Journal of Korea Multimedia Society Vol. 13, No. 9, September 2010(pp. 1299-1313)

요약된 Partitioned-Layer Index: Partitioned-Layer Index의 임의 접근 횟수를 줄이는 Top-k 질의 처리 방법

허 준 석†

요 약

Top-k 질의는 데이터베이스에서 사용자가 가장 원하는 k개의 객체를 구하는 질의이다. Top-k 질의를 효율적으로 처리하는 대표적인 연구로 Partitioned-Layer Index (간단히, PL-index) 방법이 있다. PL-index는 데이터베이스를 여러 개의 더 작은 데이터베이스로 분할하고 각 분할된 데이터베이스에 대해 sublayer들의 list (간단히, sublayer list)를 구성한다. 이때, 분할된 데이터베이스에 대해서 top-i 결과가 될 수 있는 객체들을 그 분할된 데이터베이스에 대한 i번째 sublayer로 구성한다. 그리고 주어진 질의에 맞춰 그 sublayer list들을 병합함으로써 질의 결과를 구한다. PL-index는 질의 처리 시 데이터베이스로부터 읽어 들이는 객체의 개수가 매우 작다는 장점을 가지지만, sublayer list들을 병합할 때에 임의 접근(random access)이 많이 발생하기 때문에 디스크 기반의 데이터베이스 환경에서 질의 처리 성능이 저하된다. 이에 본 논문에서는 임의 접근 횟수를 줄임으로써 디스크 기반의 데이터베이스 환경에서 PL-index의 질의 처리 성능을 크게 향상시키는 요약된(Abstracted) Partitioned-Layer Index (간단히, APL-index)를 제안한다. 먼저, PL-index의 각 sublayer를 가상의 (점) 객체로 요약함으로써 sublayer list들을 이러한 점 객체들의 list들(즉, APL-index)로 변형한다. 그리고 APL-index에 대해 질의 처리를 가상으로 수행하여 실제 질의 처리 시 접근할 sublayer를 예측한다. 그리고 예측된 sublayer들을 sublayer list별로 한꺼번에 읽어 들임으로 PL-index에서 발생하는 임의 접근 횟수를 줄인다. 합성 데이터와 실제 데이터에 대한 실험을 통하여 제안한 APL-index가 PL-index의 임의 접근 횟수를 크게 줄일 수 있음을 보인다.

Abstracted Partitioned-Layer Index: A Top-k Query Processing Method Reducing the Number of Random Accesses of the Partitioned-Layer Index

Jun-Seok Heo†

ABSTRACT

Top-k queries return k objects that users most want in the database. The Partitioned-Layer Index (simply, the *PL-index*) is a representative method for processing the top-k queries efficiently. The PL-index partitions the database into a number of smaller databases, and then, for each partitioned database, constructs a list of sublayers over the partitioned database. Here, the i^{th} sublayer in the partitioned database has the objects that can be the top-i object in the partitioned one. To retrieve top k results, the PL-index merges the sublayer lists depending on the user's query. The PL-index has the advantage of reading a very small number of objects from the database when processing the queries. However, since many random accesses occur in merging the sublayer lists, query performance of the

※ 교신저자(Corresponding Author) : 허준석, 주소 : 대전광역시 유성구 과학로 335 KAIST 전산학과(305-701), 전화 : 042-350-8386, FAX : 042-350-8380, E-mail : jsheo@mozart.kaist.ac.kr
접수일 : 2010년 5월 25일, 수정일 : 2010년 9월 2일
완료일 : 2010년 9월 13일
† 정회원, 한국과학기술원 전산학과
※ 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었음(UD060048AD).

PL-index is not good in environments like disk-based databases. In this paper, we propose the *Abstracted Partitioned-Layer Index* (simply, the *APL-index*) that significantly improves the query performance of the PL-index in disk-based environments by reducing the number of random accesses. First, by abstracting each sublayer of the PL-index into a *virtual (point) object*, we transform the lists of sublayers into those of virtual objects (i.e., the APL-index). Then, we virtually process the given query by using the APL-index and, accordingly, predict sublayers that are to be read when actually processing the query. Next, we read the sublayers predicted from each sublayer list at a time. Accordingly, we reduce the number of random accesses that occur in the PL-index. Experimental results using synthetic and real data sets show that our APL-index proposed can significantly reduce the number of random accesses occurring in the PL-index.

Key words: Top-k query processing(Top-k 질의 처리), Partitioned-Layer Index, Random access(임의 접근)

1. 서 론

데이터의 양이 많아짐에 따라 사용자가 가장 원하는 k개의 결과를 구하는 top-k 질의가 점점 더 중요해 지고 있다. 일반적으로 top-k 질의는 데이터베이스의 객체들 중에서 점수가 가장 높은 (또는 가장 낮은) k개의 객체를 구하는 질의로 정의되며, 객체의 점수는 사용자가 관심을 가지는 속성(attribute)의 값을 결합함으로써 계산된다[1-3].

예를 들어, top-k 질의는 디지털 카메라 쇼핑몰 등에서 사용될 수 있다. 판매되는 디지털 카메라는 가격, 무게, 크기, 화소수, 센서 크기, ISO 감도 등을 속성으로 가지는 릴레이션으로 나타낼 수 있다. 가격이 저렴한 콤팩트형 디지털 카메라를 구매하기 원하는 사용자는 "가격", "무게", "크기"와 같은 속성들에 대해 높은 관심을 가질 것이다. 반면, 사진의 품질에 관심이 높은 사용자는 "화소수", "센서 크기", "ISO 감도"와 같은 속성들에 높은 관심을 가질 것이다. 즉, 사용자들은 저마다 자신이 가장 중요하게 생각하는 속성들에 대해 높은 가중치를 부여하고 그렇지 않은 속성들에 대해서는 낮은 가중치를 부여함으로써 디지털 카메라에 대한 점수를 계산한다. 그리고 자신이 가장 원하는 상위 k개의 디지털 카메라를 결과로서 구한다[4].

지금까지 이러한 top-k 질의를 효율적으로 처리하기 위한 연구들이 많이 있어 왔다. Top-k 질의를 처리하는 방법은 크게 다음 세 가지 접근법으로 구분된다[4,5]. 먼저, 릴레이션의 속성들이 서로 독립적이라고 간주하는 방법이다. 이 방법은 릴레이션의 각 속성에 대해 그 속성 값만을 고려하여 객체(즉, tuple)들을 정렬하고 이를 list로 저장한다. 그리고 각 속성별로 정렬된 list들을 필요한 만큼 병합함으로써 top-k 질의의 결과를 구한다. 이 방법은 질의 처리를 위해 속성별로 정렬된 list를 유지하기 때문에 "list 기반 방법"이라 불린다[4,5]. 이 방법의 대표적인 연구로는 TA[6]가 있다. List 기반 방법은 릴레이션이 속성별로 저장되는 분산 데이터베이스 환경에서 유용하게 사용되지만, 정렬된 list를 구성할 때, 다른 속성 값을 이용하지 않기 때문에 이러한 속성 값을 이용하는 다른 접근 방법들[7]에 비해 질의 처리 시 불필요한 객체를 많이 읽는 성능상의 손해가 있을 수 있다.

다음으로, list 기반 방법과 달리 모든 속성들의 값을 이용하여 객체들을 layer들의 list로 구성하는 방법이 제안되었다. 이 방법은 주어진 데이터베이스의 전체 객체들에 대해 top-1 결과가 될 수 있는 객체들을 선택하여 첫 번째 layer를 구성하고, 나머지 객체들로부터 top-2 결과가 될 수 있는 객체들을 선택하여 두 번째 layer를 구성한다. 이와 같은 방식으로 주어진 데이터베이스의 모든 객체들을 layer들의 list (간단히, layer list)로 구성한다. 이 방법은 주어진 데이터베이스를 여러 개의 layer들로 분할하고 질의 처리 시 필요한 만큼의 layer만을 읽기 때문에 "layer 기반 방법"이라 불린다[4,5]. Layer 기반 방법은 다시 데이터베이스를 한 개의 layer list로 구성하는 "single layer list" 방법과 여러 개의 layer list로 구성하는 "multiple layer list" 방법으로 세분화 구분된다. Single layer list 방법은 데이터베이스 전체에 대해서 layer list를 구성하고 질의 처리 시 첫 번째 layer부터 최대 k 번째 layer까지 순차적으로 읽음으로써 top-k 결과를 구한다. 따라서 이 방법은 순차 접근(sequential access) 비용이 임의 접근(random

access) 비용 보다 저렴한 디스크 기반의 데이터베이스 환경에서 유용하다. 그러나 이 방법은 데이터베이스의 차원이 증가할수록 layer에 속하는 객체들의 개수가 많아져서 질의 처리 성능이 저하되는 문제를 가지고 있다. 이 방법의 대표적인 연구로는 ONION [8], AppRI[7]가 있다.

Multiple layer list 방법은 데이터베이스를 여러 개의 subregion으로 분할하고 각 subregion에 대해서 layer list (간단히, sublayer list)를 구성하고 질의 처리시 그 list들을 질의에 따라 동적으로 선택하여 병합함으로써 top-k 결과를 구한다. 이 방법은 layer에 속하는 객체의 개수를 줄임으로써 질의 처리 시 데이터베이스로부터 읽어 들이는 객체의 개수를 크게 줄인다. 그러나 질의 처리 시 여러 sublayer list들을 옮겨 다니며 sublayer를 읽어 들이기 때문에 잦은 임의 접근이 발생하고, 그 결과 disk 기반의 데이터베이스 환경에서 이로 인한 성능저하가 있을 수 있다. 이 방법의 대표적인 연구로는 PL-index[5]가 있다.

다음으로, 몇 개의 top-k 질의에 대해 질의 결과를 구하고 이를 실체화된(materialized) view로 활용하는 "view 기반 방법"이 제안되었다. 이 방법은 질의 처리 시 주어진 질의와 가장 유사한 질의에 의해 생성된 view를 선택하여 그 view의 앞부분부터 순차적으로 top-k 결과를 보장할 수 있는 만큼 객체들을 읽음으로써 질의 결과를 구한다. 이 방법의 대표적인 연구로는 PREFER[1], LPTA[9]가 있다. 이 방법은 질의와 유사한 view가 존재할 경우, 질의 처리 성능이 좋지만 그렇지 않은 경우, 질의 처리 성능이 매우 나쁠 수 있다. 따라서 view의 개수를 늘림으로써 질의 처리 성능을 향상시킬 수 있지만, 그럴 경우 이러한 view를 저장하고 관리하는 오버헤드가 커지게 된다[10].

이외에도 skyline을 효율적으로 계산하는 연구[11,12]와 전통적인 top-k query를 sensor 데이터와 같은 부정확한 데이터를 지원하도록 확장하는 연구[13,14]가 있다. 특히, skyline에는 단조함수(monotone function)를 최소화하는 객체(즉, top-1 tuple)가 적어도 한 개 존재하기 때문에[15] skyline 연구는 top k개의 결과를 구하도록 확장될 수 있다[5].

본 논문에서는 디스크 기반 데이터베이스 환경에서 PL-index의 top-k 질의 처리 성능을 향상시키는 요약된(Abstracted) Partitioned-Layer Index(간단히, APL-index)를 제안한다. APL-index는 PL-index를 주기억 장치에 상주시킬 수 있도록 요약한 색인으로 질의 처리시 PL-index와 함께 사용하여 PL-index의 단점인 잦은 임의 접근 횟수를 줄이고, PL-index가 다른 기존 방법들에 비해 가지는 장점을 거의 손실 없이 보존한다. 본 논문의 공헌은 다음과 같다.

- PL-index의 임의 접근 횟수를 줄이는 APL-index를 제안한다. 먼저, PL-index의 각 sublayer를 가상의 점 객체로 근사함으로써 PL-index의 sublayer list들을 이러한 가상의 점 객체들의 list (즉, APL-index)로 변환한다. APL-index를 사용하여 질의 처리에서 필요한 sublayer를 예측한다.
- APL-index와 PL-index를 사용하여 임의 접근 횟수를 줄이면서 정확한 top-k 결과를 구하는 알고리즘을 제시한다. APL-index를 사용하여 질의 처리를 가상으로 수행하여 필요한 sublayer를 예측한다. 그리고 PL-index를 이용하여 예측된 sublayer를 sublayer list별로 한꺼번에 읽어 들이고, 정확한 top-k 결과를 보장할 수 있도록 특정 sublayer를 추가로 읽어 들인다.
- 합성 데이터 및 실제 데이터에 대한 실험을 통해 APL-index가 PL-index의 임의 접근 횟수를 크게 줄임을 보인다. 또한, APL-index를 통해 잘못 예측된 sublayer를 읽음으로 인한 질의 처리 오버헤드가 매우 작음을 보인다.

본 논문의 구성은 다음과 같다. 제 2절에서는 본 논문과 관련이 있는 최근 연구들에 대해 살펴본다. 제 3절에서는 본 논문에서 다루는 문제를 정형적으로 정의하고 APL-index를 구성하는 방법을 제안한다. 제 4절에서는 APL-index를 이용한 질의 처리 방법을 제안한다. 제 5절에서는 APL-index의 성능 평가 결과를 제시한다. 제 6절에서는 결론을 내린다.

2. 관련 연구

지금까지 데이터베이스의 일부만을 읽음으로써 top-k 질의 결과를 구하는 여러 방법들이 제안되었다. 본 절에서는 top-k 질의 처리 방법들 중 본 논문과 직접적으로 관련이 있는 layer 기반 방법들에 대해 설명한다. 세부적으로 제 2.1절에서 single list 기반 방법을 설명하고, 제 2.2절에서 multiple list 기반

방법을 설명한다.

2.1 Single layer list 기반 방법

Single layer list 방법은 데이터베이스 전체에 대해서 top-i 결과가 될 수 있는 객체들로 i번째 layer를 구성한다. 따라서 최대 k개의 layer를 읽음으로써 top k 결과를 구할 수 있다. 대표적인 방법으로는 ONION[8]과 AppRI[7]가 있다. ONION 방법은 다차원 공간상의 점으로 표현된 객체(즉, tuple)들의 집합에 대해 구한 convex hull[16]의 정점(vertex)들로 layer들을 구성한다. 즉, 첫 번째 layer는 전체 객체들의 집합에 대해 구한 convex hull의 정점들로 구성하고, 두 번째 layer는 나머지 객체들의 집합에 대해 구한 convex hull의 정점들로 구성한다. 이와 같은 방식으로 layer들을 구성한다. 결과적으로 바깥쪽 layer가 안쪽 layer를 "양파(onion)"과 같이 기하학적으로 둘러싸게 된다. 또한, 주어진 객체들의 집합에 대해 구한 convex hull의 정점들 중에는 임의의 선형 함수(linear function)를 최소화하는 객체와 최대화하는 객체가 적어도 한 개 이상 존재하기 때문에[7] ONION 방법은 속성에 대한 양(positive)의 가중치만을 가지는 top-k 질의 (간단히, 단조 선형 질의 (monotone linear query))뿐만 아니라 음(negative)의 가중치를 가지는 질의(간단히, 비단조 선형 질의 (non-monotone linear query))도 지원할 수 있다는 장점을 가진다. 그러나 layer의 크기(즉, layer에 속하는 tuple의 개수)가 크기 때문에 질의 처리 성능이 좋지 않다는 단점을 가진다. AppRI 방법은 단조 선형 질의만을 지원한다고 가정하고 단조 함수를 위한 skyline의 "domination relation"[7]을 단조 선형 함수를 위한 것으로 제약함으로써 ONION에 비해 더욱 작은 크기의 layer를 구성하였다.

2.2 Multiple layer list 기반 방법

Multiple layer list 방법은 single layer list 방법과 달리 layer의 크기를 줄이기 위해 전체 데이터베이스인 universe[5]를 여러 개의 더 작은 데이터베이스인 subregion[5]들로 분할하고 각 subregion에 대해 layer list (간단히, sublayer list)를 구성한다. 동일한 sublayer list에 속하는 다른 sublayer들 간에는 single layer list에서와 같이 순서가 존재하지만 서로 다른 sublayer list에 속하는 sublayer들 간에는 순서가 존재하지 않는다. 따라서 top-1 결과는 모든 sublayer list들로부터 첫 번째 sublayer들을 읽어서 구하고, top-2 결과부터는 특정 sublayer list로부터 최대 하나의 sublayer를 추가로 읽어서 구한다. 그 결과, 질의 처리 시 데이터베이스로부터 읽어 들이는 객체의 개수를 크게 줄인다. 그러나 이 방법은 여러 sublayer list들을 옮겨 다니며 sublayer들을 읽어 들이기 때문에 많은 임의 접근을 유발한다. 따라서 이러한 방법은 순차 접근 비용과 임의 접근 비용 간의 차이가 거의 없는 주기억 장치 또는 플래쉬 기반 데이터베이스 환경에서 질의 처리 성능이 우수할 것으로 예상된다[5]. 그러나 그 두 비용 간의 차이가 큰 디스크 기반 데이터베이스 환경에서 잦은 임의 접근으로 인해 질의 처리 성능이 저하될 것으로 예상된다. 대표적인 방법으로는 PL-index가 있다. PL-index는 AppRI와 마찬가지로 monotone linear query를 가정한다. PL-index는 그림 1에서 보이는 바와 같이 (1) universe를 여러 개의 subregion들로 partition하는 partitioning 단계와 (2) 각 subregion에 대해 layer list를 구성하는 layering 단계를 통해 구성된다[5].

구체적으로, partitioning 단계에서는 universe를 universe의 대각선과 평행한 subregion 여러 개로 partition한다. 이렇게 하는 이유는 전체 universe에

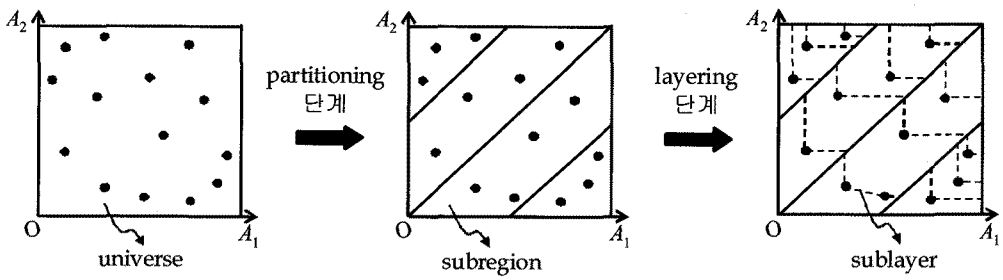
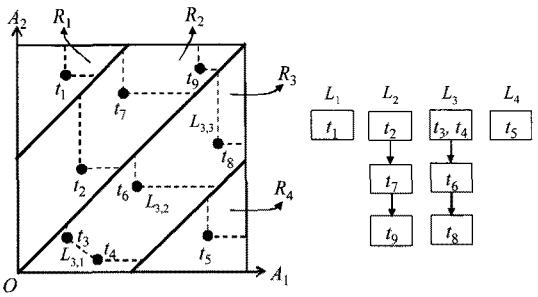


그림 1. 2차원 universe에서 PL-index를 구성하는 절차

대해 구한 layer의 크기보다 subregion에 대해 구한 layer의 크기가 더 작기 때문이다[5]. 다음으로, layering 단계에서는 skyline 보다 작은 크기의 "convex skyline"을 사용하여 subregion별로 layer list(즉, sublayer list)를 구성한다. 즉, subregion에 속하는 전체 객체들에 대해 convex skyline을 구하고 이를 첫 번째 sublayer로 저장한다. 그리고 subregion의 나머지 객체들에 대해 다시 convex skyline을 구하고 이를 두 번째 sublayer로 저장한다. 이와 같은 방식으로 sublayer list를 구성한다. 이렇게 구해진 sublayer list는 "optimally linearly ordered set 성질[8]"을 만족한다[5]. 즉, 임의의 monotone linear scoring function에 대해 i번째 layer에는 i번째 이후의 layer들에 속하는 모든 객체들보다 더 작거나 같은 질의 함수 값을 가지는 객체가 한 개 이상 존재한다. [5]에서 sublayer list의 이러한 성질을 아래의 보조정리 1로서 증명하였다.

보조정리 1: [5] Universe의 한 subregion R_i에 대해 구한 convex skyline들의 list를 $L_i = \{L_{i,1}, L_{i,2}, \cdots, L_{i,m}\}$이라 하자. 여기서, $L_{i,j}$는 R_i로부터 구한 j번째 convex skyline (즉, sublayer)를 나타낸다. ($1 \leq j \leq m$) 그리고 $o_{\min}(L_{i,j})$를 $L_{i,j}$에 속하는 객체들 중에서 임의의 단조 선형 함수 f에 대해 가장 작은 함수 값을 가지는 객체를 나타낸다고 하자. 그러면, $L_{i,j+1}, \cdots, L_{i,m}$에 속하는 어떠한 객체도 $o_{\min}(L_{i,j})$보다 더 작은 함수 값을 가지지 못한다.

예 1: [5] 그림 2는 2차원 universe에서 네 개의 sublayer list로 구성된 PL-index의 예이다. 그림 2(a)는 2차원 object t1, ···, t9이 존재하는 universe를 네 개의 subregion R_1, R_2, R_3, R_4로 분할하고 각 subregion에 대해 구한 sublayer들을 나타낸다. 예를 들어, R_3에 속하는 모든 객체들 (즉, t_3, t_4, t_6, t_8)에 대해 convex skyline을 구하고 이를 첫 번째 layer $L_{3,1} = \{t_3, t_4\}$ 로 간주한다. 그리고 $\{t_3, t_4\}$를 제외한 나머지 객체들 (즉, t_6, t_8)에 대해 convex skyline를 구하고 이를 두 번째 layer $L_{3,1} = \{t_6\}$ 로 간주한다. 이와 같은 방식으로 세 번째 layer $L_{3,3} = \{t_8\}$을 구한다. 그림 2(b)는 그림 2(a)에서 보인 subregion들에 대한 PL-index를 나타낸다. 여기서 L_1, L_2, L_3, L_4는 subregion R_1, R_2, R_3, R_4에 대해 구한 sublayer list를 각각 나타낸다.



(a) 네 개의 subregion에 대해 구성한 sublayer list들

(b) PL-index

그림 2. 2차원 universe에서 네 개의 sublayer list로 구성된 PL-index의 예[10]

3. Abstracted Partitioned-Layer Index (APL-index)의 구성

본 절에서는 top-k 질의 처리 시 PL-index의 임의 접근 횟수를 줄이기 위해 사용되는 APL-index를 구성하는 방법을 설명한다. 제 2절 관련 연구에서 설명했듯이 PL-index는 질의 결과를 구하기 위해 여러 sublayer list들을 여러 번 옮겨 다니며 sublayer를 하나씩 읽기 때문에 sublayer를 읽을 때마다 임의 접근이 발생한다. 이러한 임의 접근을 줄이기 위해 sublayer list별로 앞으로 읽을 sublayer들을 예측하고 이를 한꺼번에 읽어 들일 필요가 있다. 이를 위해 PL-index를 주기억 장치에 상주시킬 수 있도록 작은 크기의 색인(즉, APL-index)으로 요약하고, 주어진 질의를 요약된 색인만을 사용하여 가상으로 처리한다. 가상의 질의 처리를 통해 실제 질의 처리에서 필요한 sublayer들을 예측한다. APL-index를 이용한 질의 처리 방법은 제 4장에서 자세히 설명한다. 먼저, 제 3.1절에서 본 논문에서 다루는 top-k 질의 처리의 문제를 정형적으로 정의하고, 제 3.2절에서 제 2절에서 설명한 PL-index를 기반으로 APL-index를 구성하는 방법에 대해 자세히 설명한다.

3.1 문제 정의

본 절에서는 top-k 질의 처리의 문제를 정형적으로 정의한다. 질의 대상인 릴레이션 R은 실수 값을 가지는 d개의 속성 A_1, A_2, ..., A_d로 구성되어 있고 R에 속하는 tuple들의 개수는 N이다. 이러한 R의 각 tuple은 d차원 상의 점 객체 (간단히, 객체)로 간주할

수 있다. 이때, 모든 객체들이 $[0.0, 1.0]^d$ 범위의 d차원 공간 안에 존재한다고 가정한다. 이후부터는 이러한 $[0.0, 1.0]^d$ 범위의 d차원 공간을 "universe"라고 부른다. 또한, R의 tuple t와 universe상에서 그에 대응되는 객체 t를 같은 의미로 사용한다. Top-k 질의 Q는 단조 선형 함수 $f(t) = w[1]*t[1] + ... + w[d]*t[d]$(간단히, 질의 함수)와 결과 tuple들의 개수 k의 쌍(pair)인 $f(t)$, k로 정의된다. $f(t)$에서 $w[i]$는 i번째 속성에 대한 가중치를 나타내고, $t[i]$는 tuple t의 i번째 속성 값을 나타낸다. 본 논문에서는 대부분의 관련 연구[7,12, 18]와 같이 가중치 $w[i]$ 가 모두 양수 (즉, $w[i] \geq 0$ $(1 \leq i \leq d)$)이고, $w[1] + ... + w[d] = 1$이 되도록 [0, 1] 범위 안에 정규화(normalize) 되어있다고 가정한다. Top-k 질의의 결과로는 질의 함수의 값 $f(t)$이 가장 작은 k개의 tuple을 구한다. 즉, $f(t^1) \leq f(t^2) \leq ... \leq f(t^k) \leq f(t^l)$ $(k+1 \leq l \leq N)$를 만족하는 릴레이션 R의 tuple들의 시퀀스 $[t^1, t^2, ..., t^k]$를 구한다. t^j는 질의 함수에 대한 tuple들의 함수 값을 오름차순으로 정렬했을 때 j번째 위치에 있는 tuple을 나타낸다$(1 \leq j \leq N)$. 본 논문에서 사용하는 주요 표기법들을 표 1에서 요약한다. 본 절에서 아직 설명하지 않은 기호들은 제 3.2절과 제 4절에서 설명한다.

표 1. 본 논문에서 사용하는 주요 표기법들

기호	정 의
R	질의 대상이 되는 릴레이션
N	R에 속한 tuple의 총 개수
d	R을 구성하는 속성의 총 개수
A_i	R의 i번째 속성 $(1 \leq i \leq d)$
t	R의 한 tuple $(t \in R)$
$t[i]$	t의 A_i 값
$w[i]$	A_i에 대한 가중치
L_i	식별 번호가 i인 sublayer list
$L_{i,j}$	L_i에서 j번째에 위치하는 sublayer
aL_i	L_i에 대응되는 요약된 sublayer list
$aL_{i,j}$	aL_i에서 j번째에 위치하는 요약된 sublayer
$o_{\min}(S)$	객체들의 집합 S에서 질의 함수에 대한 값이 가장 작은 객체

3.2 APL-index의 구성

APL-index는 제 2.2절에서 설명한 PL-index를 요약하여 구성된다. PL-index는 질의 처리 시 sublayer 단위로 실제 객체들을 데이터베이스로부터 읽어 들여서 top k 개의 결과 객체들을 구하기 위해 사용된다. 그러나 APL-index는 질의 결과를 구하는 것이 아니라, 질의 결과를 구하기 위해 필요한 sublayer들을 예측하기 위해 사용된다. 따라서 APL-index는 PL-index의 sublayer와 같이 sublayer를 구성하는 실제 객체들을 유지할 필요가 없다. 대신, 주어진 질의에 대해 특정 sublayer에 포함된 결과 객체들의 개수를 예측하기 위해 각 sublayer에 속하는 객체들을 대표하는 가상의 객체(간단히, 대표 객체)와 그 sublayer에 속하는 객체들의 개수(즉, sublayer 크기)를 유지한다.

대표 객체는 sublayer에 속하는 실제 객체들이 주어진 질의 함수에 대해 가질 수 있는 대략적인 함수 값을 계산하기 위해 사용되며, 그 함수 값에 따라 실제 질의 처리에서 필요한 sublayer들을 예측한다(제 4절 참조). 본 논문에서는 PL-index의 각 sublayer를 그 sublayer에 속하는 모든 object들을 둘러싸는 최소 사각형(Minimum Bounding Rectangle(MBR))의 중심점을 그 sublayer의 대표 객체로 간주한다. 이는 임의의 단조 선형 함수 f에 대해 그 sublayer에 속하는 객체들이 가질 수 있는 함 수 값들에 대한 평균을 그 sublayer의 대표 객체에 대한 함수 값으로써 대략적으로 구할 수 있기 때문이다. 즉, MBR의 정의와 단조 선형 함수의 정의[1]에 의해 sublayer에 속하는 어떤 객체의 함수 값도 그 sublayer의 MBR의 좌하 점보다 크거나 같고, 우상점보다 작거나 같다. 따라서 sublayer에 속하는 객체들이 질의 함수에 대해 가질 수 있는 대략적인 평균 함수 값은 그 sublayer의 MBR의 좌하 점과 우상 점의 함수 값들의 평균, 즉, 그 MBR의 중앙 점의 함수 값과 같다.

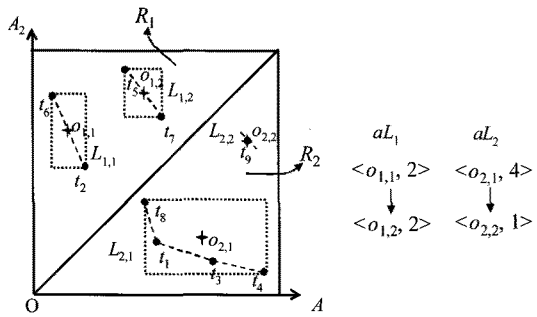
정리하면, PL-index의 각 sublayer는 그 sublayer의 대표 객체와 그 sublayer의 크기의 쌍(간단히, "요약된 sublayer"로 부름)으로 요약된다. 따라서 APL-index는 이러한 "요약된 sublayer"의 list들로 구성된다. 정의 1에서 이를 정형적으로 정의한다.

정의 1: (요약된 sublayer) PL-index의 sublayer $L_{i,j}$가 주어졌다고 하자. 그리고 $L_{i,j}$에 속하는 모든 객

1) [8] 만약 $\forall_{i=1...d}\, t[i] \leq t'[i]$ 이면 $f(t) \leq f(t')$이다.

체들을 둘러싸는 가장 작은 사각형 (즉, minimum bounding rectangle(MBR))의 중심점을 $o_{i,j}$라고 하고 $L_{i,j}$에 속하는 객체들의 개수 (간단히, 요약된 sublayer 크기)를 $|L_{i,j}|$이라 하자. (단, $L_{i,j}$에 한 개의 객체 t만 존재한다면(즉, $|L_{i,j}| = 1$), $L_{i,j}$의 MBR은 t와 같고 $t = o_{i,j}$이다) 이때, $L_{i,j}$에 대해 "요약된 sublayer" $aL_{i,j}$는 $<o_{i,j}, |L_{i,j}|>$으로 정의된다.

예 2: 그림 3은 2차원 universe에서 두 개의 sublayer list로 구성된 PL-index를 APL-index로 요약한 예이다. 그림 3(a)는 2차원 객체 t_1, …, t_9이 존재하는 universe를 두 개의 subregion R_1, R_2로 분할하고 각 subregion에 대해 구한 sublayer들을 나타낸다. R_1에 대한 sublayer list L_1은 두 개의 sublayer $L_{1,1} = \{t_2, t_6\}$, $L_{1,2} = \{t_5, t_7\}$으로 구성된다. 그리고R_2에 대한 sublayer list L_2는 두 개의 sublayer $L_{2,1} = \{t_1, t_3, t_4, t_8\}$, $L_{2,2} = \{t_9\}$으로 구성된다. 그림 3(b)는 그림 3(a)에서 보인 sublayer list로부터 요약된 sublayer list를 구한 예이고, 점선으로 표시된 사각형은 sublayer를 둘러싸는 MBR을 나타낸다. 예를 들어, sublayer $L_{2,1}$을 구성하는 모든 객체들에 대해 MBR을 구한다. 그리고 그 MBR의 중심점을 $L_{2,1}$의 대표 객체 $o_{2,1}$으로 구한다. $L_{2,1}$에는 4개의 객체가 속하므로 $|L_{2,1}|$는 4이다. 따라서 $L_{2,1}$에 대한 요약된 sublayer $aL_{2,1}$은 $<o_{2,1}, 4>$이다. 또한, $L_{2,2}$는 한 개의 객체만을 포함하고 있기 때문에 $o_{2,2}$와 t_9이 같고, $|L_{2,2}|$는 1이다. 따라서 $L_{2,2}$에 대한 요약된 sublayer $aL_{2,2} = <o_{2,2}, 1>$이다. 여기서 aL_1, aL_2는 R_1, R_2에 대해 구한 요약된 sublayer list를 각각 나타낸다.



(a) 두 개의 subregion들에 대해 구성한 sublayer list 들 (b) APL-index

그림 3. 2차원 universe에서 두 개의 sublayer list로 구성된 PL-index를 APL-index로 요약한 예

4. APL-index를 이용한 Top-k 질의 처리

APL-index를 이용한 top-k 질의 처리는 크게 "sublayer 예측 단계"와 "질의 수행 단계"로 구분된다. 먼저, sublayer 예측 단계에서는 APL-index를 사용하여 질의 처리 시 필요한 sublayer들을 예측한다. 그리고 질의 수행 단계에서는 PL-index를 이용하여 sublayer-list별로 예측된 sublayer를 한꺼번에 읽어 들이고 정확한 질의 결과를 구한다. 제 4.1절에서는 sublayer 예측 단계에 대해 설명하고, 제 4.2절에서는 질의 수행 단계에 대해 설명한다.

4.1 sublayer 예측 단계

APL-index는 그림 3(b)에서 보는 바와 같이 각 sublayer에 한 개의 객체만을 포함하고 있는 PL-index로 간주할 수 있다. 즉, 동일한 요약된 sublayer list에 속하는 요약된 sublayer들 간에는 순서가 존재하지만, 다른 요약된 sublayer list들에 속하는 서로 다른 요약된 sublayer들 간에는 순서가 존재하지 않는다. 따라서 요약된 sublayer list들의 앞부분부터 요약된 sublayer를 순차적으로 읽어 들이며 요약된 sublayer list들을 병합하여 가상으로 top k 결과를 구할 수 있다. 이 과정에서 읽혀진 요약된 sublayer들을 예측된 sublayer들로서 간주한다. APL-index를 이용한 가상의 top-k 질의 처리는 PL-index를 이용한 top-k 질의 처리와 매우 유사하다. 먼저, PL-index를 이용한 top-k 질의 처리를 설명하고 APL-index를 이용한 질의 처리와의 차이점을 설명한다.

PL-index를 이용한 top-k 질의 처리는 top-i ($1 \le i \le k$)의 i값을 기준으로 크게 (1) top-1 객체를 구하는 단계와 (2) top-i 객체 ($2 \le i \le k$)를 구하는 단계로 나뉜다[10]. Top-1 객체는 각 sublayer list의 첫 번째 sublayer를 읽음으로써 구한다. 이렇게 하는 이유는 보조정리 1에 의해 PL-index의 첫 번째 sublayer들 중에 top-1 객체가 존재하기 때문이다. Top-i 객체($2 \le i \le k$)부터는 특정 sublayer list로부터 최대 하나의 sublayer만을 읽음으로써 구한다. 구체적으로, top-(i-1) 객체가 속한 sublayer list에서 top-(i-1)이 속한 sublayer의 바로 다음 sublayer를 추가로 읽는다. 이는 보조정리 1에 의해 그 다음 sublayer에 top-i 객체가 존재할 수 있기 때문이다[5].

APL-index를 이용한 top-k 질의 처리가 PL-

index를 이용한 질의 처리와 다른 점은 top-i object를 구하기 위해 sublayer에 속하는 실제 object 대신 요약된 sublayer를 이용한다는 것이다. 구체적으로, PL-index를 이용한 질의 처리에서는 한 sublayer $L_{i,j}$에 속하는 객체들 중에서 다른 sublayer $L_{m,n}$에 속하는 $o_{\min}(L_{m,n})$보다 더 작은 함수 값을 가지는 객체들을 구하기 위해 그 두 sublayer에 속하는 객체들의 함수 값을 서로 비교한다. 그러나 APL-index를 이용한 가상의 질의 처리에서는 한 요약된 sublayer $aL_{i,j}$의 대표 객체 $o_{i,j}$가 다른 요약된 sublayer $aL_{m,n}$의 대표 객체 $o_{m,n}$보다 더 작은 함수 값을 가진다면 $L_{i,j}$에 속하는 객체들 중에서 사용자에 의해 주어진 λ 비율(적중률(hit ratio))만큼이 $o_{\min}(L_{m,n})$보다 더 작은 함수 값을 가진다고 간주한다.($0.0 < \lambda \leq 1.0$)

그림 4는 APL-index를 이용하여 sublayer를 예측하는 알고리즘을 나타낸다. 이 알고리즘은 c개의 요약된 sublayer list들로 구성된 APL-index, top-k 질의 $Q = (f(), k)$, 적중률 λ를 입력으로 받아, sublayer list별로 예측된 sublayer의 개수를 크기 c인 배열 P에 저장하여 결과로서 반환한다. 초기화 단계에서는 P의 모든 요소(element)의 값을 0으로 할당하고, 대표 객체들을 빠르게 정렬하기 위한 힙(heap) 자료구조를 초기화한다. 여기서, H의 루트(root)는 H에 존재하는 객체들 중에서 $f()$에 대해 가장 작은 함수 값을 가지는 객체를 가리킨다. 단계 1에서는 top-1 객체를 구한다. 먼저, 각 요약된 sublayer list aL_j의 첫 번째 요약된 sublayer $aL_{j,1}$ ($1 \leq j \leq c$)의 대표 객체 $o_{j,1}$를 H에 추가하고, 동시에 $P[j]$의 값을 1만큼 증가시킨다. 그리고 H의 루트가 가리키는 객체를 H에서 삭제한 후, top-1 객체로 구한다. 단계 2에서는 top-i 객체를 구한다. Top-(i-1)객체가 속한 요약된 sublayer $aL_{m,n}$을 찾는다($1 \leq m \leq c$). 즉, top-(i-1) 객체는 요약된 sublayer list aL_m에 속한다. aL_m에서 $aL_{m,n}$의 바로 다음 요약된 sublayer $aL_{m,n+1}$의 대표객

서브레이어 예측 알고리즘:

입력: (1) c개의 요약된 sublayer list $\{aL_1, aL_2, \ldots, aL_c\}$로 구성된 APL-index

(2) 질의 $Q = (f(), k)$

(3) 적중률 λ

출력: Sublayer list별로 예측된 sublayer 개수를 저장한 크기 c인 배열 P

알고리즘:

단계 0. 배열 P와 힙(heap) H를 초기화 한다.

단계 1. top-1 객체를 구하는 단계:

1.1 $1 \leq j \leq c$ 에 대해 다음을 수행한다.

1.1.1 $aL_{j,1}$의 대표 객체 $o_{j,1}$를 H에 추가한다.

1.1.2 $P[j] := P[j] + 1$ /* 예측된 sublayer 개수를 저장함 */

1.2 H의 루트가 가리키는 객체를 삭제한 후, top-1 객체로 구한다.

단계 2. top-i 객체를 구하는 단계: $2 \leq i \leq k$ 에 대해 다음을 수행한다.

2.1 만약 $i \geq k$이면 P를 반환하고 알고리즘을 끝낸다.

2.2 $aL_{m,n}$:= top-(i-1) 객체가 속한 요약된 sublayer

2.3 $aL_{m,n+1}$의 대표 객체 $o_{m,n+1}$를 H에 추가 한다.

2.4 $P[m] := P[m] + 1$

2.5 H의 루트가 가리키는 객체를 삭제한 후, top-i 객체로 구한다.

2.6 s := top-i 객체가 속한 요약된 sublayer의 크기

2.7 $i := i + s*\lambda$ /* 추가된 sublayer에 포함된 결과 객체의 개수를 반영함 */

그림 4. APL-index를 이용하여 sublayer를 예측하는 알고리즘

체를 H에 추가하고, $P[m]$의 값을 1만큼 증가시킨다. 그리고 H의 루트가 가리키는 객체를 H에서 삭제한 후, top-i 객체로 구한다. 이때, top-i 객체가 속한 요약된 sublayer의 크기를 s라 표기할 때, $s*\lambda$만큼의 질의 결과가 추가로 읽은 sublayer에 존재한다고 간주하고 지금까지 구한 결과 객체의 개수를 $i := i+s*\lambda$로 계산한다. 끝으로, 지금까지 구한 결과 객체의 개수 i가 k보다 크거나 같다면 P를 결과로서 반환하고 알고리즘을 끝낸다.

4.2 질의 수행 단계

질의 수행 단계에서는 top k 개의 결과 객체를 구한다. 먼저, 예측된 sublayer를 sublayer list별로 한꺼번에 읽어 들여서 임의 접근 횟수를 줄인다. 그리고 제 4.1절에서 설명한 PL-index의 질의 처리 방법과 같이 특정 sublayer-list로부터 추가로 sublayer를 읽어 들임으로써 정확한 top-k 질의 결과를 구한다.

먼저, 예측된 sublayer들로부터 읽어 들인 객체들의 집합을 H라고 정의할 때, top-1 객체는 H에서 가장 작은 함수 값을 가지는 객체와 같다. 이는 예측된 sublayer들을 한꺼번에 읽어 들일 때, 각 sublayer list별로 첫 번째 sublayer에 속하는 모든 객체들을 읽어 들였기 때문이다. 그리고 H에서 i번째로 가장 작은 함수 값을 가지는 객체 o가 top-i 객체라는 것을 보장하기 위해서는 아직 읽어 들이지 않은 어떤 객체도 o보다 더 작은 함수 값을 가지지 않아야 한다. 즉, 객체 o의 함수 값이 각 sublayer list로부터 가장 최근에 읽어 들인 sublayer들에 속하는 모든 객체들 보다 더 작거나 같아야 한다.

그림 5는 예측된 sublayer와 PL-index를 이용하여 top-k 질의 결과를 구하는 알고리즘을 나타낸다. 이 알고리즘은 c개의 sublayer list로 구성된 PL-index, sublayer list별로 예측된 sublayer의 개수가 저장된 크기 c인 배열 P, top-k 질의 $Q = (f(), k)$를 입력으로 받아, 결과로서 $f()$에 대해 함수 값이 가장 작은 k개의 객체를 결과로서 반환한다. 초기화 단계에서는 크기 c인 배열 A의 모든 요소(element)와 집합 H를 초기화한다. 단계 1에서는 sublayer list별로 예측된 sublayer들을 한꺼번에 읽어 들인다. 먼저, 각 sublayer list L_j별로($1 \le j \le c$) sublayer $L_{j,1}$,…, $L_{j,p[j]}$에 속하는 객체들을 한꺼번에 읽어 들여서 H에 추가한다. 그리고 $L_{j,p[j]}$에 속하는 객체들의 가장 작은 함수 값을 $A[j]$에 저장한다. 단계 2에서는 top-i 객체를 구한다($1 \le i \le k$). 먼저, H에서 i번째로 가장 작은 함수 값을 가지는 객체 o_i를 구한다. 또한, $A[1]$, …, $A[c]$중에서 가장 작은 값을 가지는 요소 $A[m]$을 구한다. 여기서, $A[m]$은 각 sublayer list로부터 가장 최근에 읽어 들인 sublayer에 속하는 객체들의 함수 값들 중에서 가장 작은 값을 나타낸다. 만약, o_i의 함수 값이 $A[m]$보다 작거나 같으면 o_i를 top-i 객체로 반환한다. 왜냐하면 $A[m]$의 정의와 보조정리 1에 의해 모든 sublayer list로부터 아직 읽지 않은 어떤 객체의 함수 값도 $A[m]$보다 더 작을 수 없기 때문이다. 이때, 만약 $i \ge k$ 이면 알고리즘을 끝내고 그렇지 않다면 i에 1을 더한다. 만약, o_i의 함수 값이 $A[m]$보다 크다면 L_m으로부터 추가로 sublayer를 읽어 들이고, $A[m]$의 값을 갱신한다.

5. 성능 평가

본 절에서는 APL-index를 사용한 질의 처리의 성능 평가 결과를 제시한다. 제 5.1절에서는 성능 평가를 수행한 실험 데이터와 실험 환경을 소개하고, 제 5.2절에서는 실험 결과를 설명한다.

5.1 실험 데이터 및 실험 환경

본 실험의 목적은 (1) APL-index를 주기억장치에 상주시킬 수 있을 정도로 APL-index의 크기(즉, 색인 크기)가 PL-index에 비해 매우 작음을 보이고, (2) APL-index를 구성하는 오버헤드가 크지 않음을 입증하기 위해 APL-index의 색인 구성 시간이 PL-index에 비해 매우 작음을 보이고, (3) APL-index를 이용하여 질의 처리 시 발생하는 랜덤 액세스의 횟수를 줄이므로 디스크 기반의 데이터베이스 환경에서 PL-index의 질의 처리 성능이 크게 향상됨을 보이는 것이다. 색인 구성 시간에 대한 척도(measure)로는 wall clock time(ms)을 사용하고 질의 처리 성능에 대한 척도로는 No_Objects_Read와 No_Random_Access를 사용한다. No_Objects_Read는 데이터베이스로부터 읽은 object들의 개수를 나타내고, No_Random_Access는 질의 처리시 발생하는 랜덤 액세스의 횟수를 나타낸다.

실험 데이터로는 합성 데이터(synthetic data)와 실제 데이터(real data)를 사용한다. 실제 데이터로는

질의 수행 알고리즘:

입력: (1) c개의 sublayer list $\{L_1, L_2, ..., L_c\}$로 구성된 PL-index

(2) Sublayer list별로 예측된 sublayer 개수가 저장된 크기 c인 배열 P

(3) 질의 $Q = (f(), k)$

출력: $f()$값이 가장 작은 k개의 객체들의 시퀀스 [top-1, top-2, ..., top-k]

알고리즘:

단계 0. 크기가 c인 배열 A와 집합 H를 초기화 한다.

단계 1. 예측된 sublayer를 sublayer list별로 읽어 들이는 단계:

1.1 $1 \le j \le c$ 에 대해 다음을 수행한다.

1.1.1 $L_{j,1}, ..., L_{j,P[j]}$에 속한 객체를 H에 추가한다.

1.1.2 $L_{j,P[j]}$에 속한 객체의 함수 값 중에서 가장 작은 값을 $A[j]$에 저장한다.

단계 2. top-i 객체를 구하는 단계: $1 \le i \le k$ 에 대해 다음을 수행한다.

2.1 $o^i := H$에서 i번째로 가장 작은 함수 값을 가지는 객체

2.2 $A[m] := A[1], ..., A[c]$ 중에서 가장 작은 값을 가지는 A의 요소

2.3 만약 $f(o^i) \le A[m]$ 이면 다음을 수행한다.

2.3.1 o^i를 top-i 객체로 반환한다.

2.3.2 만약 $i \ge k$ 이면 알고리즘을 끝낸다.

2.3.3 그렇지 않으면 $i := i + 1$

2.4 그렇지 않다면 다음을 수행한다.

2.4.1 $P[m] := P[m] + 1$

2.4.2 $L_{m,P[j]}$에 속한 객체를 H에 추가한다.

2.4.3 $L_{m,P[j]}$에 속한 객체의 함수 값 중에서 가장 작은 값을 $A[m]$에 저장한다.

그림 5. 예측된 sublayer와 PL-index를 이용하여 top-k 질의 결과를 구하는 알고리즘

PL-index[5]에서와 같이 Cover Forest Data[2]의 일부를 사용한다. 이 데이터를 Cover3D라 표기한다. Cover3D 데이터는 Cover Forest Data의 Elevation, Horizontal_Distance_To_Roadways, Horizontal_Distance_To_Fire_Points를 attribute로 가지는 10K개의 3차원 object들로 구성되고 attribute들 간의 correlation은 0.5 정도이다. 참고로, UNIFORM 데이터의 correlation은 0이다. 합성 데이터로는 PL-index[5] 및 AppRI[7]에서 실험을 위해 사용한 데이터 생성기를 사용하여 생성한 데이터를 사용한다.

실험 방법은 다음과 같다. 첫째, 색인 크기를 비교하기 위해 데이터의 크기 N, 차원 수 d, 분할 수준(partition level) h(sublayer list의 총 개수 $c = d^h$임)를 변화시키면서 APL-index와 PL-index의 크기를 측정한다. 여기서, 분할 수준은 전체 데이터베이스를 재귀적으로 분할하는 횟수를 나타낸다[5]. 본 실험에서는 (대표) 객체의 속성 한 개의 크기를 8Byte로 가정하고, "요약된 sublayer"의 크기를 저장하기 위해 8Byte를 사용한다고 가정한다. 따라서 d차원 (대표) 객체 한 개의 크기는 d*8Byte이고, 요약된 sublayer 한 개의 크기는 정의 1에 의해 (d+1)*8Byte이다. 둘째, 색인 구성 시간을 비교하기 위해 데이터의 크기 N, 차원 수 d, 분할 수준 h를 변화시키면서 APL-index와 PL-index의 색인 구성 시간을 측정한다. 셋째, 질의 처리 성능을 비교하기 위해 먼저 제4.1절에서 설명한 적중률(hit ratio)λ의 변화에 따른 APL-index의 질의 처리 성능을 분석한다. 이때, 적중률은 읽어 들인 한 sublayer에 속하는 객체들 중에서 결과에 포함되는 객체의 비율을 나타낸다. 다음으로, 질의 결과의 개수 k, 데이터의 크기 N, 차원 수 d를 변화시키면서 APL-index와 PL-index의 질의

2) http://www.ics.uci.edu/~mlearn/MLRepository.html

처리 성능을 측정한다. 질의 처리 성능은 속성에 대한 서로 다른 가중치를 가진 10개의 질의들을 랜덤하게 생성하여 No_Random_Access와 No_Object_Read를 각각 측정하며, 결과로서 그 평균값을 각각 보인다. 이때, 질의의 preference vector w에서 i번째 속성에 대한 가중치 $w[i]$ $(1 \leq i \leq d)$는 PL- index[5]에서와 같이 {1, 2, 3, 4} 중에서 랜덤하게 선택하고 $w[1]+w[2]+ \ldots + w[d] =1$이 되도록 정규화 한다.

실험을 위해 C++ 언어를 이용하여 APL-index와 PL-index를 구현한다. PL-index의 구현에서는 convex-hull의 계산을 위해 쉽게 구할 수 있고 성능이 우수한 Qhull library[18]를 사용한다. 실험은 1GB 메모리를 가진 Intel Pentium-4 2.0GHz Linux PC와 120GB Segate E-IDE 디스크를 사용하여 수행한다.

5.2 실험 결과

5.2.1 색인 크기

실험1: N의 변화에 따른 색인 크기 비교

그림 6(a)는 N이 1K부터 100K까지 증가할 때 APL-index와 PL-index의 색인 크기를 나타낸다. 제 3.2절에서 설명한 바와 같이 APL-index는 PL-index의 요약된 색인이기 때문에 PL-index에 비해 그 크기가 매우 작음을 알 수 있다. APL-index의 크기는 PL-index에 비해 6.0~94.1배 더 작음을 보였다.

실험2: d의 변화에 따른 색인 크기 비교

그림 6(b)는 d가 2부터 5까지 증가할 때 APL-index와 PL-index의 색인 크기를 나타낸다. APL-index의 크기는 PL-index에 비해 6.6 ~ 22.4배 더 작음을 보였다.

실험3: h의 변화에 따른 색인 크기 비교

그림 6(c)는 h가 1부터 3까지 증가할 때 APL-index와 PL-index의 색인 크기를 나타낸다. h가 커질수록 sublayer의 개수가 증가하는 경향이 있기 때문에[10] h가 증가할수록 커질수록 APL-index의 크기가 증가하는 경향을 가진다. APL-index의 크기는 PL-index에 비해 8.3~68.6배 더 작음을 보였다.

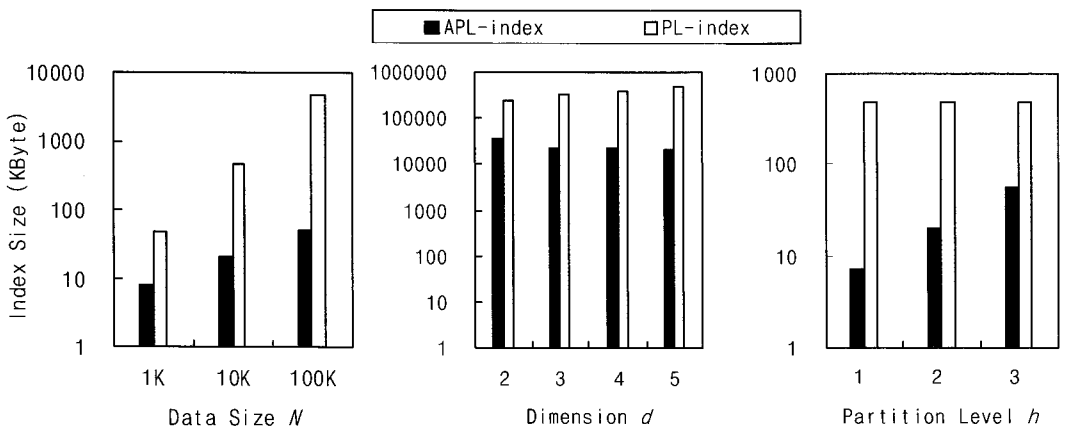
5.2.2 색인 구성 시간

실험4: N의 변화에 따른 색인 시간 비교

그림 7(a)는 N이 1K부터 100K까지 증가할 때 APL-index와 PL-index의 색인 시간을 나타낸다. 제 3.2절에서 설명한 바와 같이 APL-index는 PL-index의 sublayer에 속하는 객체들을 읽어서 단순 계산을 통해 얻어지기 때문에 복잡한 계산이 필요한 PL-index[10]에 비해 계산 시간이 매우 작음을 알 수 있다. APL-index의 색인 구성 시간은 PL-index에 비해 5.9~40.6배 더 작음을 보였다.

실험5: d의 변화에 따른 색인 시간 비교

그림 7(b)는 d가 2부터 5까지 증가할 때 APL-index와 PL-index의 색인 구성 시간을 나타낸다.

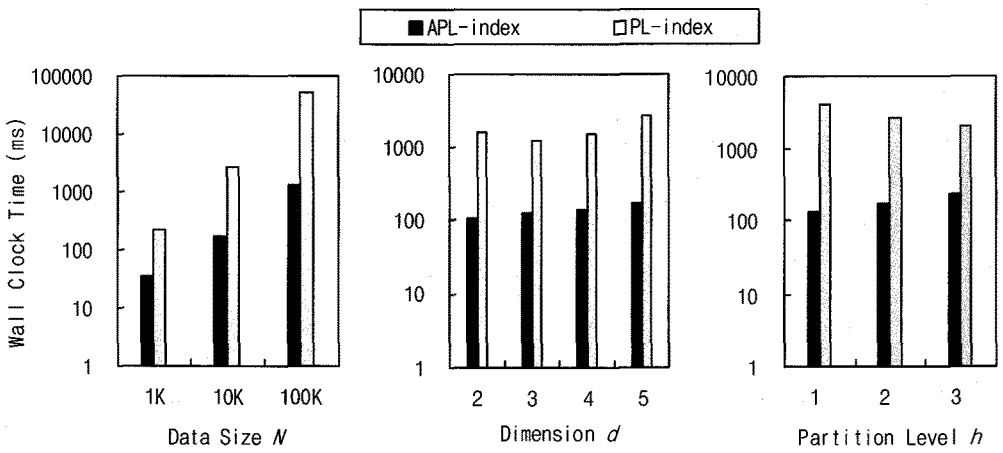


(a) N의 변화에 따른 색인 크기 (d=5인 UNIFORM 데이터)

(b) d의 변화에 따른 색인 크기 (N=10K인 UNIFORM 데이터)

(c) h의 변화에 따른 색인 크기 (d=5인 UNIFORM 데이터)

그림 6. APL-index와 PL-index에 대한 색인 크기의 비교



(a) N의 변화에 따른 색인 시간 (d=5인 UNIFORM 데이터)

(b) d의 변화에 따른 색인 시간 (N=10K인 UNIFORM 데이터)

(c) h의 변화에 따른 색인 시간 (N=10K, d=5인 UNIFORM 데이터)

그림 7. APL-index와 PL-index에 대한 색인 구성 시간의 비교

APL-index의 색인 구성 시간은 PL-index에 비해 10.4~15.6배 더 작음을 보였다.

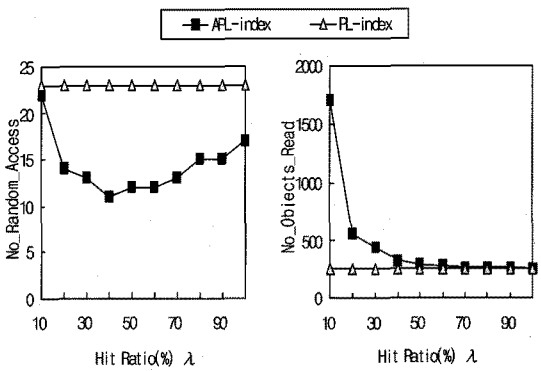
실험6: h의 변화에 따른 색인 시간 비교

그림 7(c)는 h가 1부터 3까지 증가할 때 APL-index와 PL-index의 색인 크기를 나타낸다. h가 커질수록 sublayer의 개수가 증가하는 경향이 있기 때문에[10] h가 증가할수록 커질수록 APL-index의 색인 구성 시간이 증가하는 경향을 가진다. APL-index의 색인 구성 시간은 PL-index에 비해 8.7~30.8배 더 작음을 보였다.

5.2.3 질의 처리 성능

실험7: λ의 변화에 따른 질의 처리 성능 비교

그림 8은 λ가 10%부터 100%까지 증가할 때 APL-index의 질의 처리 성능을 나타낸다. 여기서, PL-index는 λ에 영향을 받지 않기 때문에 일정한 질의 처리 성능을 나타낸다. 그림 8의 (a)와 (b)에서 알 수 있듯이 λ가 너무 커지거나 작아지면, APL-index의 No_Random_Access 또는 No_Read_Object가 커지는 경향을 나타낸다. 실험 결과를 통해 APL-index의 질의 처리 성능을 최적화하는 λ가 존재할 것으로 예상되지만, 데이터의 분포뿐만 아니라 N, d, k, h와 같은 파라미터에 따라 이러한 λ가 달라질 수 있기 때문에 이를 정확하게 예측하는 것은 불가능하다. 그림 8로부터 APL-index의 No_Random_Access와 No_Read_Objects를 다른 λ 값에 비해 모두 줄이는 λ=50을 별도의 언급이 없는 한 이후의 실험에서 계속 사용한다.



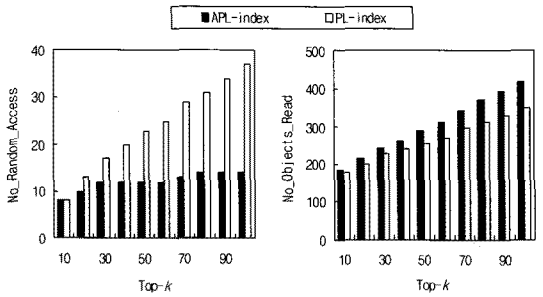
(a) 임의 접근 횟수를 비교

(b) 읽어 들인 객체의 개수를 비교

그림 8. 합성 데이터에서 λ의 변화에 따른 질의 처리 성능의 비교 (N=10K, d=3인 UNIFORM 데이터, k=50, λ=50)

실험8: k의 변화에 따른 질의 처리 성능 비교

그림 9는 k가 10부터 100까지 증가할 때 APL-index와 PL-index의 질의 처리 성능을 나타낸다. 그림 9(a)에서 보는 바와 같이 APL-index의 No_Random_Access가 PL-index에 비해 1.0~2.6배 더 작다. 반면, 그림 9(b)에서 보는 바와 같이 APL-index의 No_Objects_Read가 PL-index에 비해 같거



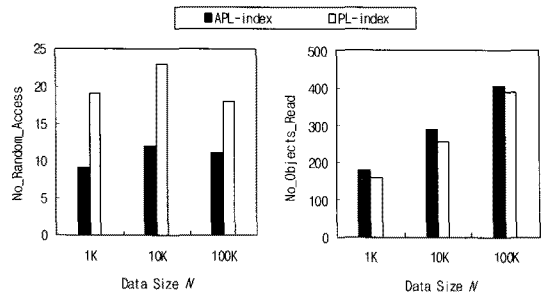
(a) 임의 접근 횟수를 비교 (b) 읽어 들인 객체의 개수를 비교

그림 9. 합성 데이터에서 k의 변화에 따른 질의 처리 성능의 비교(N=10K, d=3인 UNIFORM 데이터, λ=50)

나 최대 20% 더 많기 때문에 잘못 예측된 sublayer로 인한 오버헤드가 매우 작음을 알 수 있다.

실험 9: N의 변화에 따른 질의 처리 성능 비교

그림 10은 N이 1K부터 100K까지 증가할 때 APL-index와 PL-index의 질의 처리 성능을 나타낸다. 그림 10(a)에서 보는 바와 같이 APL-index의 No_Random_Access가 PL-index에 비해 1.6 ~ 2.1배 더 작다. 반면, 그림 10(b)에서 보는 바와 같이 APL-index의 No_Objects_Read가 PL-index에 비해 최대 1.1배에 불과하기 때문에 잘못 예측된 sublayer로 인한 오버헤드가 매우 작음을 알 수 있다.

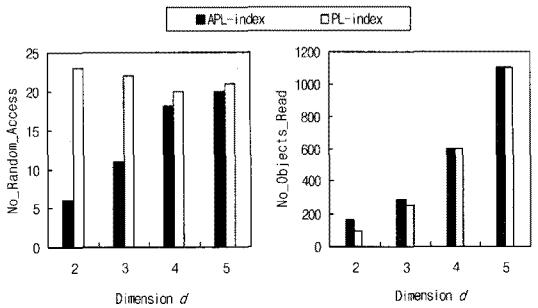


(a) 임의 접근 횟수를 비교 (b) 읽어 들인 객체의 개수를 비교

그림 10. 합성 데이터에서 N의 변화에 따른 질의 처리 성능의 비교 (d=3인 UNIFORM 데이터, k=50, λ=50)

실험 10: d의 변화에 따른 질의 처리 성능 비교

그림 11은 d가 2부터 5까지 증가할 때 APL-index와 PL-index의 질의 처리 성능을 나타낸다. 그림 11(a)에서 보는 바와 같이 APL-index의 No_Random_Access가 PL-index에 비해 1.1~3.8배 더 작다. 반면, 그림 11(b)에서 보는 바와 같이 APL-index의 No_Objects_Read가 PL-index에 비해 최대 1.6배에 불과하기 때문에 잘못 예측된 sublayer로 인한 오버헤드가 매우 작음을 알 수 있다.

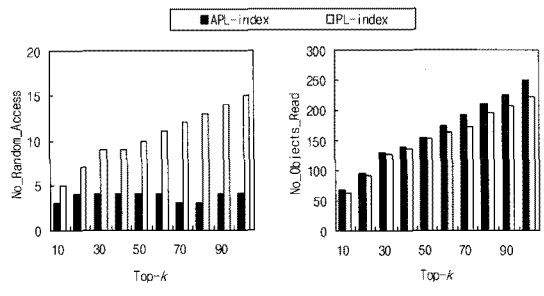


(a) 임의 접근 횟수를 비교 (b) 읽어 들인 객체의 개수를 비교

그림 11. 합성 데이터에서 d의 변화에 따른 질의 처리 성능의 비교(N=10K인 UNIFORM 데이터, k=50, λ=50)

실험 11: 실제 데이터를 사용했을 때 k의 변화에 따른 질의 처리 성능 비교

그림 12는 실제 데이터인 Cover3D 데이터를 사용했을 때 k의 변화에 따른 APL-index와 PL-index의 질의 처리 성능을 나타낸다. APL-index는 합성 데이터에 대한 실험 결과(그림 9)와 유사한 경향을 보인다. 그림 12(a)에서 보는 바와 같이 APL-index의 No_Random_Access가 PL-index에 비해 1.7~3.8배 더 작다. 반면, 그림 9(b)에서 보는 바와 같이 APL-index의 No_Objects_Read가 PL-index에 비해 최대 1.1배에 불과하기 때문에 잘못 예측된 sublayer로 인한 오버헤드가 매우 작음을 알 수 있다.



(a) 임의 접근 횟수를 비교 (b) 읽어 들인 객체의 개수를 비교

그림 12. 실제 데이터에서 k의 변화에 따른 질의 처리 성능의 비교(N=10K, d=3인 Cover3D 데이터, λ=50)

6. 결 론

본 논문에서는 디스크 기반 데이터베이스 환경에서 기존 PL-index의 top-k 질의 처리 성능을 향상시키는 Abstracted Partitioned-Layer Index (간단히, APL-index)를 제안하였다. 그리고 APL-index와 PL-index를 이용하여 정확한 top-k 질의 결과를 구하는 방법을 제안하였다. APL-index는 여러 개의 객체들로 구성된 PL-index의 sublayer를 대표 객체와 그 sublayer에 속하는 객체들의 개수로 요약하여 구성된다. 질의 처리시 APL-index를 이용하여 sublayer list 별로 access할 sublayer를 예측한다. 그리고 PL-index를 이용하여 예측된 sublayer를 sublayer list별로 sequential하게 한꺼번에 읽어 들인다. 또한, 정확한 top-k 결과를 보장하기 위해 특정 sublayer를 추가로 읽어 들임으로써 질의 결과를 구한다.

본 논문에서는 합성 데이터와 실제 데이터에 대해 색인 크기, 색인 구성 시간, 질의 처리 성능에 대해 많은 실험을 수행하였다. 첫째, APL-index의 크기가 PL-index에 비해 6.0~94.1배 더 작음을 보였다. 둘째, APL-index의 색인 구성 시간이 PL-index에 비해 5.9~40.6배 더 작음을 보였다. 셋째, APL-index의 No_Random_Access가 PL-index에 비해 최대 3.8배 더 작음을 보였고, APL-index의 No_Objects_Read가 PL-index에 비해 최대 1.6배에 불과함을 보였다.

이 같은 결과들로 볼 때, 본 논문에서 제안한 APL-index는 PL-index의 임의 접근 횟수를 크게 줄이기 때문에 임의 접근 비용이 순차 접근 비용보다 상대적으로 큰 디스크 기반의 데이터베이스 환경에서 PL-index의 성능을 크게 향상 시킬 수 있을 것으로 사료된다.

참 고 문 헌

[1] V. Hristidis and Y. Papakonstantinou, "Algorithms and applications for answering ranked queries using ranked views," The VLDB Journal, Vol. 13, No. 1, 2004.

[2] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song, "RankSQL: Query Algebra and Optimization for Relational Top-k Queries," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Baltimore, Maryland, June 2005.

[3] C. Li, K. C.-C. Chang, and I. F. Ilyas, "Supporting ad-hoc ranking aggregates," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Chicago, IL, June 2006.

[4] J.-S. Heo, J. Cho, and K.-W. Whang, "The Hybrid-Layer Index: A Synergic Approach to Answering Top-k Queries in Arbitrary Subspaces," In *Proc. 26th Int'l Conf. on Data Engineering (ICDE)*, Long Beach, California, Mar. 2010.

[5] J.-S. Heo, K.-Y. Whang, M.-S. Kim, Y.-R. Kim, and I.-Y. Song, "The Partitioned- Layer Index: Answering Monotone Top-k Queries Using the Convex Skyline and Partitioning-Merging Technique," Information Sciences, Vol. 179, No. 9, 2009

[6] R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, Santa Barbara, California, May 2001.

[7] D. Xin, C. Chen, and J. Han, "Towards Robust Indexing for Ranked Queries," In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, Seoul, Korea, Sept. 2006.

[8] Y. C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith, "The Onion Technique: Indexing for Linear Optimization Queries," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Dallas, Texas, May 2000.

[9] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis, "Answering Top-k Queries Using Views," In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, Seoul, Korea, Sept. 2006.

[10] Yi, K., Yu, H., Yang, J., Xia, G., and Chen, Y., "Efficient Maintenance of Materialized Top-k Views," In *Proc. Int'l Conf. on Data Engin-*

eering (ICDE), Bangalore, India, Mar. 2003.

[11] C.-Y. Chan, P.-K. Eng, and K.-L. Tan, "Stratified computation of skylines with partially-ordered domains," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 203-214, Baltimore, Maryland, June 2005.

[12] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. on Database Systems*, Vol. 30, No. 1, 2005.

[13] G. Beskales, M. A. Soliman, and I. F. Ilyas, "Efficient search for the top-k probable nearest neighbors in uncertain databases," In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, Auckland, New Zealand, Aug. 2008.

[14] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: a probabilistic threshold approach," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Vancouver, Canada, June 2008.

[15] S. Borzsonyi, D. Kossmann, and K. Stocker, "The Skyline Operator," In *Proc. Int'l Conf. on Data Engineering (ICDE)*, Heidelberg, Germany, Apr. 2001.

[16] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, 2000.

[17] S. G. Gass, *Linear Programming: Method and Applications*, 5th ed. An International Thomson Publishing Company, 1985.

[18] B. Barber, D. Dobkin, and H. Huhdanpaa, "The Quickhull Algorithm for Convex Hulls," *ACM Trans. on Mathematical Software*, Vol. 22, No. 4, 1996.

허 준 석

1995년 서울시립대학교 전산통계학과 학사
1997년 서울시립대학교 전산통계학과 석사
2009년 한국과학기술원 전산학과 박사
1997년~2002년 대우통신㈜ (현, 머큐리㈜) 선임연구원
2002년~2003년 한국과학기술원 첨단정보기술 연구센터 연구원
2009년~2010년 한국과학기술원 정보전자연구소 연수연구원
2010년~2010년 한국과학기술원 전산학과 연구조교수
2010년~현재 SAP Labs, Korea 연구원
관심분야 : 주기억장치 데이터베이스, 정보검색(IR), 공간데이터베이스