값과 최대값에 대해 두 번 삽입이 발생하고 하나의 노드를 만들 때 노드의 레벨을 결정하게 된다. 레벨 결정 기준은 리스트의 높이에 따라 각 노드의 레벨을 이진 트리 형태로 판정한다. 예를 들어, 노드 3 은 이진 트리에서 보면 말단 노드가 되므로 (그림 3)에서 보듯이 레벨 1 이 된다. 이렇게 레벨이 결정된 노드를 만들고 들어오는 포인터와 나가는 포인터를 이웃 노드에 연결한다. 이와 같은 연결 후에는 Interval 식별자을 삽입하고 삭제 시에는 해당 노드들과 노드들 사이의 포인터들을 찾아서 질의 ID 를 삭제 해주면 된다. 해당 노드들까지 삭제하지 않는 이유는 탐색 시 큰 영향을 미치지 않고 오히려 재삽입 시 트리 구성 비용이 발생하므로 질의 ID 만 삭제하는 것이다.

탐색은 해쉬 테이블을 이용하여 빠르고 선형적으로 만들 수 있다. 값이 들어오면 해쉬 테이블을 통해 레벨 4 를 가지는 해당 노드로 이동하고 레벨 4 를 가지는 노드로부터 출발하여 탐색을 하게 된다. 예를 들면, (그림 4)는 HMLists 의 탐색 값 4 와 15 을 가진 Interval 들을 탐색하는 과정을 보여주고 있다. 탐색 값 K 가 4 라고 하면, HMLists 는 말단 노드를 8 개를 가지는 이진 트리 형태이므로 해쉬 테이블을 통해 노드 0 에 접근하게 된다. 노드 0 의 레벨 4 에서부터 탐색 값 K 를 찾을 때까지 포인터를 따라 탐색하고 탐색 값보다 노드 값이 작으면 레벨을 내리면서 가지고 있던 Interval 식별자를 반환하게 된다. 하지만 레벨 4 에서는 탐색을 하지 않는다. 레벨 4 에서 가리키는 다음 노드는 이미 해쉬 테이블에 존재하므로 탐색이 필요할 경우 해쉬 테이블에서 직접 가리키기 때문이다. 따라서 K=4 인 값을 탐색 시 레벨 4 에서는 레벨을 한 단계 내리면서 C 와 E 를 반환하고 레벨 3 에서 포인터를 따라 탐색을 한 후 우리가 찾고 있는 K=4 인 노드를 발견하여 해당 식별자인 A 을 반환하고 탐색을 마친다.



(그림 4) Hashed Multiple Lists 의 탐색 과정

두 번째로 15 을 포함하는 Interval 을 탐색한다. 해쉬 테이블을 통해 레벨 4 인 노드 8 에 접근한다. 이것은 Interval 의 수와 상관없이 빠른 탐색이 가능하다는 것을 보여준다. 노드 8 의 레벨 4 에서 레벨 3 으로 내리고 노드 12 로 이동한다. 노드 12 의 레벨 3 에서 레벨 2 로 내리고 13 을 거쳐 탐색하지만 노드 15 는 없다. 따라서 노드 13 의 레벨 1 의 포인터에 저장된 B 을 반환한다. 이 전 과정에서 레벨을 내릴 때 반환할 Interval 식별자가 존재하지 않았기 때문에 반환하지 못했다. 따라서 결과는 B 이다.

우리는 HMLists 의 탐색 연산에 대해 살펴봤다. 제안된 기법의 탐색 연산은 간단한 해쉬 테이블을 이용해 Interval 의 수와 관계없이 선형 탐색이 가능하며 최대 노드 개수가 8 개 이내로 접근이 가능하다. 이는 빠른 스트림 데이터를 효율적으로 처리 할 수 있다는 것을 보여준다. 다음은 제안된 기법의 알고리즘을 기술하였다.

4. Hashed Multiple Lists 알고리즘

HMLists 는 삽입, 삭제, 탐색, 그리고 레벨 판정 알고리즘으로 구성된다. 하지만 이 논문에서는 공간상의 제약으로 탐색, 레벨 판정 알고리즘을 각각 단계별로 설명하겠다.

4.1 탐색 알고리즘

(그림 5)는 스트림 데이터를 통해 들어온 값을 포함하는 Interval 식별자를 찾는 탐색 함수이다. (그림 5)의 Step 1 은 탐색 노드가 레벨 4 인 노드이면 다음 노드에 상관없이 해당 Interval 식별자를 반환하

고 레벨을 한 단계 내린다. Step 2 는 노드 값이 탐색 값보다 작으면 레벨을 내리고 해당 Interval 식별자를 반환하는 작업을 반복한다. Step 3 은 탐색 값과 같은 값을 가지는 노드가 존재하지 않으면 현재 노드의 포인터가 가지고 있는 Interval 식별자를 반환하고 존재하면 Step 4 와 같이 해당 노드의 Interval 식별자를 반환하고 탐색을 마친다. 탐색 시간 비용은 $O(\log n)$을 갖는다.

```
Algorithm HML_Search(K)
INPUT     K : Search key
OUTPUT    S : set of searched query IDs
begin
  S:=Ø
  i:=maximum level

// Step 1. We don't search, down to 1 level in 4 level
  h := K / 8
  x := 8h node allocating hashtable[h]
  while i≥0 and (x is 4 level's node and x→ key≠K) do
    S:=S∪x→ edgeID[i]
    i:=i-1

// Step 2. if x>K , then down to 1 level
    while x→ forward[i]≠null and x→ forward[i]→ key<K do
      x:=x→ forward[i]
    endwhile

// Step 3. if x=K, return ID[i]
    if x is not the header and x→ key≠K then
      S:=S∪x→ edgeID[i]

// Step 4. if i=bottom level and x=K, finish search operation
    else if x is not the header then
      S:=S∪x→ ID[i]
    i:=i-1
  endwhile
  return S
end
```

(그림 5) 탐색 함수

4.2 레벨 판정 알고리즘

```
Algorithm HML_AdjustLevel (K)
INPUT     K: Key for adjusting level
OUTPUT    Level: Level of K
begin
  n:=0
  H:= height of HMLists
  Level:=0

// Step 1. computing L using hash function
  L := K % 2^(H-1)

// Step 2. After checking condition, return Level
  if(L=0) then Level = 3
  else if(L=4) then Level = 2
  else if(L=2 || L=6) then Level = 1
  else Level = 0
  return Level
end
```

(그림 6) 노드 레벨 판정 함수

(그림 6)는 하나의 Interval 는 최소 값과 최대 값을 가지게 되며 각 값은 노드가 된다. 각 노드는 레벨을 갖게 되는데 HMLists 는 (그림 3)과 같은 완전 이진 트리 형태로 레벨을 판정하게 된다. (그림 6)의 알고리즘은 HMLists 의 높이가 4 이고 하위 노드의 개수가 8 인 완전 이진 트리 형태로 구성된 알고리즘이다. 최상위 레벨부터 하위 레벨까지 판정하는데 Step 1 의 간단한 수식으로 K 의 나머지 값을 계산하여 Step 2 에서 레벨을 결정하게 된다.

5. 결론

우리는 스트림 데이터 환경에서 연속적인 질의를 효율적으로 처리하기 위한 Hashed Multiple Lists 를 제안하였다. 제안된 기법은 질의 범위를 삽입, 삭제 할 수 있고 리스트의 높이에 따라 각 노드의 레벨을 이진트리 형태로 결정하기 때문에 노드 접근 수를 최소화 할 수 있어 빠른 탐색 속도를 보인다. 또한, 해쉬 테이블을 사용하여 질의의 수가 증가해도 선형 탐색이 가능하다. 제안된 기법은 질의 처리 시스템이나 센서 네트워크를 이용한 상황인식 시스템에도 적용 가능하다. 향후 연구로는 제안된 기법의 구현 및 성능 평가가 필요하다.

참고문헌

[1] S. R. Madden, and M. Franklin, "Fjording the stream: An architecture for queries over streaming sensor data", 18th International Conference on Data Engineering, pp 555-566, 2002.

[2] J. M. Hellerslein, W. Hong, and S. R. Madden, "The Sensor Spectrum: Technology, Trends, and Requirements", ACM SIGMOD Record, Vol. 32, issue 4, pp 22-27, 2003.

[3] K. L. Wu, S. K. Chen, and P. S. Yu, "Interval Query Indexing for Efficient Stream Processing", In Proc. of ACM Int. Conf. on Information and Knowledge Management, pp 88-97, 2004.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems", Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp 1-16, 2002.

[5] L. Golab, and M. T. Özsu, "Issues in Data Stream Management", ACM SIGMOD Record, Vol. 32, No. 2, pp 5-14, 2003.

[6] H. Samet, "Design and Analysis of Spatial Data Structures", Addison-Wesley, 1990.

[7] E. N. Hanson, and M. Chaabouni, "The IBS tree: A data Structure for finding all intervals that overlap a point", Technical Report WSU-CS-90-11, Wright State University, 1990.

[8] E. N. Hanson and T. Johnson, "Selection predicate indexing for active databases using interval skip lists", Information Systems, Vol. 21, No. 3, pp 269-298, 1996.

[9] J. P. Dittrich, P. M. Fischer, and D. Kossmann, "AGILE: Adaptive Indexing for Context-Aware Information Filters", Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp 215-226, 2005.