연속 데이터 스트림에서 효율적인 이벤트 필터링 기법

김 현 규 ○ 강 우 람 ○ 김 명 호

한국과학기술원 전산학과

{hgkim, wlkang, mhkim}@dbserver.kaist.ac.kr

A Method for Efficient Event Filtering over Continuous Data Streams

Hyeon Gyu Kim, Woo Lam Kang, Myoung Ho Kim

Department of Computer Science

Korea Advanced Institute of Science and Technology

요 약

일반적으로 연속 데이터 스트림을 모니터링하는 응용은 다수의 범위 질의를 포함한다. 이러한 다수의 범위 질의는 술어 색인을 이용해 효율적으로 처리할 수 있다. IBS-tree는 연속 데이터 스트림 상에서 효과적으로 이용될 수 있는 술어 색인 기법 중 하나이다. 그러나 IBS-tree는 모든 노드에서 등호 검사와 부등호 검사를 함께 실시하며, 이는 검색 성능의 저하로 이어질 수 있다. 본 논문에서는 등호 검사와 부등호 검사를 분리하여 수행함으로써 검색 성능을 향상시키는 술어 색인 방법을 제안한다. 제안하는 방법은 등호 검사를 위해 해싱을 이용하고, 부등호 검사에는 균형 이진 검색 트리를 이용한다. 본 논문에서는 실험을 통해 IBS-tree와 제안하는 방법의 검색 성능을 비교하였으며, 실험 결과로부터 제안하는 방법의 성능이 더욱 우수한 것을 확인하였다.

1. 서 론

최근 들어 온라인 경매, 주식 거래, 네트워크 통계, 웹 페이지 방문 통계, 센서 데이터 수집 등 연속 데이터 스트림을 처리하는 응용(Application)에 대한 다양한 연구가 진행되고 있다[1, 5]. 이러한 응용들은 일반적으로 다수의 범위 질의(Range query)를 포함하고 있다. 예를 들어 주식 거래 응용에서는 사용자의 관심을 반영한 다수의 범위 질의가 주식 가격의 변화를 모니터링하기 위해 등록되고 이용된다.

술어 색인(Predicate index)[2]은 다수의 범위 질의를 효율적으로 처리하기 위한 방법 중 하나이다. 술어 색인은 입력 튜플이 주어졌을 때, 해당 튜플의 값이 어느 질의의 조건을 만족시키는지 찾아내기 위해 이용된다. 그림 1은 균형 이진 검색 트리(Balanced binary search tree)로 된 술어 색인의 예를 보여준다. 검색 트리의 노드는 주어진 질의의 범위 값의 종단점(Endpoint)으로 구성된다. 주어진 n개의 종단점은 입력 값의 도메인(Domain)을 (n+1)개의 분리된 범위(Separate region)들로 나눈다. 입력 튜플이 전달되었을 때, 해당 튜플은 검색 트리를 통해 그 범위들 중 하나에 대응된다. 예를 들어, 그림 1에서 입력 튜플 8은 범위 r5에 대응된다. 따라서 튜플 8에 대한 검색 결과는 Q1과 Q3가 된다.

본 연구는 건설교통부 첨단도시기술개발사업 - 지능형국토정보기술혁신 사업과제의 연구비 지원(07국토정보C05)에 의해 수행되었습니다.

IBS-tree(Interval binary search tree)[2]는 연속 스트림 상에서 효율적으로 이용될 수 있는 술어 색인 방법 중 하나이다. IBS-tree에서는 입력 튜플에 대해 모든 트리 노드에서 등호 검사(equality test)와 부등호 검사(inequality test)를 함께 수행한다. 그러나, 등호 검사를 매 노드마다 수행하는 것은 검색 성능을 저하시킬 수 있다. 예를 들어, 입력 값의 범위가 매우 크고(e.g. 1부터 100,000) 노드의 개수는 상대적으로 적은(e.g. 100개) 경우를 가정해 보자. 만약 입력 값이 균등하게 분포될 경우, 하나의 튜플이 등호 검사를 만족시킬 확률은 매우 낮아진다(e.g. 1/1000). 이러한 경우 등호 검사를 매 노드마다 수행하는 것은 비효율적이다. 비효율성은 노드의 개수가 늘어날 수록 더욱 증가한다. 이는 노드가 증가할수록 트리의 높이(Height)는 급격하게 증가함에 비해, 등호 검사를 만족시킬 확률은 크게 증가하지 않기 때문이다.

본 논문에서는 등호 검사와 부등호 검사를 분리하여 수행함으로써 검색 성능(Search performance)을 향상시키는 술어 색인 방법을 제안한다. 제안하는 방법에서는 등호 검사를 위해 해싱(hashing)을 이용하며, 부등호 검사에는 균형 이진 검색 트리를 사용한다. 제안하는 방법은 해싱을 이용하므로, 등호 술어(Equality predicate)의 개수와 관계없이 등호 검사에 필요한 비용이 항상 일정하다. 이 외에도, 부등호 검사를 위한 검색 트리는 부등호 술어(Inequality predicate)만을 이용해 생성되므로 트리의 높이가 IBS-tree에 비해 적어진다. 이러한 특성은 IBS-tree에 비해 제안하는 방법의 검색 성능을 더욱

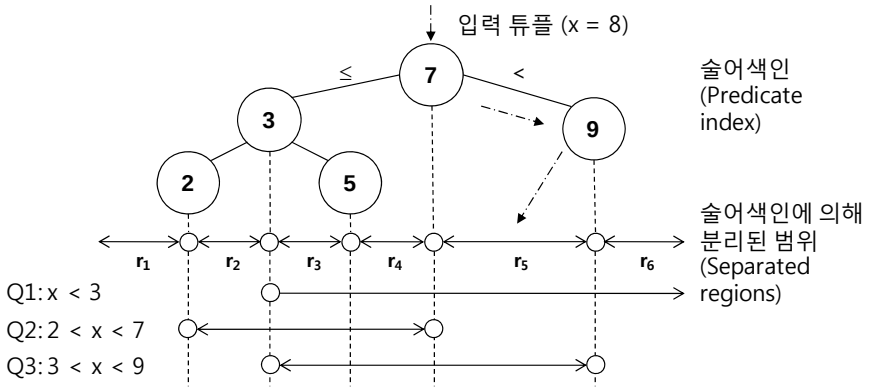


그림 1. 술어 색인을 이용한 범위 질의 처리

우수하게 만든다. 본 논문에서는 실험을 통해 IBS-tree와 제안하는 방법의 검색 성능을 비교하였으며, 실험 결과로부터 제안하는 방법의 성능이 더욱 우수한 것을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 술어 색인 방법과 관련한 기존연구에 대해 소개한다. 3장에서는 제안하는 색인 방법을 설명하고, 검색 비용 측면에서 IBS-tree와 비교한다. 4장에서는 실험을 통해 IBS-tree와 제안하는 색인 방법의 검색 성능을 비교한다. 마지막으로 5장에서는 결론을 맺는다.

2. 기존 연구

기존에 IBS-tree[2], IS-list(Interval skip list)[3], 그룹 필터(Grouped filter)[8], R-tree[7], Interval B+tree[9] 등 술어 색인과 관련된 많은 연구가 진행되었다. 그러나 이러한 방법 중 모두가 스트림 처리에 적합하지는 않다. 예를 들어, R-tree는 개방 범위(Open interval)를 지원하지 않으며, 겹치는 범위가 많을 수록 검색 성능이 급격히 저하된다[2]. 그리고 스트림 응용은 실시간 응답을 위해 메인 메모리 상에서 실행되도록 설계되는 반면, 기존의 많은 공간 색인 방법은 디스크 기반으로 설계되었다. 따라서 해당 색인 방법을 스트림 응용에 적용하는데는 무리가 있을 수 있다[4]. Interval B+tree의 경우 O(n^2)의 검색 성능을 제공하며, $O(n \log n)$을 제공하는 다른 색인 방법들에 비해 상대적으로 느리다고 할 수 있다 (n은 술어 개수를 의미함). 위 내용으로부터 이 장에서는 메인 메모리 기반 술어 색인에 해당하는 IBS-tree, IS-list, 그리고 Grouped filter만을 고려한다.

IBS-tree는 서로 겹치는 다수의 범위 질의를 효율적으로 처리하기 위한 균형 이진 검색 트리이다. 그림 2는 A부터 G까지 7개의 질의 술어를 처리하기 위한 IBS-tree의 예를 보여준다. IBS-tree에서 하나의 노드는 4개의 슬롯을 포함한다. 제일 상단의 슬롯은 부등호 술어의 종단점이나 등호 술어의 상수 값을 나타낸다. 이를 편의상 "술어 상수"라 부른다. 아래의 3개의 슬롯은 술어 상수보다 입력 튜플의 값이 각각 작거나(smaller-than), 같거나(equal), 클(greater-than) 때 만족하는 술어들의 집합을 나타낸다.

IBS-tree에서 등호 검사가 성공하면 검색은 더 이상 진행되지 않는다. 예를 들어, 그림 2에서 입력 튜플 3이 들어왔다고 하면, 검색은 2번째 레벨(second-level)의

술어 ID	술어
A	x ≤ 6
B	2 < x
C	x < 9
D	x = 3
E	x = 5
F	x = 7
G	x = 8

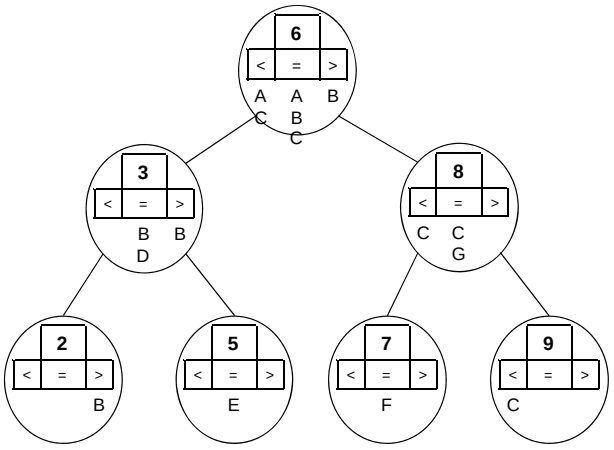
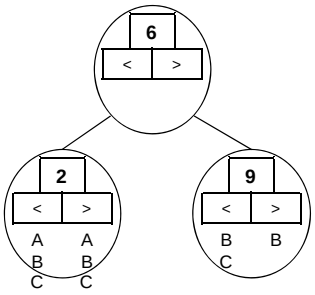


그림 2. IBS-tree의 예

술어 ID	술어
A	x≤ 6
B	2< x
C	x< 9
D	x= 3
E	x= 5
F	x= 7
G	x= 8

술어상수	술어 ID
2	A,C
3	A,B,C,D
5	A,B,C,E
6	A,B,C
7	B,C,F
8	B,C,G
9	B

(1) 균등 검사를 위한 해시 테이블



(2) 부균등 검사를 위한 균형 이진 검색 트리

그림 3. 제안하는 술어 색인의 예

노드 3에서 멈춘다. 그리고 결과는 A, B, C, 그리고 D가 된다. 이 때 A와 C는 루트 노드 6에서, 그리고 B와 D는 노드 3에서 비롯된다.

술어를 삽입하거나 삭제할 때마다 IBS-tree는 트리 구조의 균형을 바로잡기 위한 복잡한 과정의 재균형(Re-balance) 처리가 필요하다. 이러한 재균형 처리를 쉽게 하기 위해, 같은 저자는 이후에 IS-list를 제안하였다. 그리고 재균형 처리의 관점에서 IS-list가 IBS-tree보다 나은 성능을 제공한다는 점을 보였다. 그러나 검색 성능에서는 두 방법이 동일한 성능(O(log n))을 보인다. 현재 본 논문에서는 검색 성능에 중점을 두고 있으므로, 편의상 두 방법 중 IBS-tree만을 고려한다.

그룹 필터는 4개의 분리된 구조를 사용한다. '>'(Greater-than test)나 '<' (Smaller-than test)를 지닌 술어를 처리하기 위해 두 개의 균형 이진 검색 트리를 이용한다. 그리고 '=' (Equality test)나 '!=' (Non-equality test)를 지닌 술어를 처리하기 위해 두 개의 해시 테이블을 이용한다. 이러한 구조의 분리는 술어의 동적 삽입 및 삭제를 용이하게 한다. 그러나 튜플이 들어올 때마다 모든 구조를 체크해야 하므로 검색 성능은 저하된다.

그룹 필터는 등호 검사를 위해 해시 테이블을 이용한다는 점에서 본 논문에서 제안하는 방식과 비슷하다. 그러나 두 가지 관점에서 차이가 있다고 할 수 있다. 첫째로 그룹 필터는 '>'나 '<'를 포함한 부등호 검사를 두 개의 트리로 분리하여 처리하는 반면, 제안하는 방식에서는 오직 하나의 트리를 이용한다. 다음으로 그룹 필터는 등호 검사가 성공해도 검색을 계속 진행하지만, 제안하는 방식에선 검색을 멈춘다. 이 두 가지 차이로부터 제안하는 방식이 그룹 필터보다 더 좋은 검색 성능을 보인다는 것은 명확하다. 따라서 다음 장 이후에서는 IBS-tree만을 기준으로 제안하는 방식의 특성과 성능에 대해 언급한다.

3. 제안하는 술어 색인 방식

본 논문에서 제안하는 술어 색인 방식은 등호 검사와 부등호 검사를 분리하여 처리함으로써 검색 성능을 향상시킨다. 등호 검사를 위해 해시 테이블을 이용하고 부등호 검사를 위해 균형 이진 검색 트리를 사용한다. 그림 3은 그림 2에서 주어진 술어로부터 생성된 술어 색인의 예에 해당한다. 하나의 해시 엔트리(Hash entry)는 각 술어 상수에 해당하는 술어들의 집합을 가진다. 그리고 검색 트리의 한 노드는 3개의 슬롯으로 구성되어 있다. 제일 상단의 슬롯은 부등호 술어의 종단점의 값을 나타낸다. 아래의 2개의 슬롯은 각각 '<' 및 '>'을 만족시키는 술어들의 집합을 나타낸다.

하나의 튜플이 들어오면 제안하는 방식에서는 먼저 해시 테이블을 검색한다. 만약 튜플 값에 대한 해시 엔트리를 찾으면, 검색을 멈추고 엔트리 내의 술어들을 결과로 반환한다. 그렇지 않은 경우, 검색 트리를 체크한다. 예를 들어, 그림 3에서 입력 튜플 5가 들어왔다고 가정하자. 이 경우 먼저 해시 테이블을 검색한다. 해시 테이블엔 튜플 5에 대한 엔트리가 존재하며, 엔트리에는 술어 A, B, C, E가 존재한다. 따라서 검색을 멈추고 4개의 술어를 결과로 반환한다. 또 다른 예제로 튜블 4가 들어온 경우를 생각해 보자. 이 경우 해시 테이블에 튜플 4를 위한 엔트리가 존재하지 않는다. 따라서 검색 트리를 체크하게 되고, 트리의 리프 노드(leaf node) 2로부터 결과로서 A, B, C를 반환하게 된다.

제안하는 방식에서 검색 트리는 오직 부등호 술어들로부터 생성된다. 반면에 해시 테이블은 등호 술어 뿐 아니라 부등호 술어들까지 포함하여 만들어진다. 만약 부등호 술어에 해당하는 엔트리가 해시 테이블에 없다면 결과가 부정확할 수 있다. 예를 들어, 그림 3에서 술어 B의 상수 2를 위한 엔트리가 해시 테이블에 존재하지 않는다고 가정하자. 이 경우 입력 튜플 2가 들어왔을 때, 검색은 리프 노드 2에서 종료되며, 결과로는 술어 A, B, C가 반환될 것이다. 하지만, B는 $x < 2$로 정의

되어 있으므로 결과에서 제외되어야 한다. 아래의 정리는 제안하는 방법에서 해시 테이블이 모든 술어들에 대한 엔트리를 포함하고 있어야 함을 보여준다.

정리 1. 제안하는 술어 색인 방식에서 검색 트리에 n개의 노드가 있다면, 정확한 결과를 반환하기 위해 최소 n개의 해시 엔트리가 필요하다.

증명. i번째 노드를 v_i라 하자.

$$v_i < v_i + 1 \ (1 \le i \le n)$$

노드 수가 n일 때, 전체 입력값 범위는 그림 1처럼 $n+1$개의 범위로 나누어진다. i번째 범위 r_i는 아래와 같이 나타낼 수 있다.

$$r_i < v_i \ \ (i = 1)$$
$$v_i < r_i + 1 < v_i + 1 \ \ (1 \le I < n)$$
$$v_i < r_i + 1 \ (i = n)$$

여기서 r_i는 부등호 술어들로부터 만들어졌으므로, 모든 r_i가 v_i를 포함하지 않는다.
튜플 v_i가 주어지고, 해시 테이블에 v_i에 대한 엔트리가 없다고 가정하자. 이 경우 검색은 트리의 r_i에 도달할 것이다. 하지만 v_i는 r_i에 포함되지 않으므로 잘못된 결과가 발생한다. 이러한 경우를 방지하기 위해서는 v_i에 해당하는 엔트리가 해시 테이블에 존재해야 한다. □

이제 IBS-tree와 제안하는 방식의 검색 비용을 비교해 보도록 하겠다. 본 논문에서는 검색 성능을 측정하기 위해 두 종류의 검색 비용을 이용한다. 첫째는 트리 검색 비용이며, 둘째는 해시 비용이다. 전자는 검색 트리의 깊이와 트리의 한 노드를 접근하고 처리하는데 걸리는 비용의 곱으로 나타낼 수 있다. 후자는 하나의 상수로 나타낼 수 있다. 아래에서 검색 비용을 논의할 때 표 1에서 정의된 기호를 사용한다.

IBS-tree의 검색 비용은 트리 검색 비용으로 나타낼 수 있다. 트리는 모든 술어로부터 만들어지므로 높이는 $log(n+m)$으로 나타낼 수 있다. 그리고 IBS-tree는 하나의 입력 튜플에 대해 노드마다 '=' 및 '<' 또는 '>'을 포함한 최소 두 번의 비교를 수행한다. 비교 회수를 반영한 단위 비용을 α라 하면, IBS-tree의 검색 비용 C_{ibs}는 $\alpha\, log(n+m)$으로 나타낼 수 있다.

정확히 C_{ibs}를 계산하기 위해서는 등호 검사가 성공할 경우 검색이 멈추는 것을 함께 고려해야 한다. 그러나 다음 장의 실험 결과에 의하면 등호 검사에 의해 검색이 중간에 멈추는 것이 성능에 크게 영향을 주지 않는 것으로 나타났다. 이로부터, C_{ibs}에는 등호 검사에 의한 검색 멈춤을 고려하지 않는다.

Symbol	Description
n	부등호 술어의 수 (e.g. $x > 1$ or $x < 1$)
m	등호 술어의 수 (e.g. $x = 1$)
α	IBS-tree에서 하나의 노드를 처리하기 위한 단위 비용
β	제안하는 방법에서 하나의 노드를 처리하기 위한 단위 비용
h	해쉬 비용
r	입력 값의 도메인의 크기 (e.g. [11, 100]일 경우 90)

표 1. 본 논문에서 이용하는 기호에 대한 설명

제안하는 방식의 검색 비용은 트리 검색 비용과 해싱 비용의 합으로 나타낼 수 있다. 제안하는 방식에서 검색 트리는 오직 부등호 술어들로부터 생성되므로 검색 트리의 높이는 $log\, n$으로 나타낼 수 있다. 트리는 하나의 입력 튜플에 대해 노드마다 '>' 또는 '<'를 포함한 한번의 비교를 수행한다. 비교 회수를 반영한 단위 비용을 β라 하면, 제안하는 방식의 검색 비용 C_{our}는 $\beta\, log\, n+h$로 나타낼 수 있다.

해싱은 일반적으로 해싱 함수의 복잡한 계산 과정과 충돌 처리(Collision control) 과정으로부터 큰 비용을 가진다. 이 비용을 줄이기 위해, 제안하는 방식에서는 입력 튜플 값의 범위를 포함하는 충분한 크기의 배열을 해시 테이블로 이용한다. 예를 들어, 경매 응용(Auction application)에서 사용자 질의의 대상 가격이 1000원부터 100,000,000원 사이에 있을 것이라고 예측할 수 있다. 이러한 경우 100,000개의 요소를 지닌 배열을 이용하여 해시를 처리할 수 있다. 이 때 공간 사용 효율(Space utilization)을 높이기 위해, 입력 값의 분포에 따라 배열 크기를 동적으로 증가시키거나 감소시키는 확장 해싱(extensible hashing)[5] 방법을 이용할 수도 있다. 다음 장의 실험 결과에 의하면 배열 색인(Array indexing) 비용이 검색 성능에 미치는 영향은 미미한 것으로 나타났다. 이러한 실험 결과로부터 C_{our}는 $\beta\, log\, n$으로 간략히 표현될 수 있다.

위에서 얻어진 C_{ibs}와 C_{our}를 바탕으로 IBS-tree와 제안하는 방식의 성능 차이를 (1)을 이용하여 추정할 수 있다. 위에서도 언급하였듯이, α가 β보다 더욱 큰 값을 지니므로 실제 성능 차이는 추정된 차이보다 커진다 (이와 관련하여 다음 장에서 실험 결과를 제시하고 있다).

$$C_{ibs}/C_{our} \approx log(n+m) \,/\, log\, n \qquad \ldots (1)$$

실험 ID	r	n_g	n_l	n_e	튜플 개수
1	가변	512		512	1 M
2	100	가변		0	1 M
3	100	가변		가변	1 M
4	100	0 ~ 1024	$1024 - n_g$	0	1 M

표 2. 실험을 위한 파라미터 설정값

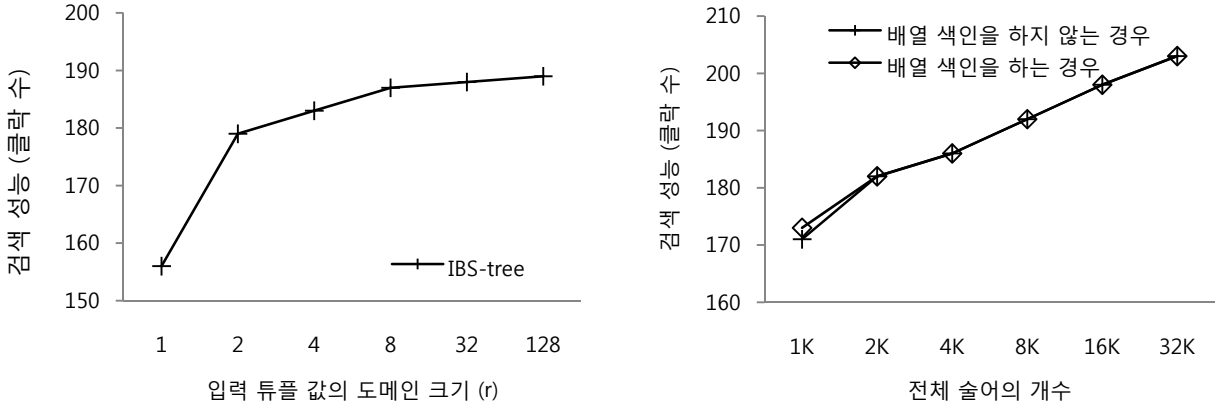


그림 4. (a) 등호 검사가 IBS-tree의 검색 성능에 주는 영향 (왼쪽) 및
(b) 배열 색인이 제안하는 방식의 검색 성능에 주는 영향 (오른쪽)

4. 실험 결과

이 장에서는 IBS-tree와 제안하는 술어 색인 방식의 성능을 실험을 통해 비교한다. 실험을 위해 두 방식의 알고리즘을 구현하였으며, 실험을 위한 파라미터 r, n, m이 주어졌을 때 데이터 집합을 생성해 주는 데이터 생성기 역시 구현하였다. 표 2는 실험을 위한 파라미터 설정을 보여준다. 표에서 r은 전체 술어 개수에 대한 배수로 나타내고 있다. 실험은 인텔 펜티엄 IV 2.4GHz의 CPU와 1G의 메인 메모리를 가지는 MS Windows XP 상에서 이루어졌다.

실험 1. 등호 검사가 IBS-tree의 검색 성능에 주는 영향:
실험 1은 IBS-tree의 등호 검사에서 일어나는 검색 멈춤이 검색 성능에 얼마나 큰 영향을 주는지 확인하고자 하였다. 등호 검사가 성공할 확률은 r의 값에 많은 영향을 받는다. 예를 들어, r이 증가하면 등호 검사가 성공할 확률은 감소한다. 따라서 본 실험에서는 변화하는 r의 값에 따른 결과를 관찰하였다. 그림 4 (a)는 실험 결과를 보여준다. 결과에서 볼 수 있듯이, r이 1에서 4로 변할 때는 검색 비용이 급격히 증가한다. 이에 반해 그 이외의 경우에 검색 비용은 일정하게 유지된다. 즉 r이 4보다 적은 극단적인 경우를 제외하고는 등호 검사에 의한 검색 멈춤이 검색 성능에 거의 영향을 주지 않는다는 것을 의미한다.

실험 2. 배열 색인 비용이 제안하는 방식의 검색 성능에 주는 영향:
실험 2에서는 제안하는 방식의 검색 비용에 대해 배열 색인을 사용한 경우와 사용하지 않은 경우 각각의 조건하에서 비교해 보았다. 즉, 검색 트리만 사용한 경우와 검색 트리와 배열을 함께 사용한 경우 검색 비용을 비교하였다. 그림 4 (b)는 실험 결과를 보여준다. 결과에서 알 수 있듯이, 두 경우의 비용은 거의 차이가 없다. 따라서, 배열 색인 비용이 검색 성능에 주는 영향은 미미하다고 할 수 있다.

실험 3. IBS-tree와 제안하는 방식의 검색 성능 비교:
실험 3에서는 IBS-tree와 제안하는 방식의 검색 성능을 비교하였다. 제안하는 방식의 검색 성능을 체크할 때 가장 나쁜 조건(m = 0)으로 설정한 후 실험을 수행하였다. 그림 5 (a)는 실험 결과를 보여준다. 실험 결과는 제안하는 방식의 검색 성능이 더 좋은 것으로 나타났다.

실험 4. IBS-tree와 제안하는 방식의 검색 성능 비율 차이:
마지막 실험은 두 방법의 검색 성능의 비율 차이를 비교하였다. 실험을 위해 등호 술어와 부등호 술어의 비율 ($m : n$)을 1024:0에서 1:31까지 바꿔가면서 변화를 관찰하였다. 그림 5(b)에서도 알 수 있듯이, 실험결과는 비율이 커질수록 성능 차이가 커진다는 것을 보여준다. 그리고 실험에 의한 차이 값을 (1)을 이용해 계산한 추

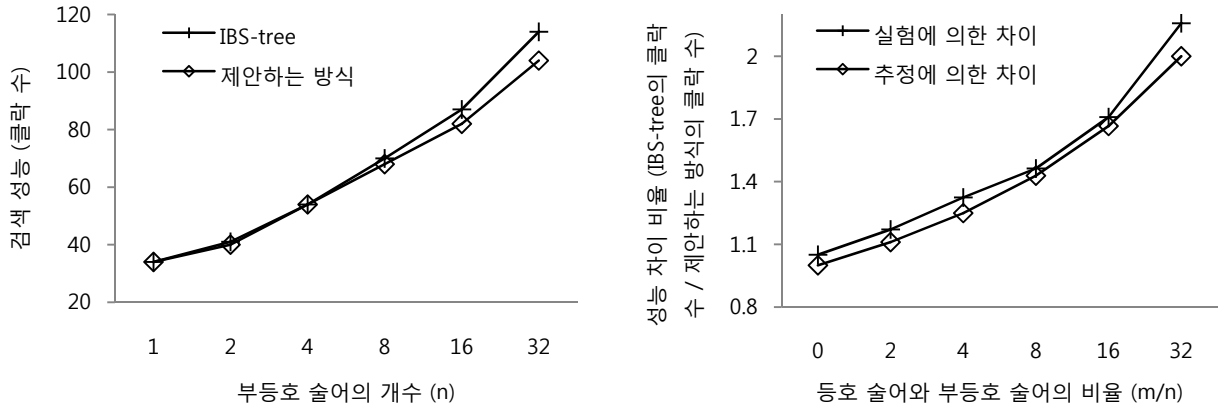


그림 5. (a) IBS-tree와 제안하는 방식의 검색 성능 차이 (왼쪽) 및
(b) IBS-tree와 제안하는 방식 간의 검색 성능에 대한 비율 (오른쪽)

정치와도 비교하였다. 실험 결과, 실험값이 추정치보다 크게 나타남을 확인할 수 있었으나 그 차이가 크지는 않았다. 이 차이는 지난 장에서 논의한 바와 같이 α와 β의 차이에서 기인한다고 할 수 있다.

5. 결 론

본 논문에서는 등호 검사와 부등호 검사를 분리한 술어 색인 방법을 제안하였다. 그리고 제안하는 방법을 스트림 환경에 가장 적합한 술어 색인 방법 중 하나인 IBS-tree와 비교하였다. IBS-tree와 비교할 때, 제안하는 방식에서는 등호 검사 비용은 항상 일정하며, 검색 트리의 높이도 적다. 이러한 특성으로부터 제안하는 방식이 IBS-tree보다 나은 검색 성능을 제공한다. 본 논문에서는 실험을 통해 IBS-tree와 제안하는 방법의 검색 성능을 비교하였으며, 실험 결과 제안하는 방법의 성능이 더욱 우수한 것을 확인하였다.

6. 참고 문헌

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, Models and Issues in Data Stream Systems, Proc. of ACM PODS 2002, Madison, Wisconsin, United States, 2002, pp.1-16.

[2] E. Hanson, M. Chaabouni, C. Kim, and Y. Wang, A Predicate Matching Algorithm for Database Rule Systems, Proc. of ACM SIGMOD 1990, pp.271-280.

[3] E. Hanson and T. Johnson, Selection Predicate Indexing for Active Database Using Interval Skip Lists, Information Systems 21 (3), (1996) 269-298.

[4] K. Wu, S. Chen, P. S. Yu, and M. Mei, Interval Query Indexing for Efficient Stream Processing, Proc. of ACM CIKM 2004, Washington DC, United States, November 2004, pp.88-97.

[5] L. Golab and M. T. Ozsu, Issues in Data Stream Management, ACM SIGMOD Record 32 (2), (2003) 5-14.

[6] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong, Extendible hashing - a fast access method for dynamic files, ACM TODS 4 (3), (1979) 315-344.

[7] R. H. Guting, An Introduction to Spatial Database Systems, VLDB Journal 3, (1994) 357-399.

[8] S. R. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. Continuously Adaptive Continuous Queries over Streams, Proc. of ACM SIGMOD 2002, Madison, Wisconsin, United States, June 2002, pp.49-60.

[9] T. Bozkaya and M. Ozsoyoglu, Indexing Transaction Time Database, Information Sciences 112 (1998).