수평 분할 방식을 이용한 병렬 셀-기반 필터링 기법의 설계 및 성능 평가

장 재 우[†] · 김 영 창[††]

요 약

데이터웨어하우징의 애트리뷰트 벡터나 멀티미디어 데이터베이스의 특징 벡터는 모두 고차원 데이터를 이루고 있기 때문에, 이러한 고차원 데이터를 효율적으로 검색하기 위해서는 고차원 색인 기법이 요구된다. 이를 위하여 다수의 고차원 색인 기법들이 제안되었는데, 제안된 대부분의 색인 기법들이 차원의 수가 증가할수록 검색 성능이 급격히 저하되는 '차원 저주(dimensional curse)' 문제를 지니고 있다. 셀-기반 필터링(Cell-Based Filtering : CBF) 기법은 이러한 차원 저주 문제를 해결하기 위해 제안되었다. 그러나 CBF 기법은 데이터의 양이 증가할수록 선형적으로 검색 성능이 감소하며, 이를 극복하기 위해 병렬 처리 기법을 사용하는 것이 필요하다. 본 논문에서는 데이터 디클러스터링(declustering) 방법으로 수평 분할 방식을 사용한 병렬 CBF 기법을 제안한다. 아울러 제안한 병렬 CBF 기법의 성능을 최대화하기 위하여, 병렬 CBF 기법을 다수의 서버로 구성된 Shared Nothing(SN) 구조의 클러스터 아키텍쳐 하에서 구축한다. 또한 SN 구조의 클러스터 아키텍쳐에 적합한 데이타 삽입 알고리즘, 범위질의 처리 알고리즘, k-최근접 질의 처리 알고리즘을 제시한다. 마지막으로 제안하는 병렬 CBF 기법이 기존 CBF 기법과 비교하여 서버 개수에 비례하여 우수한 검색 성능을 달성함을 보인다.

Design and Performance Analysis of a Parallel Cell-Based Filtering Scheme using Horizontally-Partitioned Technique

Jae-Woo Chang[†] · Young-Chang Kim[††]

ABSTRACT

It is required to research on high-dimensional index structures for efficiently retrieving high-dimensional data because an attribute vector in data warehousing and a feature vector in multimedia database have a characteristic of high-dimensional data. For this, many high-dimensional index structures have been proposed, but they have so called 'dimensional curse' problem that retrieval performance is extremely decreased as the dimensionality is increased. To solve the problem, the cell-based filtering (CBF) scheme has been proposed. But the CBF scheme show a linear decreasing on performance as the dimensionality. To cope with the problem, it is necessary to make use of parallel processing techniques. In this paper, we propose a parallel CBF scheme which uses a horizontally-partitioned technique as declustering. In order to maximize the retrieval performance of the proposed parallel CBF scheme, we construct our parallel CBF scheme under a SN (Shared Nothing) cluster architecture. In addition, we present a data insertion algorithm, a rage query processing one, and a k-NN query processing one which are suitable for the SN cluster architecture. Finally, we show that our parallel CBF scheme achieves better retrieval performance in proportion to the number of servers in the SN cluster architecture, compared with the conventional CBF scheme.

키워드 : 멀티미디어 정보 검색(Multimedia Information Retriebal), 내용-기반 검색(Content-based retrieval), 고차원 색인 기법(High-dimensional Index Structure)

1. 서 론

네트워크 기술의 발달과 더불어 인터넷 사용이 활발해지면서 데이터의 양은 방대해지고, 데이터의 내용도 이미지, 비디오와 같은 멀티미디어 정보를 포함하고 있으며, 대용량의 멀티미디어 데이터베이스에 대한 사용자의 다양한 요구를 지원하기 위해서 멀티미디어 데이터에 대한 내용-기반 검색이 요구된다. 아울러 데이터웨어 하우징은 웹 상에 분산되어 있는 방대한 데이터로부터 정보들 사이에 패턴과 추세를 찾아 사용자에게 가장 적합한 정보를 추출할 수 있도록 정보를 저장해 놓는 역할을 한다. 이렇게 추출된 정보는 여러 가지 속성들을 포함하고 있는 특징을 가진다. 이와 같이, 데이터웨어 하우징에 사용되는 데이터의 n-차원 속성이나 멀티미디어 데이터베이스에서 사용하는 멀티미디어 객체의 n-차원 특징 벡터들은 모두 고차원 데이터를 이루고 있으며, 이러한 고차원 데이터를 효율적으로 검색하기

※ 본 연구는 한국과학재단 특정기초연구(과제번호 : R05-2001-000-01474-0) 과제의 지원에 의해 수행되었음.
† 종신회원 : 전북대학교 컴퓨터공학과 교수
†† 준 회 원 : 전북대학교 대학원 컴퓨터공학과
논문접수 : 2002년 4월 18일, 심사완료 : 2003년 1월 6일

위해서는 고차원 색인 기법이 요구된다.

이를 위하여 정적인 데이터 집합에 대해서 공간 분할 기법을 사용하는 K-D-B-tree[1], VAMSplit k-d-tree 기법이 제안되었고[2], 동적인 데이터베이스 환경에서는 TV-tree[3], X-tree[4]와 같은 고차원 색인 기법들이 제안되었다. 그러나 제안된 대부분의 색인 기법들이 차원의 수가 증가할수록 검색 성능이 급격히 저하되는 '차원 저주(dimensional curse)' 문제를 지니고 있다[5, 6]. 따라서, 이러한 차원 저주 문제를 해결하기 위해 VA-File과 셀-기반 필터링(Cell-Based Filtering : CBF) 기법이 제안되었다. VA-File 기법은 근사 정보를 사용하여 필터링을 수행하며[7], CBF 기법은 시그니쳐를 사용하여 필터링을 수행하나 셀에 적합한 최대거리 및 최소거리를 새롭게 정의함으로써 VA-File 보다 좋은 검색 성능을 나타낸다[8]. 한편, VA-File이나 CBF 기법은 데이터의 양이 증가할수록 선형적으로 검색 성능이 감소한다, 이를 극복하기 위해 병렬 처리 기법을 사용하여 우수한 검색 성능을 제공하는 것이 필요하다.

본 논문에서는 CBF 기법에 병렬 처리 기법을 적용하여 데이터 양의 증가에 따른 선형적인 검색 성능 저하 문제를 해결한다. 이를 위해 데이터 디클러스터링(declustering) 방법으로 수평 분할 방식을 사용하는 병렬 CBF 기법을 제안한다. 아울러 제안한 병렬 CBF 기법의 성능을 최대화하기 위하여, 병렬 CBF 기법을 다수의 서버로 구성된 Shared Nothing(SN) 구조의 클러스터 아키텍처 하에서 구축한다. 이를 통해 SN 구조의 클러스터 아키텍처에 적합한 새로운 데이타 삽입 알고리즘, 범위질의 처리 알고리즘, k-최근접 질의 처리 알고리즘을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 CBF 기법 및 병렬 시그니쳐 화일 기법을 소개한다. 3장에서는 본 논문에서 제안한 수평 분할 방식을 이용한 병렬 CBF 기법에 대하여 기술한다. 4장에서는 실험을 통한 성능 평가 결과를 기술하고, 마지막으로 5장에서는 결론 및 향후 연구 방향을 기술한다.

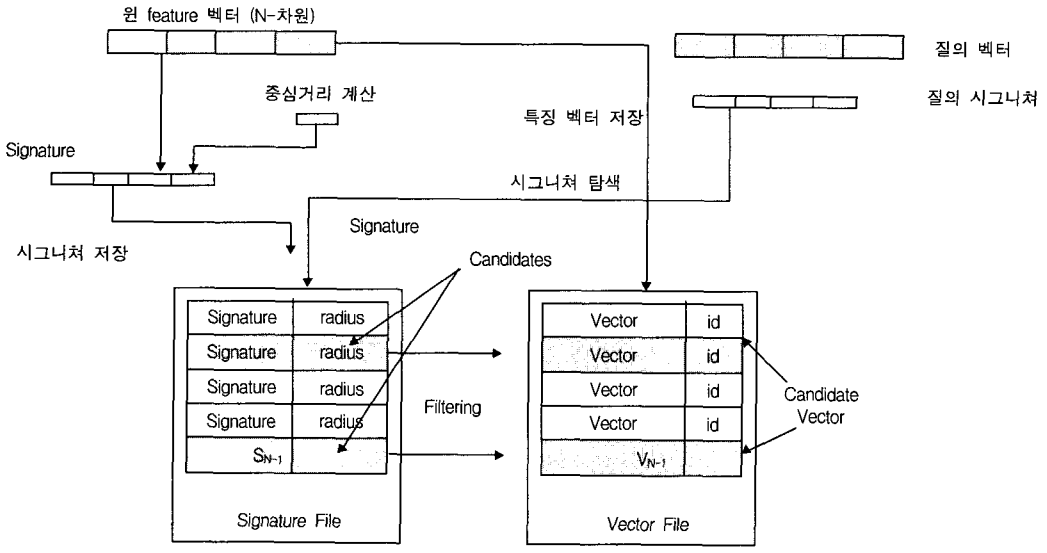
2. 관련 연구

2.1 셀-기반 필터링(Cell-Based Filtering : CBF) 기법

기존의 고차원 색인 기법들은 차원이 증가함에 따라 성능이 급격히 저하되는 차원 저주(dimensional curse) 문제를 지니고 있으며, 이러한 문제를 해결하고자 CBF 기법이 제안되었다[8]. 시그니쳐에 의한 필터링을 수행하는 CBF 기법은 객체들의 특징 벡터들에 대한 순차 탐색을 수행하기 전에, 각각의 특징 벡터에 대한 시그니쳐를 탐색하여 필터링함으로써 탐색 성능을 향상시킨다. 또한, 특징 벡터에 대한 시그니쳐 뿐만 아니라 셀 중심에서 객체까지의 거리 정보를 이용함으로써, 필터링 효과를 증가시킨다.

시그니쳐는 데이터 각각의 차원에 따른 특징 벡터의 요약들의 집합이며[9], 질의를 처리하기 위해서 질의 벡터를 시그니쳐로 표현한다. 특징 벡터를 시그니쳐로 변환하는 방법은 데이터 공간을 셀(Cell)로 나누고, 셀에 대한 시그니쳐를 사용한다. 이렇게 만들어진 시그니쳐는 각각의 셀을 대표하는 값이 되며, 셀에 대한 정보를 포함한다. 또한 차원에 따라 데이터 공간을 균등하게 분할함으로써, 셀에 대한 시그니쳐는 데이터 공간에서 특정 영역을 가지게 되며, 시그니쳐를 통해 이 범위 값을 쉽게 구할 수 있다.

이렇게 생성된 시그니쳐를 통해 질의는 다음과 같이 처리된다. 질의가 주어지면, 질의 벡터에 대한 시그니쳐 변환을 수행하여 질의 시그니쳐를 생성한다. 생성된 질의 시그니쳐를 사용하여 시그니쳐 파일에 저장된 각각의 시그니쳐들을 순차적으로 탐색하여 필터링을 수행한다. 시그니쳐를



(그림 1) CBF 기법의 전체적인 구조

통해 각 셀의 범위 값을 구하고, 이 값을 이용하여 질의 벡터의 검색 범위 안에 있는 시그니쳐들만을 다음의 검색 대상이 되는 후보 시그니쳐로 선택한다. 또한, 각 시그니쳐 정보와 함께 저장된 거리 정보를 사용함으로써 보다 효율적인 필터링을 수행한다. 시그니쳐 탐색을 통해 선택된 각 후보 시그니쳐의 실제 특징 벡터만을 데이터 파일에서 탐색함으로써 최종적으로 주어진 질의를 만족하는 특징 벡터를 검색한다. 시그니쳐와 거리 정보에 의한 필터링 효과가 클수록 검색 성능은 우수하게 된다. (그림 1)은 CBF 기법의 전체적인 구조이다.

2.2 병렬 시그니쳐 파일 기법

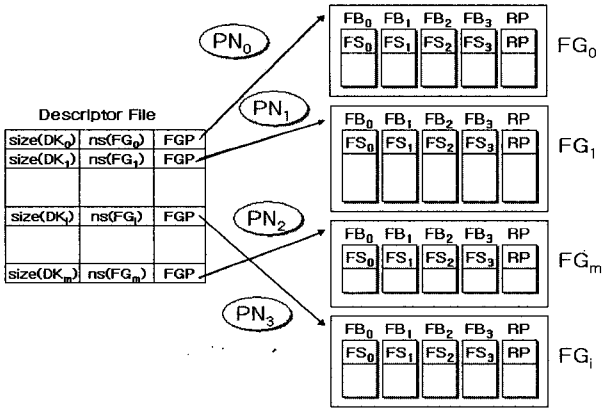
병렬 시그니쳐 파일 기법은 시그니쳐 파일을 분할하는 방법에 따라, 수평 분할 병렬 시그니쳐 파일 기법과 수직 분할 병렬 시그니쳐 파일 기법으로 구분된다.

2.2.1 수평 분할 병렬 시그니쳐 파일 기법

수평 분할 병렬 시그니쳐 방법은 각각의 프로세스가 신장 해싱(extensible hashing)을 이용하여 시그니쳐를 분할하며, 검색 효율을 높이기 위해 프레임 슬라이스 기법을 이용하여 시그니쳐 파일을 구성한다[10]. 각 프로세싱 노드(processing node) PNi는 고속의 통신망을 갖는 SN(Shared Nothing) 구조에서 다음과 같이 구성되어 있다.

$$PNi = (Pi, MMi, DAi)$$

한편, 각 프로세싱 노드 PNi는 개별적으로 신장 해싱을 통해 검색을 수행하며, 주기억 장치 MMi에는 디스크립터가 유지된다. 프레임 슬라이스 시그니쳐 파일과 레코드들은 각각의 디스크 DAi에 저장된다. (그림 2)는 수평 분할 병렬 시그니쳐 파일 기법의 프로세싱 노드 구조를 나타낸다.



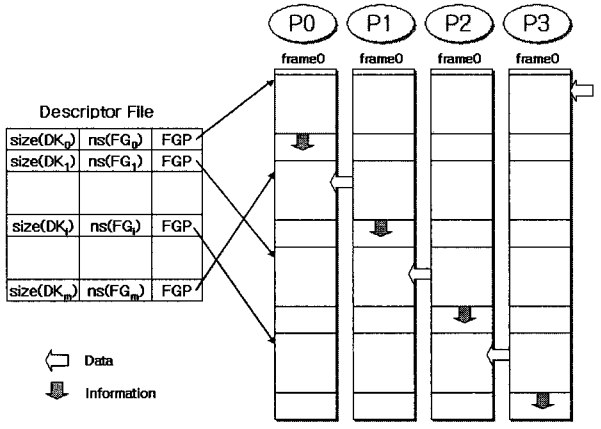
(그림 2) 수평 분할 방법의 프로세싱 노드 구조

또한, 효율적인 검색을 위하여 경험적 키 할당 알고리즘을 사용한다. 이 방법을 통하여 질의 요약이 처리될 때, 동류키들이 프로세싱 노드에 고르게 할당됨으로써 프로세스의 이용률을 높일 수 있다. 여기서 동류키(EK, equivalent keys)는 질의 요약에서 신장 해싱에 의해 만들어진 해싱키 K에 대해 다음과 같이 정의되며, 즉 디스크립터 키의 최소 키값 DK_0에서 최대 키값 DK_{n-1}까지 중에서 질의 요약의 해싱키와 같이 접근되어야 하는 키들의 집합을 의미한다. 예를 들어, 질의 요약의 해싱키가 1010이라면 동류키들의 집합은 '1*1*' 즉, 1010, 1011, 1110, 1111이 된다.

$$EK = \{ DK_i \mid (DK_i \,\&\, K),\ 0 \le i < n \}$$

2.2.2 수직 분할 병렬 시그니쳐 파일 기법

수평-분할 병렬 시그니쳐 방법은 동류키 할당이 항상 균일한 분포를 보이지 않기 때문에 실행 밀집 현상과 데이터 밀집 현상이 발생한다. 더구나 동류키를 각 프로세스에 분배해 주어야 하기 때문에 추가적인 정보 교환 시간이 필요하다. 이러한 문제점을 해결하기 위해 수직-분할 병렬 시그니쳐 파일 기법이 제안되었다[11]. 이 방법은 일정한 크기의 프레임 단위로 시그니쳐 파일을 수직으로 분할하여 데이터 밀집 현상 및 실행 밀집 현상을 제거한다. 또한, 하나의 키에 대해 모든 프로세스가 동시에 작업이 가능하도록 하여 프로세스의 효율을 증가시킨다. (그림 3)은 수직-분할 병렬 시그니쳐 파일 기법에 대한 프로세스 구조를 나타낸다.



(그림 3) 수직 분할 방법의 프로세스 구조

3. 수평 분할 방식을 이용한 병렬 CBF 기법의 설계

최근의 연구에서 트리-기반 고차원 색인 기법은 차원이 증가함에 따라 검색 성능이 급격히 저하되는 차원 저주(dimensional curse) 문제 때문에, 차원이 증가하면서 순차 탐색(Sequential Scan)보다 검색 성능이 뒤지는 결과를 초래한다는 사실이 입증되었다. 따라서 이를 해결하고자 VA-File 기법과 CBF 기법이 제안되었다. VA-File 기법은 각 셀들의 근사값(approximation)을 순차 파일에 저장하여 탐색함으로써, 차원 저주 문제를 최소화하였다. 그러나 VA-File 기법에서 사용자 질의와 셀 사이의 거리는 질의와 객체 사이의

실제 거리가 아니므로 오차 거리가 발생한다. VA-File 기법은 셀 내에서의 데이터 분포와 셀의 크기에 따라 오차 거리에 영향을 받으며, 오차 거리가 클 경우 효율적으로 탐색을 수행할 수 없다. 한편, CBF 기법은 이 오차 거리를 최소화함으로써 좀 더 효율적인 탐색 영역을 결정할 수 있는 방법이다. CBF 기법은 새로운 최소거리 및 최대거리를 사용하여 셀을 효과적으로 필터링 함으로써 검색 성능을 향상시킨다. 그러나, VA-File 기법과 마찬가지로 CBF 기법은 데이터의 양이 증가함에 따라, 시그니쳐 데이터와 특징 벡터 데이터의 양이 선형적으로 증가하므로 검색 성능이 선형적으로 감소하는 문제가 존재한다. 이러한 문제를 극복하기 위해 본 논문에서는, 기존 CBF 기법에 병렬 처리 기법을 적용하여, 데이터 양의 증가에 따른 선형적인 검색 성능 저하 문제를 해결하는 병렬 CBF 기법을 제안한다.

3.1 병렬 CBF 기법의 구조

기존 CBF 기법의 구조는 시그니쳐와 특징 벡터를 저장하고 있는 시그니쳐 파일과 특징 벡터 파일을 하나의 디스크에 저장하여 검색한다. 그러나, 병렬 CBF 기법은 데이터 디클러스터링 방법을 이용하여 다수의 서버에 시그니쳐 데이터와 특징 벡터 데이터를 분산시켜 저장한다. 한편 기존 데이터 디클러스터링 방법에는 수평 분할 방법과 수직 분할 방법이 있다. 이와 같은 두 가지 데이터 분할 방법을 병렬 CBF 기법에 적용하기 위해서는 먼저 CBF 기법의 특성을 고찰하는 것이 필요하다.

3.1.1 CBF 기법의 시그니쳐 파일 특성

CBF 기법에서 주어진 질의에 대한 검색을 위해서는, CBF의 시그니쳐 파일 전체가 순차적으로 탐색되어야 한다.

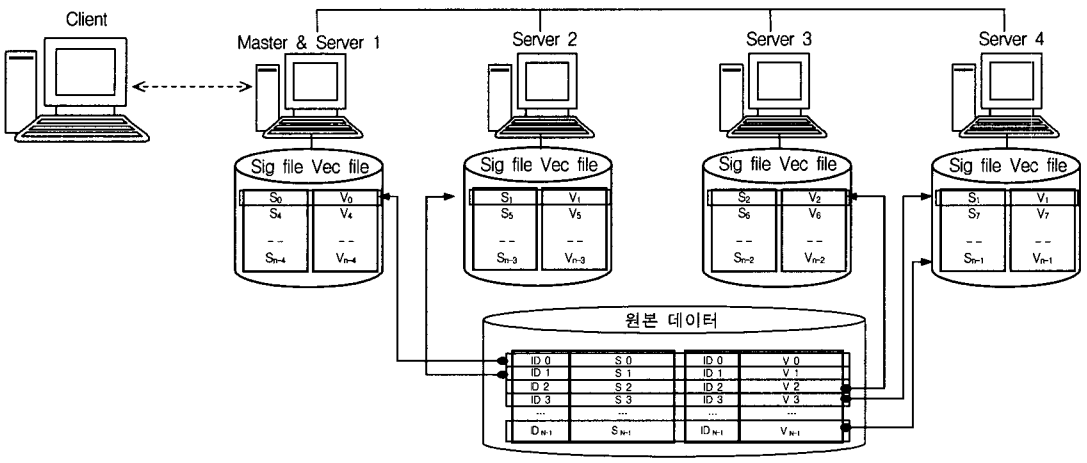
CBF 기법에 질의가 주어지면, CBF 기법은 대상 시그니쳐 파일 전체를 순차적으로 탐색한다. 따라서, 수직 분할 방법을 이용하여 D개의 서버에 시그니쳐 파일을 분산, 저장할 경우, 하나의 시그니쳐를 구성하기 위해 분할된 데이터를 병합하는 시간이 필요하다. 그러나 수평 분할 방법을 사용하여 D개의 서버에 저장한 경우, 전체 시그니쳐를 저장한 블럭 개수가 N이라 하고 데이터가 고르게 분포되어 있다고 가정할 때, 디스크 I/O 횟수는 각 서버당 약 N/D번으로 줄어들고 데이터 병합을 위한 추가비용은 필요하지 않다.

3.1.2 CBF 기법의 특징벡터 화일 특성

주어진 질의에 대한 검색을 위해서는, 시그니쳐 파일의 탐색을 통해 얻어진 후보 특징 벡터에 대해서만 CBF의 특징벡터 파일을 부분 탐색한다.

CBF에서 특징 벡터 데이터는 시그니쳐 검색 후 추출된 후보셀에 해당하는 특징 벡터, 즉, 특징 벡터 전체 데이터 중 일부분만을 탐색한다. 데이터가 수직 분할 방법으로 D개의 서버에 분산시켜 저장된 경우, 검색되어야 할 후보 특징 벡터가 임의의 위치에 저장되어 있다고 가정하면 데이터의 수가 N일 때, D×N번의 디스크 I/O 횟수가 필요하다. 그러나, 데이터가 수평 분할 방법으로 D개의 서버에 분산, 저장된 경우, 후보 특징 벡터의 수가 N이라 가정하면, 약 N번의 디스크 I/O 횟수만이 요구된다.

위의 두 특성을 통해서, 본 논문에서는 시그니쳐 파일과 특징 벡터 파일을 모두 수평 분할 디클러스터링 방법을 사용해서 다수의 서버에 분산, 저장하는 병렬 CBF 기법을 제안한다. 아울러, 제안된 병렬 CBF 기법의 성능을 최대화하기 위하여, 병렬 CBF 기법을 다수의 서버로 구성된 Shared Nothing(SN) 구조의 클러스터 아키텍쳐 하에서 구축한다. 한편 데이터를 각 서버에 분산시켜 저장할 때, 데이터를 서버 개수에 따라 모듈러(Modelar) 함수를 이용하여 저장한다. 따라서, 같은 ID의 시그니쳐와 특징 벡터 데이터는 SN 구조의 클러스터 아키텍쳐 하에서 동일한 서버에 함께 저장된다. (그림 4)는 수평 분할 디클러스터링 방법을 이용하



(그림 4) 수평 분할 방법을 이용한 병렬 CBF 기법의 구조

여 제안된 병렬 CBF 기법을 다수의 서버로($D=4$) 구성된 SN 구조의 클러스터 아키텍쳐 하에서 구축한 전체적인 구조를 나타낸다. 각 서버에는 시그니쳐 파일과 벡터파일이 분산되어 저장되고, SN 구조의 클러스터 아키텍쳐 하에서 특정한 서버는 마스터(Master)의 역할을 수행한다. 마스터는 데이터를 각 서버의 수 (D)에 따른 모듈러 함수를 통해 데이터를 분산, 저장하며, 클라이언트(client)로 부터 전송된 사용자의 질의 요청을 각 서버로 전달하여 질의를 병렬적으로 처리하는 역활을 담당한다. 질의 처리 결과는 다시 마스터로 전송되고, 전송된 결과로부터 마스터는 질의 종류에 따른 최종적인 연산을 수행하고 결과를 클라이언트에 전송한다.

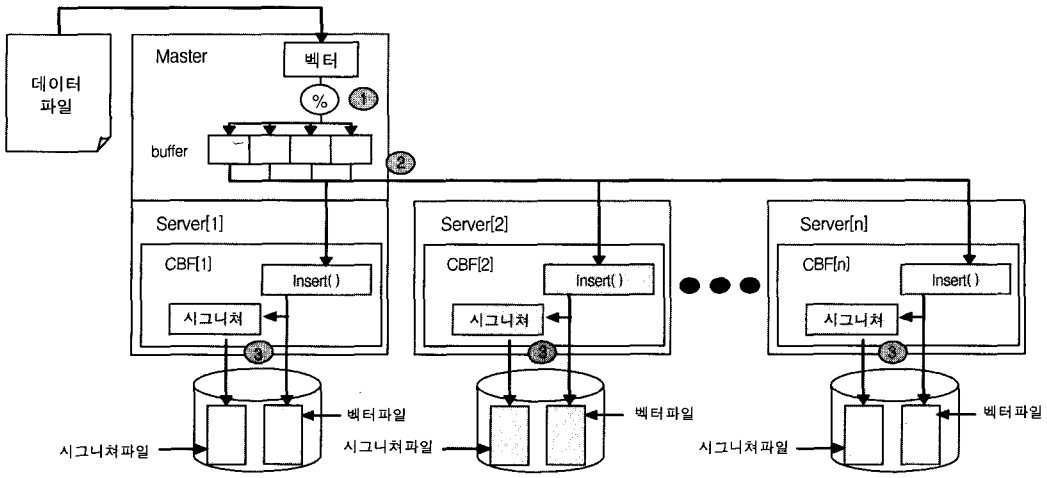
3.2 데이터 삽입

CBF 기법은 시그니쳐 파일과 특징벡터 파일의 두 개의 파일로 구성된다. 시그니쳐 파일에는 데이터의 시그니쳐 정보와 데이터가 속한 셀의 중심으로부터 객체까지의 거리에 대한 정보가 저장되고, 특징벡터 파일에는 객체의 특징 벡터와 데이터베이스로부터 부여받은 객체 식별자를 저장한다. 시그니쳐 파일을 생성할 때 삽입될 데이터들의 차원 수와 셀(cell)을 구성하는 비트 수가 정해지며, 데이터를 삽입할 때 이 비트 수를 사용해서 시그니쳐 생성 알고리즘을 통해 특징 벡터를 시그니쳐로 변환한다. 변환된 시그니쳐는 객체의 거리 값 시그니쳐와 병합되어 시그니쳐 파일에 저장되며, 객체의 특징벡터는 특징벡터 파일에 저장된다. 이때, 시그니쳐 파일과 특징벡터 파일은 다수의 서버로 구성된 SN 구조의 클러스터 아키텍쳐 하에서 수평 분할 디클러스터링 알고리즘을 통해 데이터가 각 서버에 분산되어 저장된다.

기존의 CBF 기법과는 달리 병렬 CBF 기법에서는 다수의 서버에 데이터를 병렬적으로 분산하여 저장하도록 하기 위해, 특정 서버는 마스터(Master)노드의 역할을 동시에 수행하고 모든 서버에 데이터를 보내 이를 병렬적으로 동시에 저장하도록 한다. 데이터는 삽입되는 순서에 따라 모듈러(modular) 연산을 통해 각 서버에 할당된다. 본 논문에서는 마스터(Master) 노드로부터 모든 서버에 데이터를 보내 이를 병렬적으로 동시에 저장하도록 하기위해, 소켓(socket)을 통한 네트웍 통신과 쓰레드(thread) 기법을 사용한다. 쓰레드는 프로그램에서 하나의 작업이라 말할 수 있으며, 프로세스와 개념적으로 동일하다. 차이점은 프로세스가 context 전환 시 매우 많은 데이터를 저장하여 context 전환이 무거운데(heavy weight) 반해, 쓰레드는 context 전환이 매우 가벼운(light weight) 특징을 지니고 있다. 한편 쓰레드를 여러 개 사용하면 프로그램 내에서 여러 개의 작업을 동시에 처리할 수 있으며, 정확하게 말하자면 여러 개의 쓰레드를 사용할 때 동시에 실행되는 것처럼 보이도록 각 쓰레드가 CPU 사용시간을 분배받아 조금씩 사용한다. 쓰레드는 정해진 짧은 시간동안(time slice) 자신의 일을 처리한 다음 다른 쓰레드로 CPU 사용권한을 넘겨준다. 따라서, 이를 통하여 각 서버당 하나의 쓰레드(thread)가 생성이 되고, 생성된 쓰레드는 할당받은 데이터를 버퍼에 임시저장하고, 버퍼가 차면 해당 서버로 전송해 시그니쳐 파일과 특징벡터 파일에 삽입한다. (그림 5)는 데이터 삽입 알고리즘을 나타낸다. (그림 6)은 알고리즘에 따른 전체적인 데이터 삽입 과정을 도식화한 것이다. 첫째, 데이터 파일에서 벡터 데이터를 읽어와 모듈러연산을 통해 해당 서버의 버퍼에 데이터를 삽입한다(①). 둘째, 버퍼가 차면 버퍼를 해당 서버의 CBF 인스턴스에 전송한다(②). 셋째, CBF 인스턴스의 Insert()함수는 부여받은 벡터 값으로부터 시그니쳐를 생성하며, 벡터값이 속한 셀의 중심으로부터의 거리를 구하여 시그니쳐와 함께 해당 디스크의 시그니쳐 파일에 저장하고, 벡터 데이터는 주어진 객체 식별자와 함께 벡터파일에 삽입한다(③).

```
// 데이터 삽입 쓰레드
// : 서버의 수만큼의 스레드가 생성되고 쓰레드는 데이터를 각
    서버로 전송
//  하여 저장하게 한다.
void insert_thread( )
{
    client -> callAPI (CBF_OPEN) ;

// 삽입할 데이터가 남아있는동안 각 서버로 전송할 버퍼에 데이
   터를 적재
// 버퍼가 차면 데이터를 전송하고, 삽입할 데이터가 남아있지 않
   으면
// 데이터 전송후 서버와의 연결을 해제
   while (num_of_data > 0)
   {
      while (buffer_size > 0 && num_of_data > 0)
      {
         pthread_mutex_lock (file_lock) ;
         for (int j = 0 ; j < NUM_OF_DIMENSION ; j++)
            fscanf)(fptr, "%f", vector[j]) ;
         client -> callAPI (CBF_INSERT) ;
         client -> packBuffer (vector) ;
         pthread_mutex_unlock (file_lock) ;
      }
      if (num_of_data > 0)
         client -> callAPI (SEND_BUFFER) ;
      else if (buffer_size > 0) {
         client -> packBuffer (CBF_CLOSE) ;
         client -> packBuffer (CBF_DISCONNECTION) ;
         client -> callAPI (SEND_BUFFER) ;
      } else {
         client -> callAPI (SEND_BUFFER) ;
         client -> callAPI (CBF_CLOSE) ;
         client -> callAPI (DISCONNECTION) ;
      }
   }
}
```

(그림 5) 데이터 삽입 알고리즘



(그림 6) 데이터 삽입 과정

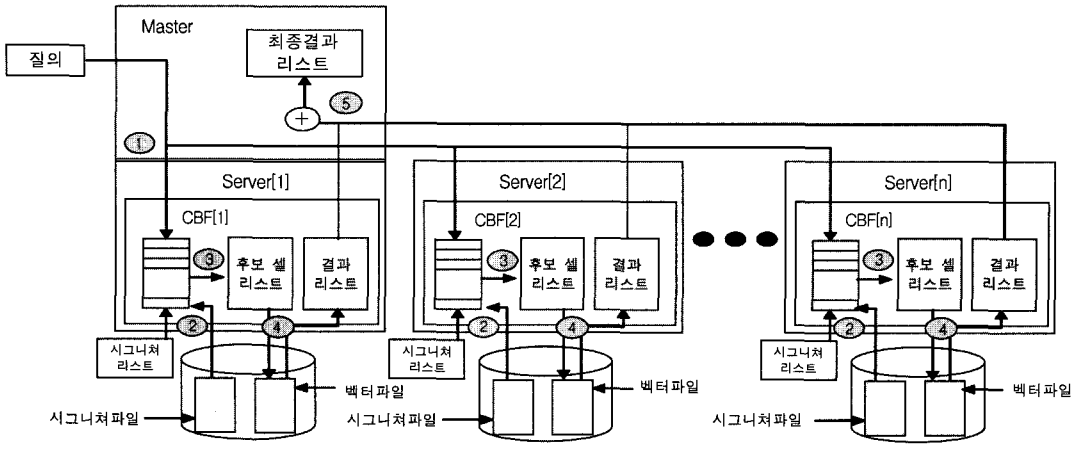
3.3 범위 질의 검색

CBF 기법은 셀의 시그니쳐 정보뿐만 아니라, 미리 계산된 셀의 중심에서 객체까지의 거리 정보를 이용함으로써 필터링 효과를 증대시킨다. CBF 기법에서는 객체의 특징 벡터에 대한 시그니쳐 뿐만 아니라 객체가 포함된 셀의 중심으로부터 객체까지의 거리를 계산하여 함께 시그니쳐 파일에 저장한다. 또한, 시그니쳐나 특징 벡터를 저장하기에 앞서, 참조 테이블(Reference Table)파일을 조사하여 파일의 적당한 위치를 찾아 저장한다. 검색을 수행할 때는 우선 질의 벡터에 대한 질의 시그니쳐를 생성하고, 시그니쳐 파일을 탐색함으로써 필터링을 수행하여 후보 셀들을 얻고, 최종적으로 데이터 파일을 접근함으로써 질의에 해당하는 결과를 얻는다.

범위 질의는 주어진 질의 점을 중심으로 임의의 거리(반경)을 갖는 영역으로써 표현되며, 그 영역에 포함되어 있는 모든 객체를 검색하는 형태로 처리된다. 이를 위해 범위 질의 영역과 겹치게 되는 모든 영역이 질의 탐색 영역에 속하게 된다. 범위 질의를 처리하기 위해 사용자는 질의 벡터 및 거리 값을 입력한다. 이렇게 입력된 값에 대하여 범위 질의는 데이터 공간상에서 사용자 질의 벡터를 중심으로 하고, 거리 값을 반경으로 하는 원(circle) 안에 포함된 모든 객체를 찾는다. 사용자로부터 질의 벡터와 거리 값이 주어지면 질의 벡터를 시그니쳐로 변환하고 변환된 시그니쳐로부터 주어진 거리 값 영역안에 해당하는 셀을 찾는다.

기존 CBF 기법이 사용자 질의시 시그니쳐 파일, 특징 벡터 파일 각각 한 개씩 접근해 탐색하는 것과는 달리, 본 논문에서 설계한 병렬 CBF 기법에서의 범위 질의처리 알고리즘은 다수의 서버로 구성된 SN 구조의 클러스터 아키텍쳐 하에서 각 서버에 데이터 파일이 분산, 저장되어 있으므로 다수의 서버에 동시에 접근해서 탐색하는 기술이 필요하다. 이를 위해 2.2절과 마찬가지로 소켓(socket)을 통한 네트웍 통신과 쓰레드(thread) 기법을 사용한다. 이를 통하여 사용자 질의가 주어지면 각 서버당 쓰레드가 생성되고, 이들 쓰레드는 질의를 해당 서버로 전송하여 각 서버에서 동시에 병렬적으로 각 데이터 파일을 탐색하고, 내부적으로 질의에 맞는 객체를 검색한다. 따라서 같은 양의 데이터가 분산 저장되어 있는 환경에서, 병렬 CBF 기법은 다수의 서버로 구성된 SN 구조의 클러스터 아키텍쳐 하에서 범위 질의를 동시에 병렬적으로 수행함으로써, 전체적인 검색 성능의 향상을 가져온다. (그림 7)은 병렬 CBF 기법에서 쓰레드를 이용한 범위 질의처리 알고리즘을 나타낸다. (그림 8)은 알고리즘에 따른 범위 질의 처리 과정을 도식화한 것이다. 첫째, 질의 파일로부터 읽어온 질의를 각 서버의 CBF 인스턴스에게 전달한다(❶). 둘째, 서버의 CBF 인스턴스는 시그니쳐 파일을 읽어서 시그니쳐 리스트를 생성한다(❷). 셋째, 질의 벡터 데이터로부터 질의 시그니쳐를 생성하고, 시그니쳐 리스트를 검색하여 후보 셀 리스트를 생성한다(❸). 셋째, 생성된 후보 셀 리스트에 존재하는 데이터만을 벡터 파일로부터 검색하여 결과 리스트를 생성하고, 이를 마스터노드에게 전송한다(❹). 마스터노드는 각 서버로부터 전송받은 결과 리스트를 합병하여 최종 결과 리스트를 생성한다(❺).

```
// 범위질의 과정
// : 서버 수만큼의 스레드가 생성되고 각 스레드는 질의 벡터를 서
    버로
// 전송하여 범위 질의를 수행하게 한다. 검색된 결과는 다시 스레드
// 로 전송되고 getFinalResult 함수를 통해 최종 결과를 얻는다
void main()
{
   pthread_t *thread ;
   FILE fptr = fopen (query_filename, "rt") ;
   for (int i = 0 ; i < NUM_OF_DIMENSION ; i++)
      fscanf)(fptr, "%f", vector[i]) ;
   for(int j = 0 ; j < NUM_OF_SERVER ; j++)
```



(그림 8) 범위 질의 처리 과정

```
        pthread_create (&thread[i], NULL, range_thread) ;
    getFinalReault( ) ;
}
// 범위 질의 처리 스레드
void range_thread (float *vector, float *result)
{
    client -> callAPI (CBF_OPEN) ;
// 각 서버로 범위 질의를 알리는 메시지 전송후 질의 벡터를 전송한다.
// 서버에서는 범위 질의에 대한 결과를 전송하고 result에 저장된다.
    client -> callAPI (CBF_RANGE) ;
    client -> packBuffer (vector) ;
    client -> callRetrieve (result) ;
    client -> callAPI (CBF_CLOSE) ;
    client -> callAPI (DISCONNECTION) ;
}
```

(그림 7) 범위 질의 처리 알고리즘

3.4 k-최근접 질의 검색

k-최근접 질의는 데이터 공간상에서 사용자의 질의 벡터와 가장 가까운 k개의 객체를 찾는 질의이다. CBF 기법에서는 이를 위해 새로운 개념의 upper bound(MAXDIST) 및 lower bound(MINDIST)를 사용한다[8, 12]. CBF 기법에서는 객체의 특징 벡터에 대한 시그니쳐 데이터를 저장하기 전에 셀의 중심과 객체의 거리(r)를 계산하여 함께 저장되며, 거리 값 r은 검색을 수행할 때 사용된다. CBF 기법에서의 lower bound는 질의 포인트(Q)와 셀 중심(C)과의 거리와 거리 값 r과의 차이며, upper bound는 질의 포인트(Q)와 셀 중심(C)과의 거리와 거리값 r과의 합이다. 이러한 새로운 lower bound, upper bound를 사용함으로써 탐색 알고리즘의 첫 번째 단계에서 더 많은 후보 셀들을 필터링하며, 마지막 단계에서의 디스크 I/O를 줄일 수 있다.

병렬 CBF 기법의 k-최근접 질의 검색 알고리즘은 기존 CBF 기법의 알고리즘과 마찬가지로 질의가 주어지면 세 단계의 필터링 단계를 거쳐서 수행된다. 그러나 시그니쳐 파일과 벡터 파일이 다수의 서버로 구성된 SN 구조의 클러스터 아키텍쳐 하에서 각 서버에 분산, 저장되어 있기 때문에 질의를 각 서버로 전달하고 각 서버에서 얻어지는 k개의 결과를 전달받아 통합하여 최종적인 k개의 결과 리스트를 통합하기 위한 추가적인 과정이 필요하다. (그림 9)는 병렬 CBF 기법에서 쓰레드를 이용한 k-최근접 질의 처리 알고리즘을 나타낸다. (그림 10)은 알고리즘에 따른 k-최근접 질의의 전체 처리 과정을 도식화한 것이다. 첫 번째 단계에서는 각 서버당 쓰레드를 생성하고 각 쓰레드에 질의 벡터를 전달한다(①). 쓰레드는 해당 서버에 k-최근접 질의를 알리는 메시지를 전송해 준비를 시킨 후 질의 벡터를 전송한다. 각 서버에서는 CBF 인스턴스가 시그니쳐 파일에 저장된 전체 시그니쳐들을 읽어 시그니쳐 리스트(sigbuf_list)를 생성하고(②), 질의벡터를 통해 이를 검색하여 1차 후보 리스트를 생성한다(③). 먼저 처음 k개의 시그니쳐가 삽입될 때까지는 각 디스크로부터 검색된 시그니쳐들을 후보리스트에 디스크 순서대로 삽입한다. 일단 k개의 리스트가 만들어지면, 다음 검색한 시그니쳐와 후보리스트의 마지막 k번째의 후보와 비교를 하고, 검색된 시그니쳐가 더 작은 거리값을 가지면 k번째의 후보를 삭제하고 그 자리에 검색된 시그니쳐를 삽입하며, 그렇지 않은 경우는 다음 디스크로부터 검색된 시그니쳐와 비교를 계속한다. 만약, 더 이상 검색된 시그니쳐가 없으면 두 번째 단계로 넘어간다.

두 번째 단계는 첫 번째 단계에서 얻어진 후보 셀 리스트 중에서 최종적으로 얻어진 k-번째 upper bound 값(k_max-dist)을 사용하여 이 값보다 큰 lower bound 값을 갖는 셀들을 후보 리스트부터 제거하고 최종 후보 셀 리스트를 만들어 마지막 단계에서의 불필요한 디스크 I/O 횟수를 줄인다(④).

k-최근접 탐색 알고리즘의 마지막 단계는 두 번째 단계에서 얻어진 최종 후보리스트로부터 실제 벡터 파일을 검색해 결과 리스트를 만드는 과정이다(⑤). 세 번째 단계에서는 필터링을 수행하기 위해 최종 후보 셀의 lower bound

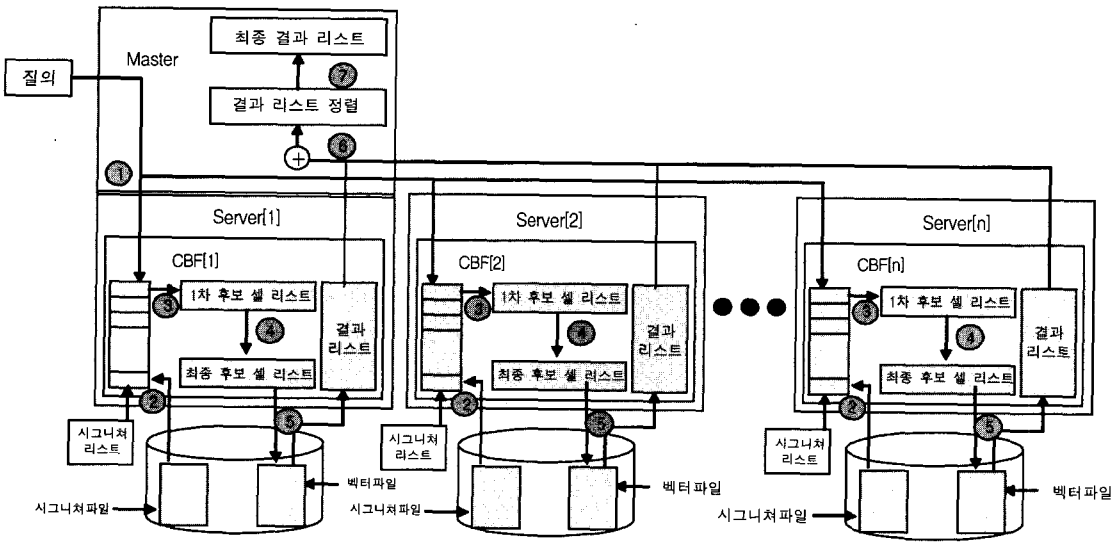
값과 현재까지 얻어진 k-번째 객체 거리값(k_dist)을 비교하여 k_dist보다 크면 결과 리스트로부터 제거하고, 작으면 데이터 파일을 접근하여 질의 포인트와 실제 객체와의 거리를 구하여 비교한 후, k개의 가장 가까운 객체를 검색하여 결과리스트를 만들고 이를 마스터노드에 전송한다. 서버의 수를 N이라 할 때 마스터노드가 각 서버로부터 전송받는 결과의 수는 k×N이 된다(⑥). 따라서 k개의 최종 결과리스트를 만들기 위해 마스터노드는 서버로부터 전송된 결과리스트를 통합하고 이를 정렬하여 질의벡터로부터 가장 가까운 k개의 최종 결과리스트를 만든다(⑦).

```
// k-최근접 질의
// : 서버 수만큼의 스레드가 생성되고 각 스레드는 질의 벡터를 서
    버로 전송하여
// k-최근접 질의를 수행하게 한다. 각 서버에서 검색된 k개의 결과
   는 다시
// 스레드로 전송되고 getFinalResult 함수를 정렬되고 그중 k개의
   최종 결과를
// 얻는다
void main()
{
   pthread_t *thread ;
   FILE fptr = fopen (query_filename, "rt") ;
   for (int i = 0 ; i < NUM_OF_DIMENSION ; i++)
       fscanf)(fptr, "%f", vector[i]) ;
   for(int j = 0 ; j < NUM_OF_SERVER ; j++)
       pthread_create (&thread[i], NULL, knn_thread) ;
   getFinalReault( ) ;
}

// k-최근접 질의 처리 스레드

void knn_thread (float *vector, float *result)
{
   client -> callAPI( CBF_OPEN) ;

// 각 서버로 k-최근접 질의를 알리는 메시지 전송후 질의 벡터를
   전송한다.
// 서버에서는 1차 pruning을 통해 주어진 질의 벡터로 시그니처를
   검색하여 1차
// 후보리스트를 얻고, 2차 pruning을 통해 2차 후보리스트를 만든
   다. 3차
// pruning에서는 2차 후보리스트를 정렬하고 실제 데이터 파일을
   검색하여 질의
// 벡터에서 가장 가까운 k개의 결과리스트를 만들고 이를 전송하
   여 result에
// 저장한다.

 client -> callAPI (CBF_KNN) ;
 client -> packBuffer (vector) ;
 client -> callRetrieve (result) ;
 client -> callAPI (CBF_CLOSE) ;
 client -> callAPI (DISCONNECTION) ;
}
```

(그림 9) k-최근접 질의 처리 알고리즘

4. 성능 평가

본 장에서는 제안하는 병렬 CBF 기법을 구현하여 성능 평가를 수행한다. 구현된 시스템 환경은 <표 1>와 같다. 실험 데이터는 200만건의 Synthetic 데이터 셋 각각 10차원, 20차원, 40차원, 50차원, 60차원, 80차원 100차원과 Hadamada 알고리즘을 이용한 리얼(real) 데이타 20만건의 10차원, 20차원, 50차원, 80차원을 사용하였으며, 성능 비교 대상은 기존 CBF 기법과 본 논문에서 제안한 병렬 CBF 기법이다. 아울러 병렬 CBF 기법은 4대의 서버로 구성된 Shared Nothing(SN) 구조의 클러스터 아키텍처하에서 구축되었다. 아울러 성능 평가는 데이터 삽입 시간, 범위 질의 검색시간, k-최근접 질의 검색시간에 대해 수행한다.



(그림 10) k-최근접 질의 처리 과정

〈표 1〉 실험적 성능평가 환경

시스템 환경	450 MHZ CPU HDD 30GB * 4대 128 MB Memory	
구현 환경	Redhat Linux 7.0 (kernel 2.4.5) gcc 2.96 (g++)	
실험 데이타	Syntheic 데이타	200만건(10, 20, 40, 50, 60, 80, 100차원)
	리얼 데이타	20만건(10, 20, 50, 80차원)
검색 시간 측정 기준	범위 질의	전체 데이터의 0.1% 검색에 소요되는 시간
	k-최근접 질의	질의 벡터에 가장 가까운 100개의 데이터 검색에 소요되는 시간

4.1 분석적 검색 성능 지수

〈표 2〉 분석적 성능 인자

인 자	설 명
Ls	시그니처의 길이 (비트수)
Lv	벡터의 길이 (비트수)
N	데이터 수
b	시그니처 비트수
d	차원
P	페이지 크기 (8kbyte)
D	디스크 수 (4, 8, 12, 16)
r	셀 중심으로부터 데이터까지의 거리
Ci	i번째 디스크의 후보 셀 수
Cmax	디스크중 가장 많은 후보 셀 수
Rs	시그니처 검색 I/O 수
Rv	벡터파일 검색 I/O 수
R	총 디스크 I/O 수

본 논문에서 제안한 병렬 CBF 기법의 분석적인 성능 지수를 측정한다. 이는 병렬 CBF 기법의 성능 향상 지수를 이론적인 성능 향상 지수와 비교하기 위함이며, 성능향상 지수 측정에 쓰인 인자는 <표 2>와 같다. 식 (1)은 주어진 질의에 대하여, 검색을 위한 디스크 I/O 횟수를 계산하기 위한 식이다. 분석적인 성능 지수를 측정하기 위해 전체 디스크 I/O 횟수는 R로 표현한다. 전체 디스크 횟수 R은 Rs와 Rv의 합으로 계산되는데, Rs는 병렬 CBF 기법에서 사용자 질의시 시그니처 파일을 탐색하기 위한 디스크 I/O 횟수이고, Rv는 특징 벡터 파일을 탐색하기 위한 디스크 I/O 횟수이다. 아울러 전체 검색성능은 디스크 I/O 횟수에 전적으로 의존한다고 가정하며, 데이타는 각 서버에 고르게 분산되어 있다고 가정하며, 데이터는 고르게 분산되었다고 가정한다.

$$R = R_s + R_v \qquad (1)$$

$$R_s = \left(\frac{L_s \times N}{P}\right)/D \quad L_s = (b \times d \times N) + r$$

$$R_v \approx \max(C_i) \qquad (\forall i,\ 0 \le i < D)$$

4.2 삽입 시간

삽입 시간은 200만건의 Synthetic 데이터를 병렬 CBF 기법을 사용하여 삽입하는데 소요되는 시간과, 20만건의 리얼 데이터를 삽입하는 데 소요되는 시간을 측정한다. <표 3>은 Synthetic 데이타에 대하여 기존 CBF 기법과 병렬 CBF 기법의 삽입시간을 나타낸다. <표 3>에서 보듯이, 기존 CBF 기법은 10차원의 경우 236초 정도가 소요되며, 20차원은 478초, 100차원의 경우에는 1862초가 소요된다. 반면, 병렬 CBF 기법은 10차원의 경우 256초, 20차원은 445초, 100차원의 경우 1930초가 소요된다. 한편, <표 4>은 리얼 데이타에 대하여 기존 CBF 기법과 병렬 CBF 기법의 삽입시간을 나타낸다. 기존 CBF에서는 10차원의 경우 약 25초, 80차원의 경우 129초를 요구하며, 병렬 CBF 기법은 10차원의 경우 약 28, 80차원의 경우 131초로 Synthetic 데이타와 거의 비슷한 결과를 나타낸다.

이러한 결과로 볼 때, 기존 CBF 기법과 병렬 CBF 기법이 삽입시간 측면에서 비슷한 성능을 보이는데, 그 이유는 다음과 같다. 즉, CBF 기법에서 파일을 생성할 때 입력 디스크로 부터 벡터 데이타를 읽어들이는 시간과 이를 새로운 파일로 쓰는 시간이 필요하다. 이때 읽는 시간은 동일하며, 쓰는 시간이 여러 대의 서버에서 동시에 수행되기 때문에 병렬 CBF 기법의 삽입시간은 줄어들 수 있다. 그러나 병렬 CBF 기법은 네트웍을 이용한 삽입을 수행하기 때문에 통신 오버헤드가 존재한다. 이러한 이유 때문에 전체적으로 기존 CBF 기법과 병렬 CBF 기법은 삽입시간이 거의 동일한 결과를 보인다. 한편, 마스터 노드와 서버간의 데이터 교환시 네트웍이 허용하는 충분한 버퍼를 할당하고 고속의 네트웍이 가능하도록 하면, 통신 오버헤드를 줄일 수 있다.

〈표 3〉 Synthetic 데이터 200만건의 삽입 시간

(단위 : 초)

기법 \ 차원수	10차원	20차원	40차원	50차원	60차원	80차원	100차원
기존 CBF	236.50	478.89	828.49	992.78	1170.40	1505.90	1862.16
병렬 CBF	256.12	445.42	824.02	1001.93	1188.42	1543.62	1930.10

〈표 4〉 리얼 데이터 20만건의 삽입 시간

(단위 : 초)

기 법 \ 차원수	10차원	20차원	50차원	80차원
기존 CBF	24.61	42.87	85.54	128.54
병렬 CBF	27.93	36.60	85.80	131.84

4.3 범위 질의 검색 시간

범위 질의는 주어진 질의 포인트로부터 일정한 거리 영역 안에 포함된 객체를 찾는 검색이다. 범위 값은 전체 데이터 중에서 0.1%의 데이터를 검색할 수 있는 값을 범위 값으로 사용한다. <표 5>는 Synthetic 데이터 200만건에 대하여 범위 질의 검색 시간을 기존 CBF 기법과 병렬 CBF 기법에서 각 차원별로 측정한 것이다. 여기서 성능향상 지수는 기존 CBF 기법에 비해 병렬 CBF의 성능 향상치를 수치적으로 나타낸 것으로써 다음 식 (2)에 의해 구할 수 있다.

$$\frac{1}{\frac{PT}{CT}} * 100 \qquad (2)$$

여기서 PT는 병렬 CBF 기법의 성능 측정치(시간)이며, CT는 기존 CBF 기법의 성능 측정치이다. <표 5>에 나타난 바와 같이, 데이터의 차원수가 기존 CBF 기법은 10차원일 때 13초, 20차원 16초, 50차원 38초, 80차원 60초를 보이며, 병렬 CBF 기법에서는 10차원일 때 1.6초, 20차원 2.2초, 50차원 7.3초, 80차원 11.6초를 보인다. 실험결과에서 볼 수 있듯이 고차원으로 갈수록 약 520% 내외의 성능향상이 있음을 알 수 있다. 이런 결과를 보이는 이유는 SN 구조의 클러스터 아키텍쳐하에서 병렬 CBF 기법을 구축한 결과로써, 이는 용량이 큰 파일을 D개 서버로 나누었을 때 디스크를 접근하는 디스크 I/O수는 1/D로 줄어들며, 버퍼 방식을 적용할 경우 디스크 I/O수는 더욱 줄어들기 때문이다. 한편, <표 6>는 리얼데이터 20만건에 대하여 범위 질의 검색 시간을 비교한 것이다. 기존 CBF의 경우 20차원은 0.49초, 80차원은 1.73초를 보이는 반면에, 병렬 CBF의 경우 20차원은 0.19초, 80차원은 0.61초를 보인다. 즉 리얼 데이타에 대해서는 고차원으로 갈수록 약 280% 내외의 성능 향상이 있음을 알 수 있다. 이와 같이 데이터를 사용한 병렬 CBF 기법의 성능향상 지수에 비해 리얼 데이터를 이용한 성능향상 지수가 낮은 이유는, 리얼 데이터의 경우 Synthetic 데이타에 비해 상대적으로 적은 데이타 셋을 사용하며, 이를 통해 통신 오버헤드의 영향에 민감하기 때문이다.

〈표 5〉 Synthetic 데이터 200만건의 범위질의 검색 시간

(단위 : 초)

기법 \ 차원수	10차원	20차원	40차원	50차원	60차원	80차원	100차원
기존 CBF	13.20	16.42	31.91	37.93	44.96	59.50	74.27
병렬 CBF	1.59	2.24	5.92	7.27	7.87	11.60	14.04
성능향상 지수	830%	733%	539%	522%	571%	513%	529%

〈표 6〉 리얼 데이터 20만건의 범위질의 검색 시간

(단위 : 초)

기법 \ 차원수	10차원	20차원	50차원	80차원
기존 CBF	0.49	0.49	1.08	1.73
병렬 CBF	0.30	0.19	0.39	0.61
성능향상 지수	166%	266%	278%	285%

4.4 k-최근접 질의 검색 시간

k-최근접 질의는 주어진 질의 벡터로부터 가장 가까운 k개의 객체를 찾는 검색이다. 본 성능평가에서는 k값을 100으로 정하여 가장 가까운 100개의 객체를 검색하는 시간을 측정한다. <표 7>는 각각 Synthetic데이타 200만건을 사용하여, 기존 CBF 기법과 병렬 CBF 기법의 k-최근접 질의 검색 시간을 나타낸다. <표 7>에서 볼 수 있듯이, 데이터의 차원수가 10차원일 때, 기존 CBF기법은 1.44초, 병렬 CBF 기법은 3.59초로 병렬 CBF 기법이 기존 CBF 기법과 비교하여 249%정도의 성능향상을 보이며, 이는 입력 데이터를 다수의(D = 4) 서버로 수평 분할하여 저장함에 따라 디스크 I/O 수가 크게 줄어들기 때문이다. 한편 40차원의 경우 기존 CBF 기법이 31.92초, 병렬 CBF 기법이 8.28초로 386%의 성능향상이 있으며, 아울러 100차원인 경우 기존 CBF 기법이 76.39초, 병렬 CBF 기법이 25.62초로 298%의 성능향상을 나타낸다. 이는 범위질의 검색 시간의 성능향상보다는 적지만 거의 300% 정도의 성능향상을 보임을 알 수 있다. 한편, <표 8>은 리얼 데이타 20만건을 사용하여 측정한 k-최근접 질의 검색 시간이다. 기존 CBF 기법은 10차원 0.29초, 50차원 1.57초의 검색시간을 보이는 반면, 병렬 CBF 기법은 10차원에서 0.36초, 50차원 0.91초의 검색시간을 보인다. 즉 기존 CBF 기법에 비해 약 150% 내외의 성능향상을 보임을 알 수 있다.

〈표 7〉 Synthetic 데이터 200만건의 k-최근접 질의 검색 시간

(단위 : 초)

기법 \ 차원수	10차원	20차원	40차원	50차원	60차원	80차원	100차원
기존 CBF	3.59	17.80	31.92	39.28	46.72	61.60	76.39
병렬 CBF	1.44	2.62	8.28	10.78	11.02	18.97	25.62
성능향상 지수	249%	679%	386%	364%	424%	325%	298%

〈표 8〉 리얼 데이터 20만건의 k-최근접 질의 검색 시간

(단위 : 초)

기법 \ 차원수	10차원	20차원	50차원	80차원
기존 CBF	0.29	0.72	1.57	2.38
병렬 CBF	0.36	0.46	0.91	1.60
성능향상 지수	82%	159%	172%	149%

한편 범위질의와 비교할 때 k-최근접 질의 검색 성능이

저조한 이유는, 저장된 벡터 데이타의 분포에 따라 각 서버마다 k 개의 최근접 벡터 데이터를 검색하는 시간이 가변적이기 때문이다. 즉, 병렬 CBF 기법에서 k-최근접 질의를 처리함에 있어 다른 서버에 비해 상대적으로 느린 서버가 전체 성능을 좌우하기 때문이다. 아울러 k-최근접 질의를 처리함에 있어 클러스터 내의 서버의 수가 D개일 때 검색 시간이 1/D로 줄어들지 않는 이유는, k-최근접 질의의 특성상 각 서버마다 k개의 최근접 벡터를 찾고, 아울러 마스터 노드에서 이를 통합하여 최종적인 k개의 최근접 벡터를 재계산하는 시간이 부가적으로 필요하기 때문이다. 특히 10차원의 경우와 같이 데이터 양이 적을 경우, 이러한 부가적인 시간이 큰 오버헤드가 된다.

5. 결 론

유사성에 기반한 검색 질의는 데이터베이스에 저장된 대량의 멀티미디어 객체들 중에서 사용자가 원하는 객체와 유사한 객체를 찾는 검색이다. 유사성 기반 검색을 위해 멀티미디어 객체로부터 n(>1)개의 특징 벡터(feature vector)를 추출하여 데이터베이스에 저장한 다음, 사용자가 검색하려고 하는 멀티미디어 객체의 특징 벡터와 가장 유사한 값을 갖는 객체를 찾는다. 범위 질의, k-최근접 탐색 질의와 같은 유사성에 기반한 검색 질의는 내용-기반 검색에서 매우 중요하며, 기존의 데이터베이스에서의 검색 질의에 비해 매우 복잡한 형태를 갖는다. 기존의 고차원 색인 구조는 멀티미디어 객체로부터 추출된 특징벡터의 차원이 증가함에 따라 검색 성능은 급격히 저하된다. 이러한 차원의 증가에 따라 발생하는 기존의 고차원 색인 기법의 비효율성을 극복하기 위해 CBF 기법이 제안되었다. CBF 기법에서 검색 성능은 시그니쳐 파일과 특징 벡터 파일을 탐색하기 위한 디스크 접근 횟수에 의존한다. 한편, 데이터의 양이 증가하고, 차원이 증가할수록 전체 시그니쳐 파일의 크기와 특징 벡터 파일의 크기는 선형적으로 증가하므로 검색 성능을 저하시키는 결과를 가져온다.

본 논문에서는 CBF 기법의 성능향상을 위해, 다수의 서버로 구성된 Shared Nothing 구조의 클러스터 아키텍쳐 하에서 데이터를 각 서버에 분산시켜 저장 및 검색하는 병렬 CBF 기법을 제안하였다. 아울러, 시그니쳐 파일은 전체 탐색되어야 하고, 특징 벡터 파일은 시그니쳐 파일의 탐색을 통해 얻어진 후보 특징 벡터만 부분적으로 탐색되어지는 특성을 고려하여 시그니쳐와 특징 벡터 데이터를 수평 분할 방법을 이용하여 분산시켜 저장하였다. 아울러 본 논문에서 제안한 병렬 CBF 기법을 기존의 CBF 기법과 성능 비교를 수행하였다. 성능 평가 결과 제안한 병렬 CBF 기법이 범위 및 k-최근접 질의에 대해 클러스터 아키텍쳐 하에서의 서버의 수에 비례하여 거의 선형적인 성능 향상을 달성하였다.

향후 연구로는 데이터 웨어하우징이나 멀티미디어 데이터베이스 응용의 고차원 색인 기법으로서의 효율성을 증명하기 위해, 제안한 병렬 CBF 기법을 VA-file 의 병렬화 기법과 성능 비교를 수행하는 것이다.

참 고 문 헌

[1] J. T. Robinson, "The K-D-B-tree : A Search Structure for Large Multidimensional Dynamic Indexes," Proc. ACM SIGMOD Int. Conf. on Management of Data, pp.10-18, 1981.

[2] D. A. White and R. Jain, "Similarity Indexing : Algorithms and Performance," In Proc. of the SPIE : Storage and Retrieval for Image and Video Databases IV, Vol.2670, pp.62-75, 1996.

[3] H. I. Lin, H. Jagadish and C. Faloutsos, "The TV-tree : An Index Structure for High Dimensional Data," VLDB Journal, Vol.3, pp.517-542, 1995.

[4] S. Berchtold, D. A. Keim, H-P. Kriegel, "The X-tree : An Index Structure for High-Dimensional Data," Proceedings of the 22nd VLDB Conference, pp.28-39, 1996.

[5] S. Arya, D. M. Mount, O. Narayan, "Accounting for Boundary Effects in Nearest Neighbor Searching," Proc. 11th Annaual Symp. on Computational Geometry, Vancouver, Canada, pp.336-344, 1995.

[6] Berchtold S., Bohm C., Keim D., Kriegel H.-P, "A Cost Model for Nearest Neighbor Search in High-Dimensional Data Space," ACM PODS Symposium on Principles of Databases Systems, Tucson, Arizona, 1997.

[7] Roger Weber, Hans-Jorg Schek, Stephen Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," Proceedings of 24rd International Conference on Very Large Data Bases, pp.24-27, 1998.

[8] S.-G. Han and J.-W. Chang, "A New High-Dimensional Index Structure Using a Cell-based Filtering Technique," In Lecture Notes in Computer Science 1884(Current Issues in Databases and Information Systems), Springer, pp.79-92, 2000.

[9] C. Faloutsos, "Design of a Signature File Method that Accounts for Non-Uniform Occurrence and Query Frequencies," ACM SIGMOD, pp.165-170, 1985.

[10] J.-K. Kim and J.-W. Chang, "Horizontally-divided Signature File on a Parallel Machine Architecture," Journal of Systems Architecture, Vol.44, No.9-10, pp.723-735, June, 1998.

[11] J.-K. Kim and J.-W. Chang, "Vertically-partitioned Parallel Signature File Method," Journal of Systems Architecture, Vol.46, No.8, pp.655-673, June, 2000.

[12] N. Roussopoulos, S. Kelley, F. Vincent, "Nearest Neighbor Queries," Proc. ACM Int. Conf. on Management of Data (SIGMOD), pp.71-79, 1995.

장 재 우

e-mail : jwchang@dblab.chonbuk.ac.kr
1984년 서울대학교 전자계산기공학과 (공학사)
1986년 한국과학기술원 전산학과(공학석사)
1991년 한국과학기술원 전산학과(공학박사)
1996년~1997년 Univ. of Minnesota, Visiting Scholar.
1991년~현재 전북대학교 컴퓨터공학과 부교수
관심분야 : 멀티미디어 데이타베이스, 멀티미디어 정보검색, 하부저장구조

김 영 창

e-mail : yckim@dblab.chonbuk.ac.kr
2001년 전북대학교 컴퓨터공학과(공학사)
2001년~현재 전북대학교 컴퓨터공학과 석사과정
관심분야 : 데이터베이스, 컴퓨터시스템, 분산 및 병렬처리