http://dx.doi.org/10.3745/KIPSTD.2012.19D.2.151

독립적인 벡터 근사에 의한 분산 벡터 근사 트리의 성능 강화

최 현 화[†] · 이 규 철[††]

요 약

지금까지 제안된 분산 고차원 색인의 대부분은 균일한 분포를 가지는 데이터 집합에서 좋은 검색 성능을 나타내나, 편향되거나 클러스터를 이루는 데이터의 집합에서는 그 성능이 크게 감소된다. 본 논문은 강하게 클러스터를 이루거나 편향된 분포를 가지는 데이터 집합에 대한 분산 벡터 근사 트리의 k-최근접 검색 성능을 향상시키는 방법을 제안한다. 기본 아이디어는 전체 데이터를 클러스터링하는 상위 트리의 말단 노드가 담당하는 데이터 공간의 크기를 계산하고, 그 공간 상의 특징 벡터를 근사하는 데 사용되는 비트의 수를 달리하여 벡터 근사의 식별 능력을 보장하는 것이다. 즉, 고밀도 클러스터에는 더 많은 수의 비트를 할당하는 것이다. 우리는 합성 데이터와 실세계 데이터를 가지고 분산 hybrid spill-tree와 기존 분산 벡터 근사 트리와의 성능 비교 실험을 수행하였다. 실험 결과는 확장된 분산 벡터 근사 트리의 검색 성능이 균일하지 않은 분포의 데이터 집합에서 크게 향상되었음을 보인다.

키워드 : 고차원 색인, 분산 색인, 근사 k-최근접 검색, 분산 병렬 알고리즘

Performance Enhancement of a DVA-tree by the Independent Vector Approximation

Hyun-Hwa Choi[†] · Kyu-Chul Lee[††]

ABSTRACT

Most of the distributed high-dimensional indexing structures provide a reasonable search performance especially when the dataset is uniformly distributed. However, in case when the dataset is clustered or skewed, the search performances gradually degrade as compared with the uniformly distributed dataset. We propose a method of improving the k-nearest neighbor search performance for the distributed vector approximation-tree based on the strongly clustered or skewed dataset. The basic idea is to compute volumes of the leaf nodes on the top-tree of a distributed vector approximation-tree and to assign different number of bits to them in order to assure an identification performance of vector approximation. In other words, it can be done by assigning more bits to the high-density clusters. We conducted experiments to compare the search performance with the distributed hybrid spill-tree and distributed vector approximation-tree by using the synthetic and real data sets. The experimental results show that our proposed scheme provides consistent results with significant performance improvements of the distributed vector approximation-tree for strongly clustered or skewed datasets.

Keywords : High-Dimensional Indexing, Distributed Indexing, Approximate k-NN Queries, Distributed and Parallel Algorithms

1. 서 론

인터넷 서비스에 따른 멀티미디어 데이터의 증가와 클라우드 컴퓨팅 서비스의 도래는 대용량 고차원 데이터의 관리 및 검색을 위한 분산 고차원 색인의 필요성을 더욱 가중시키고 있다. 멀티미디어 데이터베이스에서 멀티미디어 객체들은 보통 고차원의 벡터 공간 상에 특징 벡터들로 매핑되며, 특징 벡터의 k-최근접 검색은 여러 응용 서비스에서 멀티미디어 객체의 내용 기반 검색 방법으로 활용되고 있다.

여러 해에 걸쳐 다양한 분산 고차원 색인 방법[1-11]이 제안되어 왔다. 이들의 대부분은 대용량의 고차원 데이터의 관리를 위한 고확장성 혹은 빠른 검색을 위한 질의의 병렬 처리에 주안점을 두고 있다. 최근 우리는 분산 벡터 근사 트리[12]라는 새로운 분산 고차원 색인 구조를 제안하였다. 분산 벡터 근사 트리는 대용량의 고차원 데이터를 관리하기 위하

※ 본 연구는 지식경제부 및 한국산업기술평가관리원의 IT산업원천기술개발사업의 일환으로 수행하였음[10038768, 유전체 분석용 슈퍼컴퓨팅 시스템 개발].
† 정 회 원 : 한국전자통신연구원 선임연구원
†† 정 회 원 : 충남대학교 컴퓨터공학과 교수(교신저자)
논문접수 : 2011년 11월 25일
수 정 일 : 1차 2012년 1월 4일, 2차 2012년 3월 9일
심사완료 : 2012년 3월 13일

여 샘플 데이터를 바탕으로 전역 색인 구조인 hybrid spill-tree[13]를 단일 컴퓨팅 노드 상에 구축하고, 구축된 hybrid spill-tree 말단 노드 각각에 분산 컴퓨팅 노드를 매핑하여 VA-file[14]를 관리하는 두 단계의 분산 색인 구조이다. 분산 벡터 근사 트리는 다수의 노드들에 대용량의 고차원 데이터를 분산 관리하도록 하여 확장성을 지원한다. k-최근접 검색 시에는 hybrid spill-tree 탐색을 통해 연관성이 적은 특징 벡터 클러스터들의 접근을 생략하고, 트리 탐색 결과인 말단 노드들과 매핑된 분산 컴퓨팅 노드들에서 VA-file 기반 k-최근접 검색을 병렬로 수행한다. 그리하여, 분산 벡터 근사 트리는 기존 클러스터 기반 분산 색인 방법과 비교해 높은 검색 정확도를 유지하면서 빠른 검색을 지원한다.

분산 벡터 근사 트리를 포함하여 지금까지 제안된 분산 고차원 색인은 분할이 단순하고, 분할된 공간에 객체가 균일하게 분포되는 경우 좋은 검색 성능을 제공한다. 그러나, 이들은 모두 편향되거나 클러스터를 이루는 데이터의 집합에서 그 성능이 크게 감소되는 경향이 있다.

본 논문은 대체적으로 비균일한 분포를 가지는 응용 서비스의 대용량 멀티미디어 데이터의 빠른 검색을 지원하기 위하여 분산 벡터 근사 트리를 확장한 방법에 대해서 설명한다. 기본적으로 우리는 분산 벡터 근사 트리의 말단 노드별로 데이터 분포에 따라 데이터 공간을 독립적으로 분할한 벡터 근사 파일을 가지도록 확장하여 VA-file의 질의 성능을 향상시켰다. 또한, 분산 벡터 근사 트리의 비용 모델을 설명하고 독립적인 벡터 근사 파일을 가지도록 함으로써 향상된 검색 성능을 실험을 통해 보인다.

본 논문의 구성은 다음과 같다. 먼저, 2장에서는 분산 고차원 색인에 대한 관련 연구를 소개하고, 3장에서는 분산 벡터 근사 트리의 이해를 위한 기본적인 개념 및 정의에 대해서 설명한다. 4장에서 연구 동기 및 확장된 분산 벡터 근사 트리에 대해서 설명하고, 검색 비용 모델을 제시한다. 5장에서는 확장된 분산 벡터 근사 트리의 성능 실험 및 그 결과에 대해서 설명한다. 그리고 6장에서 결론을 맺는다.

2. 관련 연구

분산 고차원 색인은 P2P 환경을 기반으로 하는 색인 방법과 클러스터 환경을 바탕으로 한 색인 방법으로 크게 분류할 수 있다. P2P 환경 하에서 운영되는 고차원 색인 구조는 대용량의 특징 벡터 관리를 위한 확장성 지원을 목표로 제안되었다. 먼저, Bawa[4]와 Haghani[5]는 LSH(Local Sensitive Hashing)을 바탕으로 근사 k-최근접 검색을 지원하는 방법을 제안하였다. 그러나, 해쉬 기반 접근은 실 세계 데이터와 같이 편향된 분포를 가지는 데이터의 경우 검색 정확도가 많이 떨어진다. 뿐만 아니라, 해쉬 함수에 의해 결정된 저장소에 충분한 근접점이 없을 경우 이를 확장할 방법이 없기 때문으로, k-최근접 검색에는 적합하지 않다. SkipIndex[1]는 데이터 공간을 분할하는 방식의 k-d-tree를 바탕으로 데이터 공간을 skip graph에 매핑하는 데 있어, 데이터 값을 하나의 키로 압축하여 사용하였다. Dist[2]와 VBI-tree[3] 역시 트리를 바탕으로 한 접근 방법이나, 이들은 모두 고차원 데이터로 잘 확장되지 못한다는 문제점이 있다. P2P 환경 기반 고차원 색인은 기본적으로 클러스터 환경 기반 색인과 비교해 k-최근접 검색 시에 많은 노드 간 메시지 전달로 빠른 검색을 지원하기 어렵다.

클러스터 환경 기반 분산 고차원 색인 방법에 있어서도 다양한 연구가 있어왔다. 클러스터 환경 기반 분산 고차원 색인의 대부분은 단일 노드 환경에서 제안된 고차원 색인 방법을 분산 컴퓨팅 노드들로 확장한 형태이다. 먼저, Master R-trees [6], Master-Client R-trees [7], GPR-tree [8]는 R-tree를 분산 컴퓨팅 노드들로 분산 배치한 색인 방법이다. Master R-trees는 마스터 컴퓨팅 노드가 R-tree의 중간 노드(internal node)들을 포함하고 여러 분산 컴퓨팅 노드들에게 말단 노드의 페이지들을 할당하였다. 한편, Master-Client-trees는 마스터 컴퓨팅 노드에 전역 R-tree를 구축하고 말단 노드들을 매핑한 분산 컴퓨팅 노드들에 각각 R-tree를 구축하는 것이 특징이다. GPR-tree는 클러스터 내 분산 컴퓨팅 노드들에게 R-tree의 일부분을 가지도록 중복 배치하고 데이터 변경의 일관성을 유지하기 위한 방법을 소개하였다. 최근에 제안된 분산 hybrid spill-tree[9]는 메트릭 트리(metric tree)를 마스터 컴퓨팅 노드에서 샘플 데이터를 바탕으로 구축하고, 말단 노드 각각에 하나의 분산 컴퓨팅 노드를 매핑하도록 하여 병렬 검색을 수행하였다. 한편, Weber[10]는 대용량의 특징 벡터 및 근사 데이터를 분할 배분하여 병렬적으로 모두 스캔하는 병렬 VA-file 를 소개하였으며, Chang[11]은 VA-file의 벡터 근사값에 추가 정보를 더하여 필터링의 효과를 높인 CBF를 병렬 스캔하는 방법을 제안하였다.

트리 구조 기반의 색인은 분산 컴퓨팅 노드에서 운영하기 어려울 뿐만 아니라, 병렬적으로 트리 탐색을 수행하는데 어려움이 있다. 한편, VA-file의 분할 및 병렬 검색은 정확한 k-최근접 검색을 지원하나, 검색 요청 시에 매번 대용량의 데이터를 모두 스캔해야 하기 때문에 응용 서비스를 지원하는데 한계가 있다.

3. 기본 정의 및 개념

멀티미디어 데이터를 바탕으로 한 유사 검색에서 객체는 주요한 특징들을 고정된 수의 차원을 가지는 벡터 공간 상의 점으로 매핑된다. 그리하여, 우리가 가정하는 데이터베이스 DB는 D-차원의 데이터 공간 DS 상의 점들의 집합이라 할 수 있으며, N개의 객체들을 포함하는 D-차원의 데이터베이스 DB는 아래와 같이 정의된다.

$$DB = P_0, \ldots, P_{N-1} \tag{1}$$

$$P_i \in DS,\ i = 0, \ldots, N-1, DS \subseteq R^D \tag{2}$$

객체들 간의 유사성은 거리 함수를 통해서 측정된다. 여기서, 거리 함수(distance function)는 다음의 3가지 특징을 가진다.

$$Symmetry : d(O_x, O_y) = d(O_y, O_x) \tag{3}$$

$$Non negativity : d(O_x, O_y) > 0 \quad (O_x \neq O_y) \; and \; d(O_x, O_x) = 0 \tag{4}$$

$$Triangle inequality : d(O_x, O_y) \leq d(O_x, O_z) + d(O_z, O_y) \tag{5}$$

거리 함수로 가장 많이 사용되는 유클리디언 거리함수를 바탕으로 두 객체 P, Q의 L_2거리는 아래와 같이 계산된다.

$$L_2(P, Q) = d(P, Q) = \sqrt[2]{\sum_{i=0}^{D-1} (Q_i - P_i)^2} \tag{6}$$

이러한 거리 함수를 바탕으로 기본적인 유사 질의(similarity query)인 범위 질의(range query)와 k-최근접 질의(k-nearest neighbors query)는 다음과 같이 정의된다.

정의 1. 범위 질의

질의 객체 Q (단, $Q \in DS$)와 최대 검색 거리 r이 주어졌을 때, 범위 질의 $range(Q, r)$은 $d(O_j, Q) \leq r$를 만족하는 모든 객체 O_j를 반환한다.

정의 2. k-최근접 질의

질의 객체 Q (단, $Q \in DS$)와 정수 $k \geq 1$이 주어졌을 때, k-최근접 질의 $NN(Q, k)$는 Q로부터 가장 짧은 거리에 있는 k개의 객체를 반환한다.

유사 질의를 효과적으로 처리하기 위하여, 특징 벡터들 간의 거리를 바탕으로 계층적인 분할을 통해 클러스터링을 수행하는 트리 구조 기반 색인이 제안되었다. 트리는 객체들을 포함하는 MBR(Minimum Bounding Region)의 라우팅 노드(routing nodes)로 구성되며, 유사 질의 처리 시에 질의 객체와 라우팅 노드, 질의 객체와 색인 객체 간의 거리(distance), 최소 거리(minimum distance), 최대 거리(max distance)를 이용하여 가지치기를 수행된다. 예를 들어, M-tree[15]에서 질의 객체 Q가 주어졌을 경우 노드 탐색 시에, 반경(radius) r을 가지는 라우딩 노드 O_r를 기준으로 한 서브 트리 $T(O_r)$에서 내부 객체들 간의 최소 거리 및 최대 거리는

$$d_{\min}(T(O_r)) = \max\{d(O_r, Q) - r, 0\} \tag{7}$$

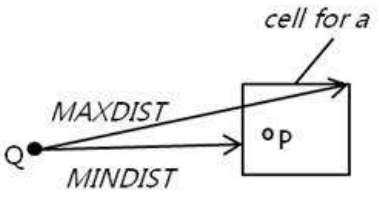
$$d_{\max}(T(O_r)) = d(O_r, Q) + r \tag{8}$$

와 같이 계산될 수 있다. 질의 객체 Q의 k-최근접 점을 찾기 위하여 M-tree를 탐색하는 과정에서 특정 노드 O_r의 최소 거리 $d_{\min}(T(O_r))$가 현재까지의 k번째 객체의 거리 d_k보다 큰 경우, 해당 노드 O_r를 루트 노드로 한 서브 트리는 탐색 대상에서 제외될 수 있다. 이와 같이 트리 구조 기반 색인은 질의 객체와 거리가 먼 객체들을 포함한 노드들을 가지치기 해 나감으로써 질의 처리 성능을 향상시킨다.

한편, 전체 특징 벡터의 근사값을 바탕으로 특징 벡터들을 필터링하는 방법 VA-file(vector Approximation file)[14]이 제안되었다. 특징 벡터의 근사값은 데이터 공간의 각 차원을 여러 구간으로 나누고, 나누어진 각 영역에 비트(bit)를 할당하여 생성한다. 즉, 각 특징 벡터는 비트 문자열의 근사값이 가리키는 특정 하나의 셀(cell)에 매핑된다. 예를 들어, 특징 벡터의 근사값을 표현하기 위한 비트 수를 b, 차원의 수를 D라 하였을 경우, 각 차원당 할당되는 비트 수 b_j와의 관계는 다음을 따른다.

$$b_j = \left\lfloor \frac{b}{D} \right\rfloor + \begin{cases} 1 & if \; j \leq (b \bmod D) \\ 0 & otherwise \end{cases} \tag{9}$$

각 차원에 할당된 비트 수는 차원 내의 구간 축을 결정하기 위하여 필요하다. j차원에서 2^{b_j}의 구간으로 분할된다면, $2^{b_j}+1$개의 구간 축 $m_j[0], ..., m_j[2^{b_j}]$이 필요하고, 구간들은 $0, ..., 2^{b_j}-1$로 표현될 수 있다. 예를 들어, 2차원의 데이터 공간에서 특징 벡터의 근사값이 4bit로 구성되는 경우, 각 차원 별로 2bit가 할당된다. 각 차원은 $2^2 = 4$의 구간으로 분할되고, 객체의 실제 값이 0에서 1사이인 경우 5개의 구간 축 $m_j[0], ..., m_j[2^2]$은 각각 0, 0.25, 0.5, 0.75, 1을 가리킨다. 상기 구간 축을 바탕으로 분할된 각 구간들은 0부터 3까지의 비트 00, 01, 10, 11로 표현된다. 따라서, 전체 2차원의 데이터 공간은 $(4)^2 = 16$개의 셀로 구성되며, 각 셀은 0부터 15까지를 나타내는 4bit의 비트 문자열에 의해 지칭될 수 있다.



(그림 1) VA-file에서의 최대, 최소 거리

VA-file에서 (그림 1)과 같이 질의 객체 Q가 주어졌을 경우, 근사값을 바탕으로 한 필터링은 객체 P가 매핑된 셀 a을 바탕으로 최소, 최대 거리 계산을 통해 이뤄진다[16]. 예를 들어 질의 객체 Q와 객체 P의 거리는 다음과 같다.

$$MINDIST \le L_2(Q,P) \le MAXDIST \quad (10)$$

$$MINDIST = \sqrt[2]{\sum_{j=1}^{D}(MINDIST_j)^2}$$
$$where\ MINDIST_j = \begin{cases} Q_j - m_j[a_j+1] & a_j < Q_j \\ 0 & a_j = Q_j \\ m_j[a_j] - Q_j & a_j > Q_j \end{cases} \quad (11)$$

$$MAXDIST = \sqrt[2]{\sum_{j=1}^{D}(MAXDIST_j)^2}$$
$$where\ MAXDIST_j = \begin{cases} Q_j - m_j[a_j] & a_j < Q_j \\ \max(Q_j - m_j[a_j], m_j[a_j+1] - Q_j) & a_j = Q_j \\ m_j[a_j+1] - Q_j & a_j > Q_j \end{cases} \quad (12)$$

VA-file에서는 위에서 설명한 질의와 근사값의 최소, 최대 거리를 바탕으로 필터링을 수행한다. 그리고, 필터링 후 남겨진 근사값에 해당하는 특징 벡터들과 질의 특징 벡터간의 실제 거리 계산을 수행하여 최종 유사 질의 결과를 산출한다. 이때, 순차적인 읽기와 빠른 근사값의 거리 연산에 따른 특징 벡터의 대량 필터링을 통해 디스크 I/O를 줄임으로써 유사 검색의 성능은 향상되었다.

4. 분산 벡터 근사 트리 강화

이 장에서 우리는 병렬 VA-file과 분산 벡터 근사 트리를 포함하여, 기존 VA-file의 문제점을 언급한다. 언급한 문제점을 해결하고 k-최근접 검색을 좀더 효과적으로 수행하기 위하여, 확장된 분산 벡터 근사 트리에 대해서 자세히 설명한다.

4.1 연구 동기

특징 벡터의 근사값을 활용한 VA-file은 데이터 공간의 높은 차원 수에 의하여 특징 벡터들이 같은 근사값을 가질 가능성이 거의 없으며, 대다수의 셀들은 비어있다는 가정 [14] 하에 이뤄지고 있다. 단위 하이퍼 큐브(unit hybper-cube) $\Omega = [0,1]^D$ 내의 데이터 집합을 고려해보자. i 차원에 b_i의 비트 수가 할당되었다고 가정하면, i 차원은 2^{b_i}의 동일 크기의 구간으로 나누어진다. 그리하여 D-차원의 데이터 공간은 $\sum_{i=1}^{D} b_i = b$와 같은 길이의 비트 문자열로 표현되는 2^b개의 셀로 구성된다. D-차원의 데이터 공간에서 N개의 특징 벡터가 균일하게 분포되는 경우, 한 특징 벡터가 특정 한 셀에 놓일 확률은 그 셀의 크기와 비례하게 된다.

$$Vol(cell) = \frac{1}{2^b} \quad (13)$$

따라서, 다른 특징 벡터가 같은 셀에 놓일 확률은

$$1 - (1 - \frac{1}{2^b})^{N-1} \approx \frac{N}{2^b} \quad (14)$$

와 같다. 여기서, 특징 벡터의 수 $N = 1{,}000{,}000 \approx 2^{20}$, 차원의 수 $D = 100$, 각 차원 당 비트 수 $b_i = 4$라 가정하면, 두 특징 벡터가 같은 셀에 놓일 확률은 2^{-380}으로 희박하다. 한편, 특징 벡터의 수에 비하여, 셀의 수 $2^{b_i D} = 2^b$가 훨씬 크기 때문에 대부분의 셀은 비어있다는 가정은 성립한다.

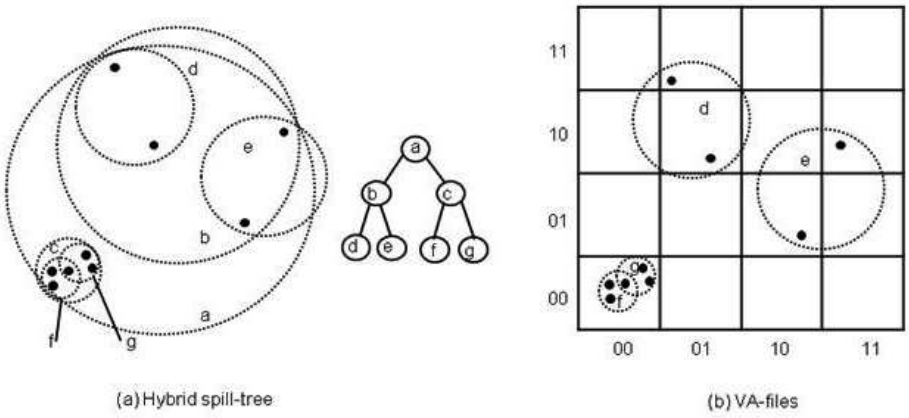
그러나, 실세계 데이터들은 대부분 균일하게 분포되어 있지 않기 때문에, 한 셀에 여러 개의 특징 벡터가 놓일 가능성은 증가한다. 또한, 데이터 공간 내 특징 벡터의 개수가 많아질수록 동일한 근사값을 사용하는 특징 벡터들의 수도 함께 증가한다. 따라서 균일하게 분포하지 않은 특징 벡터들을 동일한 비트 수를 사용하여 근사하는 경우 필터링의 효과는 떨어지게 된다. 즉, 특징 벡터의 근사값을 구축하는데 있어 데이터 공간을 같은 크기로 분할한 셀을 할당하는 경우 각 셀에 매핑되는 특징 벡터의 수를 나타내는 셀 간의 밀도 차가 극심해질 수 있다. 이는 근사값의 구분 능력을 떨어뜨리게 되어 k-최근접 검색의 성능 또한 떨어뜨리게 되는 요인이 된다. 특히, 대용량의 특징 벡터를 대상으로 k-최근접 검색을 수행하는 경우 근사값의 구분 능력은 검색 성능을 좌우하는 중요한 요인 중의 하나라 하겠다. 이러한 문제를 극복하고, 대용량의 데이터의 관리 및 검색을 지원하기 위하여 우리는 데이터 공간의 지역적 통계 특성을 바탕으로 독립적인 특징 벡터의 근사 파일을 사용하는 확장된 분산 벡터 근사 트리를 제안한다.

4.2 확장된 분산 벡터 근사 트리

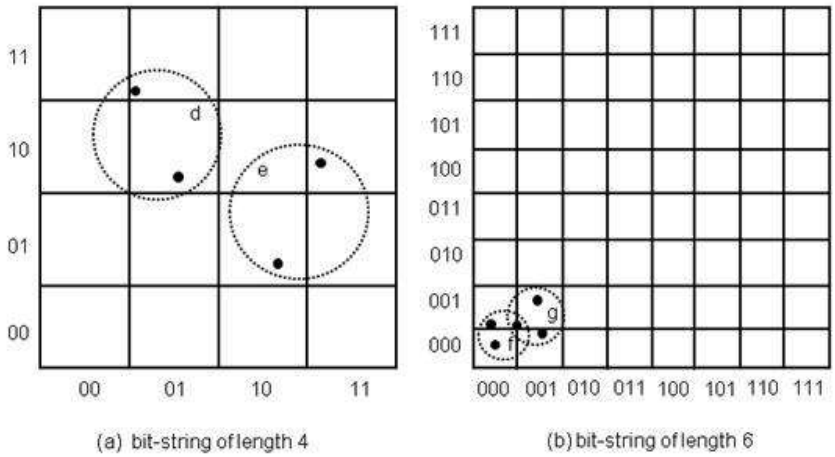
분산 벡터 근사 트리는 한 컴퓨팅 노드에서 감당할 수 없을 만큼 많은 특징 벡터들을 여러 컴퓨팅 노드들에 분산시키기 위하여 제안되었다. 분산 벡터 근사 트리는 전역 색인과 여러 개의 지역 색인들로 구성된다. 우리는 전역 색인을 위하여 전체 특징 벡터 $N = \{O_1, ..., O_n\}$로부터 랜덤으로 m 개의 샘플 데이터 $S = \{O_{s_1}, ..., O_{s_m}\}$를 추출한다. 샘플링 데이터의 최소 크기는 Yamane [17]에 의해 제안된 공식

$$m \ge \frac{n}{n * e^2 + 1} \quad (15)$$

으로 계산될 수 있다. 여기서, m은 샘플 데이터 크기이며, n은 데이터 모집단의 크기, e는 표본 오차를 나타낸다. 우리는 샘플 데이터를 바탕으로 빠른 검색과 정확도를 제공하는 최근 개발된 hybrid spill-tree를 구축하고, 구축된 hybrid spill-tree의 각 말단 노드에 분산 컴퓨팅 노드를 매핑하였다. 전체 특징 벡터 집합 N에 속한 각각의 객체들은



(그림 2) 분산 벡터 근사 트리



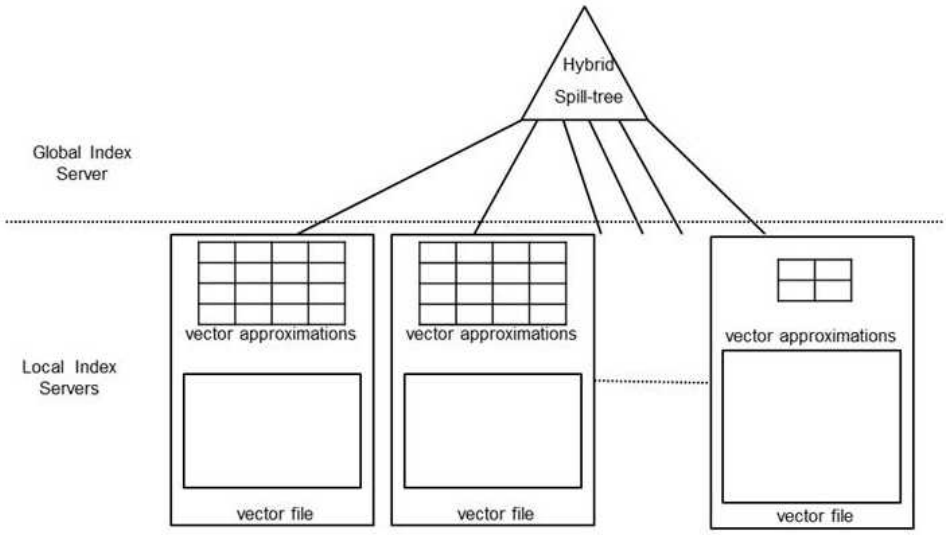
(그림 3) 말단 노드 별 독립적인 VA-file

hybrid spill-tree 탐색을 통해 삽입할 말단 노드들이 결정되면, 해당 말단 노드들과 매핑된 분산 컴퓨팅 노드들에 전달되어 저장된다. Hybrid spill-tree의 각 말단 노드에 매핑된 분산 컴퓨팅 노드들은 저장할 특징 벡터들을 바탕으로 VA-file을 구축한다.

(그림 2)의 (a)는 균일하지 않은 분포를 가지는 샘플 데이터를 바탕으로 구축된 hybrid spill-tree를 나타내며, (b)는 말단 노드 별로 구축되는 2-차원의 VA-file들의 모습을 보여준다. 각 VA-file은 각 차원에 2 비트를 할당하여 2-차원의 데이터 공간을 2^4개의 동일한 크기의 셀로 분할함으로써 특징 벡터의 근사값을 생성한다. Hybrid spill-tree의 말단 노드 d와 e에 삽입되는 특징 벡터들은 각 말단 노드가 관할하는 데이터 공간에 해당하는 여러 셀 중의 한 셀로 매핑된다. 그러나, 말단 노드 f와 g에 삽입되는 특징 벡터들은 모두 동일한 셀 값을 가지게 된다. 그리하여, VA-file를 바탕으로 k-최근접 검색을 수행하는 경우 근사값을 바탕으로 한 필터링의 효과를 얻을 수 없다.

이러한 문제점에 착안하여, 우리는 hybrid spill-tree의 각 말단 노드가 관할하는 데이터 공간 크기에 적절한 비트 문자열을 할당하여 특징 벡터를 근사하도록 한다. (그림 3)은 상기 구축된 hybrid spill-tree의 말단 노드 d와 e에는 차원당 2 비트를 할당하여 4비트의 비트 문자열로 특징 벡터를 근사하는 반면, 말단 노드 f와 g에 대해서는 차원당 3 비트를 할당하여 6 비트의 특징 벡터 근사값을 생성한다. 그리하여, k-최근접 검색 시 hybrid spill-tree 탐색을 통해 결정된 말단 노드들은 모두 근사값에 기반한 필터링 효과를 얻을 수 있다.

(그림 4)는 확장된 분산 벡터 근사 트리의 구조를 나타낸다. 확장된 분산 벡터 근사 트리는 전역 색인 hybrid spill-tree와 트리 내 각 말단 노드와 매핑된 분산 컴퓨팅 노드 내의 독립적인 공간 분할에 따른 VA-file를 가지는 지역 색인들로 구성된다. 각 분산 컴퓨팅 노드들은 벡터 근사값들을 메인 메모리에 저장하기 때문에 서로 다른 수의 특징 벡터들을 저장할 수 있다. 즉, 상대적으로 적은 비트를 사용하여 특징 벡터를 근사한 분산 컴퓨팅 노드들은 더 많은 특징 벡터를 저장할 수 있다. 여기서, 우리는 각 분산 컴퓨팅 노드에서 적절한 벡터의 근사 정도를 결정하는 것에 대해서 생각해 볼 필요가 있다. 색인 구축 시 각 분산 컴퓨팅 노드



(그림 4) 확장된 분산 벡터 근사 트리 구조

들은 모두 비슷한 수의 객체를 저장하고 있다. 분산 컴퓨팅 노드가 관할하는 데이터 공간의 크기가 큰 경우, 즉 데이터 공간 상에 존재하는 객체의 개수에 따른 밀도가 낮은 경우, 너무 많은 비트를 사용한 특징 벡터의 근사는 근사값의 크기를 불필요하게 증가시키게 된다. 반대로, 밀도가 높은 경우 너무 적은 비트로 특징 벡터를 근사하면 많은 객체들이 같은 근사값을 공유하게 된다. 가장 좋은 방법은 단말 노드의 해당 특징 벡터들을 서로 다른 수의 비트로 압축한 후 검색을 수행하여 가장 좋은 성능을 나타내는 비트로 결정하는 것이다. 그러나, VA-file의 경우 특징 벡터를 압축하는데 있어 한 차원당 4-8비트를 사용하는 것이 가장 좋은 성능을 나타내는 것으로 알려져 있다[14]. 그리하여 우리는 분산 벡터 근사 트리의 검색 성능을 향상시키면서 비트 설정을 용이하게 하기 위하여, 각 말단 노드에 해당하는 클러스터 크기를 계산한 후 노드 간의 상대적인 클러스터 크기에 맞춰 비트 수를 할당하였다. 반지름 r를 가지는 단말 노드에 해당하는 클러스터 크기는

$$Vol(node) = V_{sphere}(r) = \frac{\sqrt{\pi^d}}{\Gamma(d/2+1)} \cdot r^d \approx V_{cube}(r) = (2r)^d \tag{16}$$

으로 계산될 수 있다. 단, $\Gamma(x+1) = x \cdot \Gamma(x), \Gamma(1) = 1$, 그리고 $\Gamma(1/2) = \sqrt{\pi}$ 이다. 먼저 벡터 근사에 사용할 비트 수의 리스트를 결정한다. 결정된 비트 수 리스트에서 각 비트 수로 근사할 수 있는 최대 클러스터의 크기를 계산한다. 최대 클러스터 크기는 계산된 말단 노드들의 클러스터 크기를 반영하기 위하여 최소 및 최대 클러스터의 합을 비트 수 리스트의 크기로 나눈 평균 클러스터 크기의 배수로 구성한다. 여기서, 최대 클러스터 크기를 저장한 리스트는 비트 수 리스트보다 1개 작게 구성되며,

```
Algorithm DetermineBitLengthPerLeadNode (N: top-tree_nodes, B: bit-list_entries)
Input: the leaf nodes of a top-tree on sample data
       the list of bit lengths for vector approximation in descending order

1.  Let leafBitLengthList be a list to store the bit length assigned to each leaf node.
2.  leafClusterSizeList = calculateLeafClusterSize(N);
3.  avgClusterSize = ( min(leafClusterSizeList.entries) + max(leafClusterSizeList.entries) )
    / size(B);
4.  maxClusterSizeList = calculateMaxClusterSizePerBitLength(avgClusterSize, size(B));
     // the size of maxClusterSizeList in ascending order is the B size -1.
     // The values of the max cluster size  are multiples of the average cluster size.
5.  For each leaf node n ∈ N
6.     int idx = getMaxClusterIndex(maxClusterSizeList, leafClusterSizeList.get(n));
7.     leafBitLengthList.add(n, B(idx));
```

(그림 5) 말단 노드 별 비트 수 결정 알고리즘

최대 클러스터 크기와 비트 수와는 반비례 관계이다. 각 말단 노드는 해당 클러스터 크기보다 같거나 작은 최대 클러스터 크기와 매핑되는 비트 수를 사용하여 특징 벡터를 근사한다. 만약 같거나 작은 최대 클러스터 크기를 찾을 수 없는 경우에는 가장 큰 비트 수가 할당된다. 이를 요약한 알고리즘은 (그림 5)와 같다.

이렇게 분산 컴퓨팅 노드마다 서로 독립적으로 벡터를 근사함으로써 단축된 k-최근접 검색 시간은 성능 실험 결과를 통해 확인할 수 있다.

4.3 k-최근접 검색 비용 모델

확장된 분산 벡터 근사 트리를 기반으로 한 k-최근접 검색은 기존 분산 벡터 근사 트리의 k-최근접 검색 방법을 그대로 따른다. 그리하여, k-최근접 검색 비용은 다음의 3단계로 구성된다.

(1) 트리의 탐색 비용 : T_{1st}
(2) VA-file 검색 비용 : T_{2nd}
(3) 후보 벡터 병합 비용 : T_{3rd}

분산 벡터 근사 트리에서 hybrid spill-tree의 탐색은 접근해야 할 분산 컴퓨팅 노드를 결정하기 위한 것이다. 트리 탐색을 통해 결정된 말단 노드들과 매핑된 각 분산 컴퓨팅 노드들은 병렬적으로 k-최근접 검색을 수행한다. 분산 컴퓨팅 노드들은 VA-file을 기반으로 k-최근접 검색을 수행하여 k 개의 후보 집합을 결정한다. 마지막으로 분산 컴퓨팅 노드들로부터 수집된 후보 집합을 대상으로 최종 k개의 최근접 점을 결정한다. <표 1>은 비용을 예측하는데 있어 사용되는 기호들을 나열한 것이다.

〈표 1〉 검색 비용 예측을 위해 사용된 기호

Symbol	Description
n	전체 색인된 객체의 수
d	객체의 차원 수
Q	질의 객체
k	요청된 최근접 점의 수
$\bar{k}$	평균 k번째 거리
$F(x)$	거리 분포
O_r	라우팅 객체
$r(N_r)$	노드 N_r의 반지름
l	트리의 전체 노드 수
m	질의 검색 시 선택된 말단 노드의 수
v	분산 컴퓨팅 노드에서 색인된 객체의 수
b	벡터 근사에 사용되는 비트 수
$t_{approx}(b)$	차원당 경계값 계산 시간
t_{vector}	차원당 두 점 사이의 거리 계산 시간
C_{read}	디스크로부터 블록을 읽어들이는 비용
w	근사값을 바탕으로 필터링 후 남겨진 객체의 수
$t_{compare}$	두 거리 값의 비교 시간

먼저, Hybrid spill-tree의 탐색 비용을 예측해보자. 전체 n개의 객체로부터 추출된 샘플 데이터 집합을 바탕으로 hybrid spill-tree를 구축할 때, 우리는 데이터 간의 k번째 평균 거리를 계산해 놓았다. 이를 이용하여 k-최근접 질의 $NN(Q,k)$은 범위 질의 $Range(Q,\bar{k})$로 변환되어 트리를 탐색하게 된다. 트리 내 노드 N_r은 질의 객체 Q를 중심으로 반지름 $\bar{k}$을 가지는 구와 노드 N_r의 영역이 교차되면 접근되어야 한다. 이는 $d(Q,O_r) \le r(N_r)+\bar{k}$을 만족하기 때문이다. 거리 분포 $F(x)=\Pr\{d(O_1,O_2) \le x\}$로 정의하였을 경우, 노드 N_r이 접근되어야 하는 가능성(probability)은 아래와 같이 표현될 수 있다.

$$\begin{aligned}\Pr\{node\ N_r\ is\ accessed\} &= \Pr\{d(Q,O_r) \le r(N_r)+\bar{k}\} \\ &= F_Q(r(N_r)+\bar{k}) \approx F(r(N_r)+\bar{k})\end{aligned} \quad (17)$$

범위 질의 수행 시에 접근해야 하는 노드의 개수는 트리 노드들의 접근 가능성의 합으로 계산될 수 있다.

$$nodes(Range(Q,\bar{k})) = \sum_{i=1}^{l} F(r(N_{r_i})+\bar{k}) \quad (18)$$

말단 노드를 포함한 트리 내 모든 노드 정보는 메모리 구축이 가능하므로, 트리 탐색 비용은 I/O 비용 없이 벡터 간의 거리 연산 비용의 합으로 예측할 수 있다.

$$T_{1st} = nodes(Range(Q,\bar{k})) \cdot (d \cdot t_{vector}) \quad (19)$$

트리 탐색을 통해 결정된 m개의 말단 노드와 매핑된 각 분산 컴퓨팅 노드에서 VA-file을 바탕으로 k-최근접 검색이 병렬로 수행된다. 분산 컴퓨팅 노드에 저장된 v개의 특징 벡터들은 모두 b비트의 근사값을 가지며, 모든 근사값이 메인 메모리에 올라와 있음을 가정하기 때문에 필터링 단계에서는 I/O 연산이 발생하지 않는다. 그리하여, 필터링 단계의 비용은

$$T_A = v \cdot d \cdot t_{approx}(b) \quad (20)$$

이다. 반면, 필터링 후 남겨진 데이터 w에 한하여 벡터 파일의 임의 접근이 이뤄지며, 이때 디스크 I/O 가 발생한다. 벡터 정제 단계의 비용은

$$T_v = w \cdot (C_{read} + d \cdot t_{vector}) \quad (21)$$

으로 계산된다. 여기서, w의 수에 대한 예측은 [19]을 참고하기 바란다. 그리하여, VA-file의 검색 비용은 필터링 단계 및 벡터 정제 단계의 합으로 구성된다.

$$T_{2nd} = T_A + T_v \quad (22)$$

m개의 분산 컴퓨팅 노드들로부터 얻어진 $m \cdot k$ 개의 후보 벡터들을 비교하여 최종 k개의 최근접 점을 결정한다. 각 분산 컴퓨팅 노드들은 k개의 후보 벡터들을 정렬하여 보내기 때문에 후보 벡터들의 병합 비용은

$$T_{3rd} = k \cdot (m-1) \cdot t_{compare} \quad (23)$$

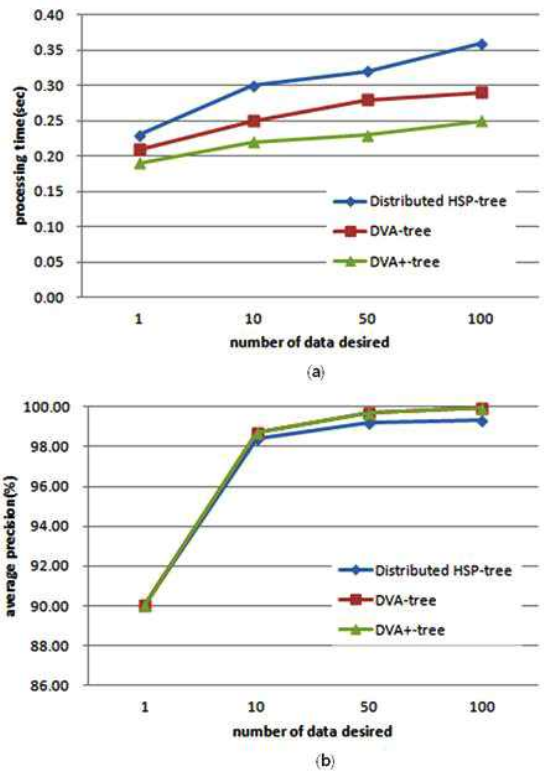
이다. 분산 벡터 근사 트리에 대한 k-최근접 검색 비용은 병렬적으로 수행되는 T_{2nd}에 많이 좌우된다. 기존의 분산 벡터 근사 트리는 각 분산 컴퓨팅 노드가 담당하는 데이터 공간의 크기와 상관없이 같은 비트 수로 특징 벡터를 압축하였다. 그리하여 비균일한 데이터 집합에서 분산 벡터 근사 트리는 분산 컴퓨팅 노드에서 수행하는 근사값 기반 필터링의 효과가 적어지면서 검색 성능이 저하되었다. 확장된

분산 벡터 근사 트리는 각 분산 컴퓨팅 노드마다 특징 벡터의 독립적인 근사를 수행하여 필터링 효과를 보장하였다. 그리하여, 필터링 후 남겨지는 데이터에 따른 벡터 파일의 I/O를 줄임으로써 전체 분산 벡터 근사 트리의 k-최근접 검색 시간을 단축시켰다.

5. 성능 평가

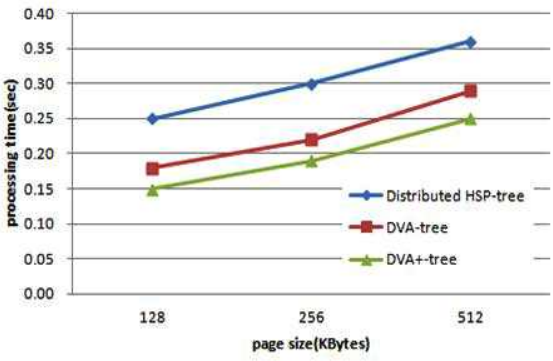
확장된 분산 벡터 근사 트리의 성능 평가를 위하여 편향된 데이터 분포를 가지는 270,000개의 61-차원의 실세계 데이터 Aerial40[20]을 사용하여 실험을 수행하였다. 또한, 더 큰 데이터 집합을 형성하기 위하여 각 분산 컴퓨팅 노드로 떨어지는 특징 벡터를 대상으로 무작위로 2개의 특징 벡터를 선택하고 이에 대한 평균치로 새로운 데이터를 만들어 기존 데이터의 분포를 유지하면서 1,000,000 개까지의 합성 데이터 집합을 구성하였다. K-최근접 검색 수행 시에 거리 계산에서는 유클리디언 거리 L_2를 사용하였고, 성능은 100개의 다른 질의 특징 벡터를 사용한 평균 k-최근접 실행 시간과 평균 검색 정확도를 바탕으로 평가하였다. 실험은 8대의 리눅스 기반 컴퓨팅 노드에서 수행되었으며, 컴퓨팅 노드들은 하나의 글로벌 파일 시스템을 가지도록 하였다. 각 컴퓨팅 노드들은 3.40 GHz Pentium® D CPU 프로세서와 2.4GB의 메모리의 동일 스펙을 가진다.

확장된 분산 벡터 근사 트리의 성능은 기존 분산 벡터 근사 트리[12]와 구글에 의해 제안된 분산 hybrid spill-tree[9]의 성능과 비교 분석하였다. 사용된 분산 색인 알고리즘은 모두 M-tree C++ 패키지[21]을 사용하여 개발하였다. 공정한 실험을 위하여, 우리는 같은 샘플 데이터를 바탕으로 3개의 색인 알고리즘의 상위 트리를 구축하였다. 전역 색인인 상위 트리에 사용된 페이지 크기는 512Kbytes이며, 지역 색인 VA-file에서 특징 벡터를 근사하기 위하여 사용된 비트 수는 차원당 4 혹은 8비트이다.



(그림 6) k-최근접 검색 시간 및 정확도

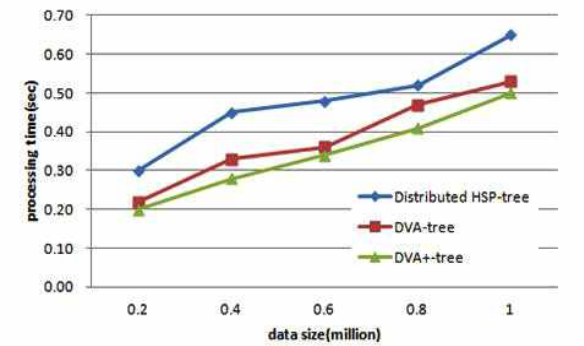
(그림 6)는 k-최근접 검색의 수행 결과로 (a)는 검색 시간을 (b)는 검색의 정확도를 나타낸다. 지역 색인으로 hybrid spill-tree를 사용하는 분산 hybrid spill-tree와 비교하여, VA-file을 사용하는 분산 벡터 근사 트리는 좋은 성능을 보인다. 한편, 확장된 분산 벡터 근사 트리는 기존의 분산 벡터 근사 트리의 검색 시간을 20%까지 단축시켰다. 같은 데이터 샘플링을 바탕으로 상위 트리를 구축하여 k-최근접 검색 시 접근하는 분산 컴퓨팅 노드의 수가 같으므로, 검색 시간의 차이는 병렬로 수행되는 분산 컴퓨팅 노드들의 VA-file의 k-최근접 검색 시간의 차와 같다. VA-file의 k-최근접 검색이 전체 벡터 근사값을 모두 스캔하여 이뤄짐을 감안하면, 확장된 분산 벡터 근사 트리가 필터링 효과를 보장함으로써 접근해야 할 특징 벡터 수를 감소시켜 성능 향상이 이뤄졌음을 알 수 있다. 즉, 두 분산 벡터 근사 트리의 성능 차이는 디스크 I/O 횟수 감소에 따른 것으로, k 값이 커질수록 그 차이도 커짐을 (a)에서 확인할 수 있다. 우리는 데이터 분포의 편향도가 클수록 검색 시간의 차이는 더 커질 것이라 생각한다. 한편, 확장된 분산 벡터 근사 트리의 검색 정확도는 hybrid spill-tree보다는 좋으면서 기존 분산 벡터 근사 트리와 동일하다. 확장된 분산 벡터 근사 파일은 $k=10$일 때 98.7%를 보이며, $k \geq 50$ 인 경우에는 99.5%를 넘는 정확도를 보였다.



(그림 7) 페이지 크기에 따른 k-최근접 검색

(그림 7)은 상위 트리의 페이지 크기를 달리함으로써, 각 분산 컴퓨팅 노드가 담당하는 데이터 공간 변경에 따른 검색 시간을 측정하였다. 이때 $k=100$으로 고정하여 최근접 검색을 수행하였다. 페이지 크기가 커질수록 상위 트리의 말단 노드의 수는 줄어드는 반면, 각 분산 컴퓨팅 노드가 저장하는 특징 벡터의 수는 증가한다. 그리하여, 페이지 크기와 함께 최근접 검색의 시간도 증가한다. 한편, 페이지 크

기가 작아질수록 데이터 공간은 더 조밀하게 분할되나, 확장된 분산 벡터 근사 트리는 데이터 공간의 크기를 감안하여 특징 벡터를 근사함으로써 분산 hybrid spill-tree의 검색 시간을 40% 감소시켰다.



(그림 8) 데이터 크기에 따른 k-최근접 검색

우리는 데이터의 크기를 200,000 에서 1,000,000까지 증가시켜 확장된 분산 벡터 근사 트리의 성능 실험을 수행하였다. (그림 8)은 성능 실험의 결과를 나타낸다. 확장된 분산 벡터 근사 트리는 데이터의 크기의 증가와 함께 꾸준히 기존의 분산 벡터 근사 트리의 검색 시간을 단축시킴을 알 수 있다.

6. 결 론

이 논문에서 우리는 클러스터를 이루거나 편향된 분포를 가지는 실세계 이미지 데이터 집합을 위하여 데이터 분포를 감안한 새로운 고차원 색인 구조를 제안하였다. 분산 벡터 근사 트리를 포함하여 지금까지 제안된 분산 고차원 색인의 대부분은 분할이 단순하고, 분할된 공간에 균일하게 객체가 할당되는 경우의 검색 성능에 비하여, 편향되거나 클러스터를 이루는 데이터의 집합에서 성능이 많이 감소되는 경향을 보였다. 확장된 분산 벡터 근사 트리는 상위 트리의 말단 노드가 담당하는 데이터 공간의 크기에 따라 다른 길이의 비트를 할당하여 특징 벡터를 근사하였다. 이는 벡터 근사의 구분 능력, 즉 필터링 능력을 보장하는 것이다. 즉, 전체 데이터 공간에서 조밀한 지역에는 데이터 공간을 세밀히 분할하여 특징 벡터의 근사에 많은 비트를 할당함으로써 필터링 능력을 향상시키고, 밀도가 희박한 지역에는 적은 수의 비트만을 할당하도록 하였다. 그리하여, 병렬로 수행되는 VA-file의 k-최근접 검색 시간의 차이를 최소화하여 분산 벡터 근사 트리의 k-최근접 검색 성능을 향상시켰다.

참 고 문 헌

[1] C. Zhang, A. Krishnamurthy, R. Y. Wang, "SkipIndex: Towards a Scalable Peer-to-Peer Index Service for High Dimensional Data", Technical Report TR-703-04, Princeton University, 2004.

[2] B. Nam, A. Sussman, "DiST: Fully Decentralized Indexing for Querying Distributed Multidimensional Datasets", Technical Report CS-TR-4720 and UMIACS-TR-2005-28, Maryland University, 2005.

[3] H. V. Jagadish, B. C. Ooi, Q. H. Vu, et al., "VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes", ICDE, 2006.

[4] M. Bawa, T. Condie, P. Ganesan, "LSH Forest: Self-Tuning Indexes for Similarity Search", WWW, 2005.

[5] P. Haghani, S. Michel, P. Cudré-Mauroux, et al., "LSH At Large-Distributed KNN Search in High Dimensions", WebDB, 2008.

[6] N. Koudas, C. Faloutsos, I. Kamel, "Declustering Spatial Databases on a Multi-computer Architecture", EDBT, 1996.

[7] B. Schnitzer, S.T. Leutenegger, "Master-Client R-trees: A New Parallel R-tree Architecture", SSDBM, 1999.

[8] X. Fu, D. Wang, W. Zheng, M. Sheng, "GPR-tree: A Global Parallel Index Structure for Multiattribute Declustering on Cluster of Workstations", APDC, pp.300-306, 1997.

[9] T. Liu, C. Rosenberg, H.A. Rowley, "Clustering Billions of Images with Large Scale Nearest Neighbor Search", IEEE WACV, 2007.

[10] R. Weber, K. Böhm, H.-J. Schek, "Interactive-Time Similarity Search for Large Image Collection Using Parallel VA-Files", ICDE, 2000.

[11] J. Chang, A. Lee, "Parallel High-dimensional Index Structure for Content-based Information Retrieval", CIT, 2008.

[12] H.-H Choi, M.-Y. Lee, Y.-C. Kim, J.-W Chang, K.-C. Lee, "A Distributed High Dimensional Indexing Structure for Content-based Retrieval of Large Scale Data", KIISE:Databases Journal, Vol.37, No.5, pp.228-237, 2010.

[13] T. Liu, A.W. Moore, A. Gray, "An Investigation of Practical Approximate Nearest Neighbor Algorithms", ANIPS, 2004.

[14] R. Weber, H. J. Schek, S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces", VLDB, pp.194-205, 1998.

[15] P. Ciaccia, M. Patella, P. Zezula, "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces", VLDB, pp.426-435, 1997.

[16] R. Weber, S. Blott, "An Approximation-Based Data Structure for Similarity Search", Technical Report 24, ESPRIT project HERMES (No.9141), 1997.

[17] T. Yamane, Statistics: An Introductory Analysis, second ed., 1976.

[18] P. Ciaccia, M. Patella, P. Zezula, "A Cost Model for Similarity Queries in Metric Spaces", PODS, pp.59-68, 1998.

[19] R. Weber, K. Böhm, "Trading Quality for Time with Nearest-Neighbor Search", EDBT, pp.21-35, 2000.

[20] Real Data source website, http://www.autonlab.org/autonweb/15960.html.

[21] M-tree homepage, http://www-db.deis.unibo.it/research/Mtree.

최 현 화

e-mail : hyunwha@etri.re.kr
2000년 충남대학교 컴퓨터공학과(공학사)
2002년 포항공과대학교 컴퓨터공학과 (공학석사)
2002년~현 재 한국전자통신연구원 선임연구원
관심분야 : 질의 처리, 분산 병렬 처리, 이벤트 스트림 처리 등

이 규 철

e-mail : kclee@cnu.ac.kr
1984년 서울대학교 컴퓨터공학과(공학사)
1986년 서울대학교 컴퓨터공학과(공학석사)
1990년 서울대학교 컴퓨터공학과(공학박사)
1989년~1994년 IBM Almaden Research Center, 초빙 연구원
1995년~1996년 Syracuse University, CASE Center, 초빙 교수
1997년~1998년 교육부 학술진흥재단 부설 첨단학술센터, 파견 교수
1989년~현 재 충남대학교 컴퓨터공학과 교수
관심분야 : XML, 정보통합, 유비쿼터스 웹 서비스 등