효율적인 병렬 고차원 색인구조 설계
(Design of an Efficient Parallel High-Dimensional Index Structure)

박 춘 서 [†] 송 석 일 [††] 신 재 룡 [††] 유 재 수 [†††]
(Choon Seo Park) (Seok Il Song) (Jae Ryong Shin) (Jae Soo Yoo)

요 약 일반적으로 이미지나 공간 데이타베이스와 같은 다차원의 특징을 갖는 데이타들은 대용량의 저장공간을 요구한다. 이 대량의 데이타를 하나의 워크스테이션에 저장하고 검색을 수행하는 데는 한계가 있다. 최근 활발히 연구되고 있는 병렬 컴퓨팅 환경에서 이들에 대한 저장 및 검색을 수행한다면 훨씬 더 높은 성능 향상을 가져 올 수 있을 것이다. 이 논문에서는 기존에 존재하는 병렬 컴퓨팅 환경의 장점을 최대한 이용하는 병렬 고차원 색인구조를 제안한다. 제안하는 색인구조는 nP(프로세서)-nD(디스크)와 1P-nD의 결합 형태인 nP-n×mD의 구조라고 볼 수 있다. 노드 구조는 팬-아웃을 증가시키고 트리의 높이를 줄일 수 있도록 설계되었다. 또한 I/O의 병렬성을 최대화하는 범위 탐색 알고리즘을 제안하고 이것을 K-최근접 탐색 알고리즘에 적용하여 탐색 성능향상을 꾀한다. 마지막으로, 다양한 환경에서의 실험을 통해 제안하는 색인구조의 탐색 성능을 테스트하고 기존에 제안된 병렬 다차원 색인구조와의 비교를 통해 제안한 방법의 우수함을 보인다.

키워드 : 병렬 환경, 고차원 색인구조, 범위 질의, K-최근접 질의

Abstract Generally, multi-dimensional data such as image and spatial data require large amount of storage space. There is a limit to store and manage those large amount of data in single workstation. If we manage the data on parallel computing environment which is being actively researched these days, we can get highly improved performance. In this paper, we propose a parallel high-dimensional index structure that exploits the parallelism of the parallel computing environment. The proposed index structure is nP(processor)-n×mD(disk) architecture which is the hybrid type of nP-nD and 1P-nD. Its node structure increases fan-out and reduces the height of a index tree. Also, A range search algorithm that maximizes I/O parallelism is devised, and it is applied to K-nearest neighbor queries. Through various experiments, it is shown that the proposed method outperforms other parallel index structures.

Key words : parallel environment, high-dimensional index structure, range quer, K-nearest neighbor query

1. 서 론

최근 중요한 응용들로 주목받고 있는 지리 정보 시스템, 캐드 데이타베이스, 의료 데이타베이스, 내용기반 이미지 검색 시스템, 멀티미디어 데이타베이스 등에서 고차원 색인구조는 매우 중요한 비중을 차지한다. 고차원 색인구조의 역할은 대용량의 고차원의 특징을 갖는 데이타들을 색인 하여 사용자가 원하는 데이타를 효과적으로 검색하는 것이다. 고차원 색인 구조의 중요성은 이미 여러 해 전부터 인식되었고 이에 대한 연구가 매우 활발히 진행되어 왔다. 현재 제안된 고차원 색인 구조들은 TV-트리[1], X-트리[2], SS-트리[3], SR-트리[4], CIR-트리[5]와 같은 데이타 분할을 사용하는 색인 구조와 KDB-트리[6], hB-트리[7], LSDh-트리[8], BANG 화일[9], GRID 화일[10]과 같이 공간분할을 사용하는 색인 구조들이 있다. 또한, 이들의 혼합 형태의

· 본 연구는 한국과학재단 목적기초연구(두정기초연구 과제번호 R01-1999-0024) 지원으로 수행되었음
† 정 회 원 : 한국전자통신연구원 연구원
parkcs@ctri.re.kr
†† 비 회 원 : 충북대학교 정보통신공학과
princc@netdb.chungbuk.ac.kr
jrshin@netdb.chungbuk.ac.kr
††† 종신회원 : 충북대학교 정보통신공학과 교수
yjs@cbucc.chungbuk.ac.kr
논문접수 : 2001년 1월 15일
심사완료 : 2001년 11월 20일

색인구조로 Hybrid-트리[11]가 존재하며 이 외에도 LS(Locality Sensitive)-해쉬[12] 기법을 사용하는 색인구조와 VA-파일[13]과 IQ-트리[14]처럼 요약 기법을 사용하는 색인 구조도 존재한다.

일반적으로 다차원의 특징을 갖는 이미지, 공간, 동영상 데이타들은 대용량이라는 특징을 갖는다. 대용량의 데이타를 하나의 워크스테이션에 저장하고 검색을 처리하는 데는 성능측면에서 한계가 있다. 이 한계를 극복하기 위해 대용량의 데이타를 분산저장하고 질의와 삽입 및 삭제를 병렬로 처리하여 성능을 향상시키려는 연구가 90년대 중반부터 진행되었다. 이들 연구에서도 병렬로 질의를 처리하는 것이 단일로 처리하는 것에 비해 훨씬 우수한 성능을 발휘한다는 것이 보고 되고 있다. NOW(Network of Workstation)[15]는 대표적인 무공유(Shared Nothing)형태의 병렬환경이다. 이 외에도 최근 제시된 새로운 개념의 저장 구조인 SAN(Storage Area Network)[16]이 있는데 이것은 여러 컴퓨터가 대량의 저장장치들을 공유하는 공유디스크(Shared Disk)형태의 병렬환경이라 할 수 있다.

이 논문에서는 NOW나 SAN과 같은 병렬 환경에서 I/O의 병렬성과 CPU 연산의 병렬성을 효과적으로 이용하는 병렬 고차원 색인 구조를 제안한다. 제안하는 병렬 색인구조는 하나의 컴퓨터가 여러 디스크로부터 I/O의 병렬성을 취하고 다시 각각의 컴퓨터는 상호 협동작업을 통해 질의를 병렬로 처리하는 nP-n×mD 형태의 구조를 갖게 된다. 또한 보다 향상된 I/O의 병렬성을 획득하는 범위질의 알고리즘을 고안하고 이를 K-최근접 질의에 적용하여 기존의 K-최근접 질의와 실험을 통해 비교한다. 마지막으로 제안하는 병렬 고차원 색인구조를 최근 중요한 저장시스템으로 부각되고 있는 SAN 환경에 어떻게 적용시킬 수 있는지에 대해 기술한다.

이 논문의 구성은 다음과 같다. 먼저 2장에서는 기존 관련 연구에 대한 분석을 통해 문제점을 제시하고 병렬 고차원 색인구조의 요구조건을 도출한다. 3장에서는 제안하는 병렬 색인구조를 그림을 통해 설명하고 각 연산에 대한 의사코드를 제시한다. 4장에서는 다양한 환경의 시뮬레이션을 통한 성능 평가 결과를 제시하고 5장에서는 제안하는 방법을 SAN에 어떻게 적용할 수 있는지를 설명한 후 마지막으로 6장에서 결론을 맺는다.

2. 관련 연구

2.1 고차원 색인 구조

이미 언급했듯이 고차원 색인 구조에 대한 연구는 지난 십여년 동안 매우 활발히 진행되었다. 기존에 제안된 고차원 색인 구조들을 분류해보면 TV-트리, X-트리, SS-트리, SR-트리, CIR-트리와 같은 데이타 분할을 사용하는 색인 구조와 KDB-트리, hB-트리, LSDh-트리, BANG 화일, GRID 화일과 같이 공간분할을 사용하는 색인 구조들로 크게 나누어 볼 수 있다. 또한, Hybrid-트리와 같은 이들의 혼합 형태의 색인구조가 존재하며 기타 LS(Locality Sensitive)해쉬 기법을 사용하는 색인구조와 VA-파일과 IQ-트리처럼 요약 기법을 사용하는 색인 구조도 존재한다.

공간 분할을 사용하는 색인구조들은 공간을 서로 겹치지 않도록 분할하여 표현한다. 이들의 특징은 비 단말 노드의 엔트리의 크기가 차원과 독립적인 장점을 가지고 있다. 하지만 차원이 증가할수록 Dead Space가 증가하고 하향 연쇄 분할(Downward cascading split)의 빈도수가 증가하여 저장공간 활용률이 현저히 떨어지는 문제점을 가지고 있다. 또 다른 방법인 데이타 분할을 사용하는 색인 구조들은 MBR(Minimum Bounding Region)형태로 비 단말 노드의 엔트리를 표현하기 때문에 Dead Space가 발생하지 않는 장점을 가지고 있다. 또한 겹침을 허용하기 때문에 공간 분할 방법의 문제점인 하향 연쇄분할 같은 문제는 발생하지 않는다. 하지만 MBR 표현 방식 때문에 차원이 증가할수록 비 단말 노드의 팬-아웃은 떨어지며 겹침 영역이 증가하게 되어 검색 성능이 현격히 떨어지는 문제점을 가지고 있다. 공간 분할 방법과 데이타 분할 방법의 장점을 혼합한 방식이 바로 Hybrid-트리이다. 여기에서는 공간분할방식의 비단말 노드 엔트리의 크기가 차원에 독립적이라는 특성과 MBR로 엔트리를 표현함으로써 겹침을 허용하여 하향 연쇄 분할을 피하고 있다. 하지만 이 구조에서도 완벽하게 비 단말 노드의 엔트리의 크기가 차원과 독립적이라고 말할 수 없다.

2.2 병렬 다차원 색인 구조

기존에 제안된 병렬 다차원 색인 구조는 크게 1P-nD의 구조와 nP-nD의 구조로 나누어 볼 수 있다. 여기서 P와 D는 각각 프로세서와 디스크를 의미한다. 1P-nD 구조는 하나의 처리기에 다중 디스크가 연결되어 있는 병렬환경을 말한다. 이 구조에서는 다중 디스크의 병렬 I/O를 통해서 다차원 색인 구조의 성능을 개선한다. 하지만 프로세서와 다중 디스크들 사이에 단 하나의 채널이 존재하므로 데이타를 주 기억공간으로 적재하는 작업은 직렬로 처리된다. 이 부분에서 병목현상이 발생할 수 있으며 색인 구조의 성능 향상에 한계가 있게 된다. 1P-nD의 대표적인 병렬 색인구조들은 MXR-트리[17], 병렬 X-트리[18], PML-트리[19] 등이 있다.

nP-nD는 다중 처리기와 다중 디스크가 존재하는 병렬 환경이다. NOW와 같은 환경이 그 한 예가 될 것이다. 이런 환경에서 구축된 병렬 색인 구조들은 처리기와 디스크 I/O의 병렬성을 모두 이용할 수 있어서 보다 색인 구조의 성능을 향상시킬 수 있다. 이에 대표적인 병렬 색인 구조들로는 M-R트리[20], MC-R트리[21], DSVM(Distributed Shared Virtual Memory)상에서의 병렬 R-트리[22], GPR-트리[23], 병렬 VA-화일[24] 등이 있다. 다음에서 1p-nD 와 nP-nD 각각의 대표적 색인 구조인 병렬 R-트리와 M R트리에 대한 분석 내용을 기술한다.

[22]에서 제안한 병렬 R-트리는 디스크 I/O의 병렬성을 이용해 지리정보 시스템, 캐드 등의 응용에서 가장 일반적인 질의 형태인 범위 질의의 성능개선을 시도했다. 다중 디스크를 사용하는 이유는 병렬 I/O를 통해 전체적인 I/O 시간을 줄이고 또한 데이타의 양이 많아져 하나의 디스크에 저장하지 못하는 경우 이를 나누어 저장하려는 목적을 갖는다. [22]에서는 구성 가능한 병렬 R-트리의 구성방법으로 독립 R-트리, 수퍼노드, 멀티 플렉스 R-트리 형태로 구분하고 실험을 통해 멀티플렉스 R-트리가 독립 R-트리나 수퍼노드보다 월등히 앞서는 것을 보여준다. 멀티플렉스 R-트리에서 성능향상의 요점은 각 노드들이 저장될 디스크를 어떻게 결정할까 하는 것이다.

[22]에서는 이를 결정하기 위한 기준으로 데이타의 개수 균형, 데이타들의 영역 균형, 저장되는 노드들이 서로 유사할 경우에는 다른 디스크로 저장하는 유사성을 들고 있다. 이들 모두를 만족하는 방법은 어려우며 실험을 통해 여기에서는 유사성이 가장 효과적이라는 결론을 내었다. 이때 유사성을 계산하는 방법은 같은 질의를 수행할 때 접근되는 노드들은 유사성이 높은 노드들이라는 데에서 출발한다. 그래서 유사성 측정 척도는 노드 R, S가 있을 때 이 둘을 모두 접근하는 질의의 비율로 결정한다. 하지만, 이 연구에서 제시된 방법은 처리기에 대한 병렬성은 고려되지 않고 단지 I/O 시에 대해서만 병렬로 처리하기 때문에 성능향상에 있어서 한계가 있다. 또한 이 연구에서는 고차원의 데이타에 대한 고려가 되지 않았으며, 유사도 검색에 있어서 중요한 질의 형태인 K-최근접 질의에 대한 고려가 되지 않았다.

[20]에서 제안한 MR-트리는 그림 1과 같은 구조를 갖는다. MR-트리는 디스크 I/O의 병렬성 뿐 아니라 랜으로 연결된 컴퓨터들이 상호 협동하여 병렬로 질의를 처리하는 1P-nD 구조이다. 이 구조에서는 주 서버와 서버들이 랜으로 연결되어 있으므로 통신에 대한 비용을 무시할 수 없다. 따라서 통신에 대한 비용을 줄이기 위해 메시지 교환을 최소화하는 방향으로 설계되었다. 주 서버(Master Server)가 R 트리의 비단말 노드를 저장하고 있고 R-트리의 단말노드들은 서버에 분산되어 있다. 주 서버에 저장된 R-트리의 단말노드에는 (MBR, SiteID, PageID)형태의 엔트리가 저장되어 있다. (SiteID, PageID)가 의미하는 것은 단말노드가 어느 사이트의 어떤 페이지인지를 나타낸다.

이미 언급한대로 [22]에서는 병렬 R-트리를 구성하는 방법으로 멀티 플렉스 R-트리, 수퍼 노드, 독립 R-트리를 제안했다. 이중 멀티 플렉스 R-트리 형태로 구성하게 되면 서버들을 가로질러 포인터로 연결되어 있기 때문에 메시지 교환이 빈번히 발생하게 된다. 따라서 통신에 대한 비용이 많이 발생하게 되므로 효율적이지 않다. 두 번째 방법인 수퍼노드 형태의 구성은 작은 질의에 대해서도 거의 모든 디스크에 접근하기 때문에 전체 시스템의 성능을 저하시키는 문제점이 생긴다. 따라서 독립적인 R-트리 형태로 구성하는 것이 통신비용을 줄일 수 있으므로 가장 효율적이다.

MR-트리의 질의 처리과정은 다음과 같다. 사용자로부터의 질의는 주 서버로 보내진다. 주 서버는 질의를 받아서 자신이 저장하고 있는 R-트리에서 탐색을 수행한다. 질의와 부합하는 모든 (PageID, SiteID)쌍을 리스트에 저장하고 탐색이 끝나면 질의와 페이지 리스트를 해당 서버(SiteID)로 전송한다. 질의와 페이지 리스트를 전달받은 서버들은 탐색을 수행하고 탐색 결과를 주 서버로 반환한다. 주 서버는 서버로부터의 탐색결과를 종합하여 사용자에게 전송한다. 이 방법은 주 서버와 서버 사이에 비교적 메시지 교환이 많다는 문제점이 있다. 그리고 유사도 검색에 있어서 범위 질의와 함께 자주 발생하는 K-최근접 질의에 대한 고려가 되지 않았다.

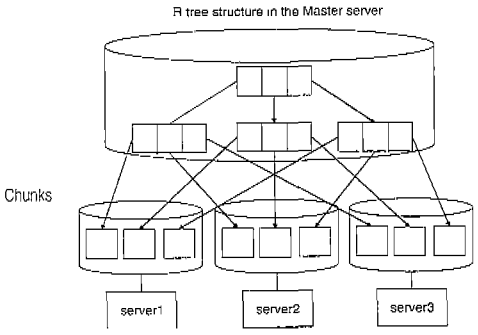

그림 1 M-R tree 구조

3. 제안하는 병렬 고차원 색인구조

3.1 제안하는 병렬 고차원 색인구조의 구성

제안하는 색인구조는 그림 2와 같은 구조를 갖는다. 디스크들을 서버의 개수만큼 그룹으로 형성하고 각 서버가 디스크 그룹을 관리한다. 서버는 주 서버와 부 서버로 구분을 할 수 있는데, 주 서버는 부 서버의 역할과 탐색 결과를 통합 관리하는 기능을 하고, 부 서버는 색인 구성에 관한 연산을 수행하게 된다. 이런 구조는 nP-n×mD 형태라 할 수 있다. 기존에 제안된 병렬 색인구조를 보면 nP-nD 또는 1P-nD의 구조를 갖는다. 이 논문에서는 nP-nD와 1P-nD의 구조를 혼합한 nP-n×mD 형태를 제안한다. 전체적으로 CPU의 병렬성을 이용하고 각각의 CPU는 다시 I/O의 병렬성을 이용하는 형태이다.

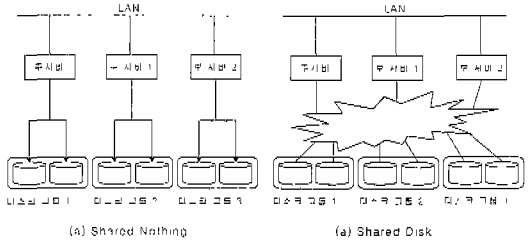

그림 2 제안한 시스템 구성도

하나의 서버는 하나의 디스크 그룹을 관리하며 각 디스크 그룹에는 별도의 색인구조가 존재한다. 그림 3은 각 서버가 관리하는 디스크 그룹내의 트리 구조를 보여준다. 그림에서 노드는 논리적인 의미로써 색인구조의 단말노드나 비단말 노드를 의미한다. 페이지란 물리적인 디스크의 입출력 단위이다. 제안하는 방법에서는 이 노드와 페이지가 다른 의미로 사용되고 있다. 디스크 그룹내에서 한 노드는 그룹내 디스크 페이지들에 나뉘어 저장된다. 노드의 한 엔트리에는 그것이 가리키는 하위 노드의 MBR과 그 노드를 구성하는 페이지들(각 디스크에 분산된)에 대한 포인터가 포함되어 있다. 그림 3을 다시 보면 루트 노드의 첫 번째 엔트리는 node 1을 가리키고 있다. node 1은 디스크 A, B, C의 첫 번째 페이지들이 모여서 이루어지게 되며 루트 노드의 첫 번째 엔트리에는 그 페이지들에 대한 포인터와 페이지들에 저장된 객체를 포함하는 MBR이 저장된다.

이 구조의 장점은 다음과 같다. 먼저 유사한 데이타는 서로 다른 디스크에 저장되는 디클러스터링 효과를 얻을 수 있다. 고차원 색인구조에서 한 노드에 저장되는 데이타들은 가장 유사한 것들이다. 제안하는 구조에서는 하나의 노드를 여러 디스크의 페이지에 나누어 저장하기 때문에 자연스럽게 디클러스터링 효과를 얻는다. 두 번째, 트리의 하나의 노드 크기가 (그룹내의 디스크 개수 × 페이지 크기) 가 되므로 트리의 높이가 낮아진다. 노드의 크기가 크지만 이것은 한번의 I/O 시간에 병렬로 읽어 올 수 있으므로 문제가 되지 않는다. 데이타 분할 기법의 색인구조에서 고차원 데이타로 갈수록 노드의 팬-아웃이 작아지게 되어 트리의 높이가 높아지게 되고 이로 인해 겹침 영역도 증가하는 연쇄효과가 발생한다. 물론 이것은 질의 성능 저하로 이어진다. 하지만 제안하는 구조에서는 디스크 수에 따라서 팬-아웃 수를 조정할 수 있으므로 트리의 높이를 낮게 유지할 수 있다.

마지막으로, 고차원 색인 구조에서는 차원이 증가함에 따라 탐색 수행시 접근해야하는 노드의 수가 증가하게 된다. 고차원일수록 질의에 참여하는 페이지의 수가 많을 것이다. 따라서, 병렬 구조에 있어서 중요한 성질들 중 최소 활성 디스크(Minimum Load) 성질보다는 균등 활성 디스크 (uniSpread) 성질을 최대한 지원하는 것이 더 효과적이다. 제안된 구조는 하나의 노드를 읽기 위해서는 반드시 노드를 구성하는 페이지들이 존재하는 디스크들을 읽어야 하기 때문에 uniSpread성질을 최대화 할 수 있게 되어 범위 질의 및 K-최근접 질의 처리시 효과적이다. 이상으로 전체적인 구조에 대한 설명을 마치고 제안하는 색인구조에서는 어떻게 삽입과 탐색을 수행하는지에 대해서 자세히 설명한다.

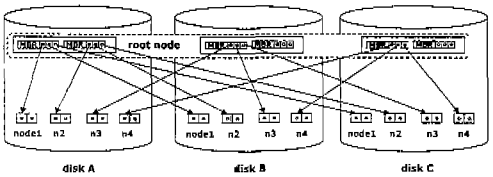

그림 3 각 디스크 그룹에서의 트리 구조

3.2 삽 입

그림 4는 제안하는 색인구조에 엔트리가 삽입되는 과정을 보여준다. 그림에서와 같이 엔트리 a, b, c, d, e, f, g, h 가 차례로 삽입될 때 a는 첫 번째 디스크 그룹에 b는 두 번째 디스크 그룹, c는 세 번째 그룹에 차례로 할당된다. 삽입할 엔트리를 디스크 그룹에 할당할 때는 라운드로빈 방법을 이용한다. 고차원으로 갈수록 다른 디클러스터링 방법이 크게 효과를 보이지 못하고 라운드로빈 방법과 비교했을 때 크게 나은 성능을 보이지 못한다[14]. 또한 라운드 로빈 방법은 계산에 대한 비용이 적게 들고 구현하기 쉬워 이 논문에서는 라운드 로

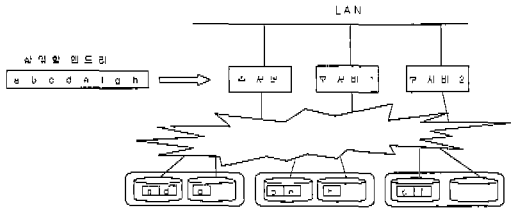

그림 4 전체 SAN 환경에서 삽입 과정

빈 방법을 사용한다.

각각의 엔트리는 각 디스크 그룹에 할당되고, 할당된 엔트리는 각 디스크 그룹내의 색인 트리에 삽입된다. 새로운 엔트리를 삽입할 단말노드를 찾기 위해 트리를 순회한다. 적당한 노드를 찾는 알고리즘은 어떤 방법을 사용해도 관계없다. 트리 순회를 통해 적당한 노드를 찾으면 엔트리를 삽입할 노드에 여유 공간이 있는지를 체크한다. 삽입할 노드에 여유 공간이 있으면 삽입하고 부모 노드에 MBR 변경을 반영하고 종료하면 된다. 노드에 여유 공간이 있지만 페이지가 여유 공간이 없을 때는 현재 노드에 페이지가 할당되지 않은 디스크로부터 페이지를 할당받아서 삽입한다.

노드를 구성하는 페이지들이 차서 노드 넘침이 발생하면 분할을 수행한다. 분할을 수행할 때 분할된 두 노드를 구성하는 페이지가 어느 디스크에 위치하게 할 것인지를 탐색성능 관점에서 고려해 볼 필요가 있다. 일반적으로 다차원 색인 구조의 노드는 항상 100% 차 있는 것이 아니다. 이런 이유로, 하나의 노드를 접근할 때 그룹내의 모든 디스크로부터 I/O의 병렬성을 얻을 수 없다. 이런 문제를 해결하기 위해서 분할된 두 노드를 구성하는 페이지들이 되도록 모두 서로 다른 디스크에 저장될 수 있도록 하는 정책을 취한다. 이것은 범위질의가 수행될 때 I/O의 병렬성을 높이는 역할을 하는데 이에 대한 자세한 설명은 3.3절에서 한다.

제안하는 색인구조의 분할 전략은 분할로 생성되는 두 노드를 구성하는 페이지는 되도록 서로 다른 디스크에 저장하는 것이다. 이를 위해서 그룹내의 각 디스크들 중 현재 분할이 발생한 트리 레벨의 노드들을 구성하는 페이지들이 가장 적게 분포하는 디스크에 우선적으로 페이지가 할당되게 한다. 이렇게 함으로써 특정 디스크에 페이지가 집중되는 것을 방지할 수 있다. 그림 5와 그림 6에서 예를 통해 노드 분할을 설명한다. 그림 5 에서 노드 2에 새로운 엔트리를 삽입할 때 넘침이 발생한다. 노드 2의 넘침은 그림 7에서처럼 분할을 수행하여 처리한다. 분할을 위해서 3번 노드를 새로 할당하고 2번 노드의 엔트리 일부를 3번으로 이동한다. 3번 노드를 할당할 때 노드를 구성하는 페이지를 현재 트리 레벨에서 할당된 페이지가 가장 적은 디스크부터 할당을 시작한다. 그림 6을 보면 D, E가 가장 적은 페이지 수를 유지하기 때문에 디스크 D와 E에서부터 할당하게 된다. 노드 2를 디스크 D부터 할당하고 다음 디스크에 노드 3을 차례로 할당하게 된다. 이렇게 분할된 노드를 다른 디스크에 할당함으로써 범위 질의 처리시에 여러 노드들을 보다 작은 I/O를 통해 읽을 수 있게 된다. 그림 7에서 삽입 연산의 의사코드를 보여준다.

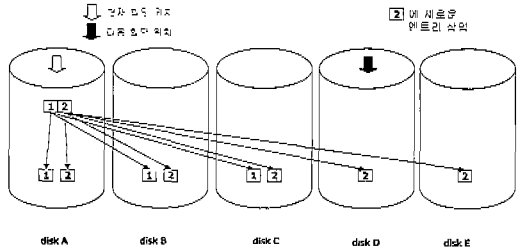

그림 5 노드 분할 예 (1)

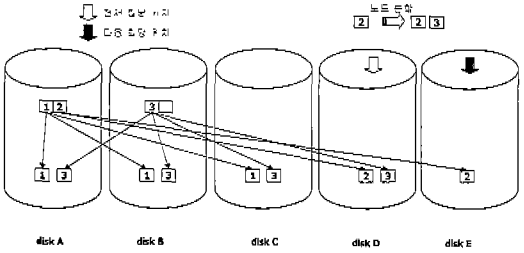

그림 6 노드 분할 예 (2)

```
Function Insert()
주 서버: 입력(new_entry)
        server = 라운드 로빈으로 new_entry를 저장할 (서버)그룹 결정;
        new_entry를 서버에 전달;
모든 서버: 입력(new_entry)
        leaf_node := 트리 순회를 통해 new_entry를 삽입할 단말노드 선택;
        if(leaf_node의 노드에 여유공간이 없으면)
                새로운 노드 할당;
                현재 분할이 발생한 레벨의 페이지들이 가장 적게 분포하는 디스크에 먼저 페이지가 할당;
        else if(leaf_node의 페이지에 여유공간이 없으면)
                현재 노드에 페이지가 한당되지 않은 디스크로부터 페이지를 할당받아서 삽입;
        else
                새로운 엔트리 삽입;
                변경된 MBR 부모 노드에 반영;
        end if
End Function
```

그림 7 삽입 알고리즘

3.3 검 색

3.3.1 범위 질의

다차원 색인구조의 범위 질의 알고리즘은 이미 여러 연구에서 언급해왔다[1, 10]. 다차원 색인구조에서 범위 질의를 수행할 때는 다중경로를 순회한다. 즉, 어떤 노드에서 다음 탐색할 하위노드를 결정할 때 여러 하위노드가 선택될 수 있다. 기존의 방법은 그림 8에서처럼 하위 노드를 순차적으로 읽어 가면서 질의를 처리했다. 질의 수행을 시작하면 먼저 루트 노드에 접근해서 1에서 8까지의 엔트리들중 질의에 부합하는 엔트리 2, 4, 7을 선택한다. 다시 엔트리 2, 4, 7이 가리키는 자식노드를 접근하기 위해 하나씩 읽는다. 노드 2는 디스크 A, D, E에 분산되어 저장되기 때문에 노드 2를 접근할 때는 디스크 A, D, E를 동시에 접근하게 된다. 다음으로 4, 7 노드 순으로 접근해서 데이타를 읽어 온다. 탐색을 위한 총 디스크 접근 수는 루트 노드에 접근할 때의 I/O와 단말노드를 접근할 때의 I/O 회수의 합이 된다. 루트 노드 접근 수는 1이고 단말노드에 대한 디스크 접근수가 3이므로 총 디스크 접근 수는 4가 된다.

제안하는 방법에서는 이렇게 하지 않고 각 노드를 구성하는 페이지들의 I/O 순서를 조정하여 디스크 접근 수를 줄인다. 그림 9에서 그 예를 보여준다. 앞의 예에서 2, 4, 7이 선택되었을 때 각 엔트리가 가리키는 노드들을 구성하는 페이지에 대한 정보를 수집하여 디스크 I/O 계획을 세운다. 여기에서 엔트리 2의 포인터 A3, D1, E1이 의미하는 것은 각각 A 디스크의 세 번째 페이지, D 디스크의 첫 번째 페이지, E 디스크의 첫 번째 페이지를 의미한다. 디스크 I/O 계획은 되도록 I/O 회수를 줄일 수 있는 방향으로 세운다. 그림 9에서 A3, B3, C3, D4, E1은 모두 서로 다른 디스크에 존재하는 페이지이고 한번의 I/O로 읽어 낼 수 있다. 다음으로 A5, D4, E2 역시 한번의 I/O를 통해 읽기가 가능한 페이지들이다. 이처럼 I/O를 수행할 때 전체 I/O 회수는 2회의 단말노드 접근 회수와 1회의 루트 노드 접근 회수를 합한 3이 된다. 첫 번째 방법보다 1회가 줄어들게 된다. 그림 10에서 제안하는 범위검색 알고리즘의 의사코드를 보여 준다.

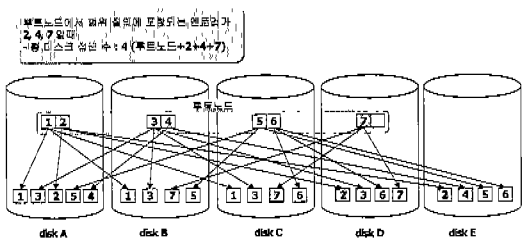

그림 8 범위 질의 탐색 (노드별 접근)

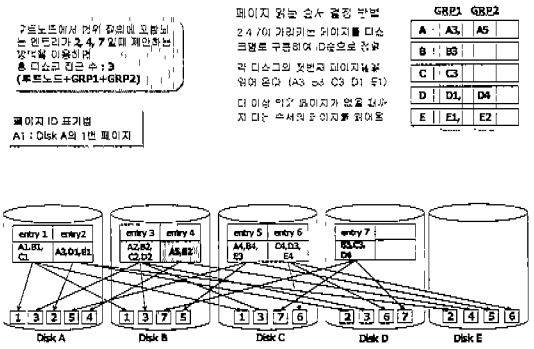

그림 9 범위 질의 탐색(여러 노드의 페이지를 동시에 접근)

```
RangeSearch()
모든 서버 : 입력 ( 질의 특징벡터, 범위 )
  root 를 I/O 스케줄러에 전달;
  while ( 1 )
    if (I/O 스케줄러에서 관리하는 정보가 없으면)
      break;
    end if
    cur_node : = I/O 스케줄러로부터 여러 노드를 병렬로 접
                 근;
    if ( cur_node == leaf )
      노드에 포함된 객체와 질의의 유사도 계산하여 result_
      set에 저장;
    else
      범위에 포함되는 엔트리를 선택;
      선택된 엔트리에 대한 정보를 I/O 스케줄러에 전달;
    end if
  end while
  결과를 주 서버에 전달;

주 서버: 반환 ( 결과 엔트리 )
  각 서버에서 처리된 결과를 취합해서 유사도 순으로 정렬;
  정렬된 결과를 클라이언트에 반환;
```

그림 10 범위 질의 알고리즘

3.3.2 K-최근접 질의

기존의 병렬 다차원 색인 구조에서는 K-최근접 질의 방법에 대한 연구가 거의 되어있지 않았다. 고차원 색인구조에 있어서 K-최근접 질의는 매우 중요하다. 이 논문에서는 세 가지 방법을 제시하고 실험을 통해서 가장 적절한 것을 선택한다.

3.3.2.1 각 서버에서 독립적으로 K-최근접 질의 수행

첫 번째 방법은 각 서버에서 독립적으로 K-최근접 질의를 수행하는 것이다. 질의가 들어오면 주 서버와 각

각의 부 서버에서 K-최근접 질의를 독립적으로 수행한 후 각 서버의 결과를 주 서버가 통합하여 클라이언트에게 반환한다. 이 방법은 구조가 간단하고 구현이 쉬운 장점이 있다. 하지만, K-최근접 질의의 특성상 범위질의에 적용한 I/O 병렬성의 장점을 얻을 수 없다. 그림 11은 제안하는 첫 번째 K-최근접 질의 알고리즘의 의사코드를 보여준다.

```
KNNSearch() - 1
모든 서버: 입력 ( 질의 특징벡터, k )
  k-distance = ∞;
  PQ(우선순위 큐) 초기화;
  result_set 초기화;
  while (1)
    if (node == 비단말 노드)
      질의와 엔트리와의 유사도 계산;
      엔트리들을 질의와의 유사도를 키로 하여 PQ에 삽입;
    else
      질의와 엔트리와의 유사도를 계산;
      노드에서 k-distance 보다 작거나 같은 엔트리를
      result_set에 저장;
    end if
    if (result_set에 저장된 값이 k 개 보다 크거나 같을 때)
      k-distance = result_set의 k번째 엔트리와의 거리;
      if (k 번째 결과 <= PQ의 첫 번째 엔트리의 유사도 값)
        break;
      end if
    end if
    node := PQ 에서 엔트리를 꺼내고 엔트리가 가리키는 자
            식 노드를 병렬로 읽음;
  end while
  각 서버에서의 결과 k 개를 주 서버에 전달;

주 서버 : 반환 ( k 개의 결과 엔트리 )
  각 결과를 취합하여 유사도 순으로 정렬;
  K 개의 결과를 선택하여 클라이언트에 반환;
```

그림 11 K-최근접 탐색 알고리즘(1)

3.3.2.2 변형된 K-최근접 질의 (1)

두 번째 방법은 K-최근접 질의에 3.3.2.1에서 제안한 범위질의를 적절히 혼용한 방법이다. K-최근접 질의가 들어오면 우선 그림 12와 같이 주 서버에서 부분적인 K-최근접 질의 검색을 수행한다. 부분적인 K-최근접 질의에서는 일단 첫 번째 K개의 결과를 얻으면 이를 통해 얻은 K번째 객체와 질의 사이의 거리를 구해서 범위질의 형태로 변환한다. 즉, 완전한 K-최근접 검색을 하는 것이 아니라, 부분적인 K-최근접 검색을 하고 K 번째 결과를 범위질의 형태로 전환하고, 범위 질의를 다른 부 서버에 전달한다. 부 서버에는 병렬성을 최대화해서 검색 결과를 주 서버로 전달하고, 주 서버에는 각 서버의 결과를 통합하여 유사도 순으로 정렬해서 최종 K 개의 결과를 얻는다. 이 방법은 범위 질의 형태로 변환해서 병렬성을 최대화하는 장점이 있다. 그러나 주 서버에서 얻은 범위질의의 크기가 크면 필요 없는 연산을 많이 할 수 있다는 단점을 가지게 된다. 그림 13은 지금까지 설명한 K-최근접 질의 알고리즘의 의사코드를 보여준다.

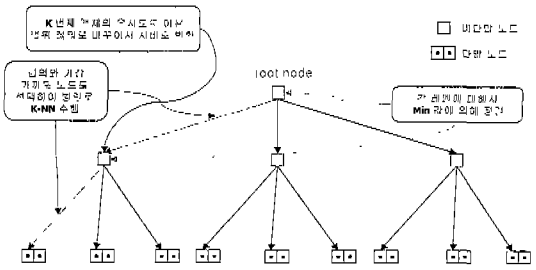

그림 12 주 서버에서 부분적인 K-최근접 탐색 알고리즘

```
KNNSearch() - 2
주 서버 : 입력 ( 질의 특징 벡터, k ), 반환 ( k 개의 결과
엔트리 )
  k-distance = ∞;
  PQ(우선순위 큐) 초기화;
  result_set 초기화;
  while (1)
    if (node == 비단말 노드)
      질의와 엔트리와의 유사도 계산;
      엔트리들을 질의와의 유사도를 키로 하여 PQ에 삽입;
    else
      질의와 엔트리와의 유사도를 계산;
      노드에서 k-distance 보다 작거나 같은 엔트리를
      result_set에 저장;
    end if
    if (result_set에 저장된 값이 k 개 보다 크거나 같을 때)
      break;
    end if
    node := PQ 에서 엔트리를 꺼내고 엔트리가 가리키는 자
            식 노드를 병렬로 읽음;
  end while
  K 번째 결과 객체와 질의와의 유사도를 계산하여 범위 질의
  형태로 변환;
  계산한 범위 질의를 각 서버에 전달;

모든 서버 : 입력 ( 질의 특징벡터, 범위 )
  제안한 방법의 범위 질의 수행;
  범위 질의 결과를 주 서버에 전달;
  결과를 취합하여 정렬하여 K개를 선택;
  K개의 결과를 클라이언트에게 반환
```

그림 13 K-최근접 탐색 알고리즘(2)

3.3.2.3 변형된 K-최근접 질의 (2)

두 번째 방법은 주 서버에서 부분적인 K-최근접 질의 검색을 하기 때문에 범위 질의 형태로 변환했을 때 범위가 너무 크면 탐색 효율이 떨어진다는 문제점이 있다. 이러한 단점을 개선하기 위해 부분적인 K-최근접 질의를 주 서버에서만 수행하는 것이 아니라, 모든 서버

가 동시에 부분적인 K-최근접 질의를 수행하게 한다. 이 결과들은 모두 주 서버로 전송되어 그 중에 가장 범위가 작은 것을 선택하여 이를 가지고 각 서버가 범위 질의를 수행한다. 그림 14는 지금까지 설명한 K-최근접 질의 알고리즘의 의사코드를 보여 준다.

```
KNNSearch() - 3

모든 서버 : 입력 ( 질의 특징 벡터, k )
  k-distance = ∞;
  PQ(우선순위 큐) 초기화;
  result_set 초기화;
  while (1)
    if (node == 비단말 노드)
      질의와 엔트리와의 유사도 계산;
      엔트리들을 질의와의 유사도를 키로 하여 PQ에 삽입;
    else
      질의와 엔트리와의 유사도를 계산;
      노드에서 k-distance 보다 작거나 같은 엔트리를
      result_set에 저장;
    end if
    if (result_set에 저장된 값이 k 개 보다 크거나 같을 때)
      break;
    end if
    node := PQ 에서 엔트리를 꺼내고 엔트리가 가리키는
            자식 노드를 병렬로 읽음;
  end while
  k 번째 결과 객체와 질의와의 유사도를 계산하여 범위 질의
  형태로 변환;
  계산한 범위 질의를 각 서버에 전달;

주 서버
각 서버에서 전달된 범위 질의중 가장 범위가 작은 것을 선택;
선택된 범위 질의를 각 서버에 전달;
제안한 방법의 범위 질의 수행;
범위질의 결과 주 서버에 전달;

주 서버 : 반환 ( k 개의 결과 엔트리)
결과를 취합하여 정렬하여 K개를 선택;
k 개의 결과를 클라이언트에게 반환
```

그림 14 K-최근접 탐색 알고리즘(3)

4. 시뮬레이션

본 장에서는 제안한 병렬 고차원 색인구조의 성능을 평가하기 위한 시뮬레이션 환경과 방법에 대해서 설명하고 시뮬레이션 결과를 통해 분석을 수행한다.

4.1 시뮬레이션 환경

실험에 사용된 플랫폼은 Sun Enterprise 250의 CPU에 1GBytes의 주기억 공간을 가지고 있고 Solaris 2.7의 OS를 탑재하고 있으며 gcc 2.8을 컴파일러로 사용하였다. 성능평가 파라미터로는 페이지 크기, 서버의 개수, 디스크 수, 차원 수를 사용했고 성능평가 척도로 검색수행시 디스크 접근수, 전체 노드수에 대한 디스크 접근수 비율, 검색 응답시간을 측정했다. 표 1에서 성능평가 파라미터와 값을 보여준다. 페이지 크기는 2-48Kbytes로 변화시켰고 실험에 사용된 데이타는 10 - 80차원의 균등분포와 9차원의 동영상으로부터 추출한 실데이타를 각각 100,000개씩 사용하였다. 서버의 개수는 3개로 고정하였고 전체 디스크 수는 3 - 18개까지 변화시켜 가면서 실험을 수행하였다.

표 1 성능평가 파라미터와 값

성능평가 파라미터	값
페이지 크기	2Kbytes - 48Kbytes
서버 개수	3
디스크 개수	3 - 18
차원 수	10 80 균등 분포, 9 실 데이타

성능 평가 척도인 디스크 접근수(N_{disk})는 검색을 수행할 때 접근한 총 디스크 접근수를 의미하는데 n개의 디스크로부터 데이타를 읽어올 때 이를 병렬로 수행했다면 1로 한다. 노드 접근 비율이란 (총 디스크 접근수/색인구조의 총 노드 수)를 의미한다. 응답시간(RT)은 디스크 I/O 시간, CPU 시간(T_{cpu}), 결과 전송을 위한 통신 시간의 합으로 계산하였다. [25]의 데이타를 참조하여 디스크 I/O 시간(T_{diskIO})은 1/20,000초로하고 통신속도(T_{comm})는 1.544Mbit/s로 하여 CPU 시간과 합하여 응답시간을 계산한다.

4.2 시뮬레이션 방법

그림 15에서처럼 각각의 디스크는 파일에 대응한다. 일정 크기의 파일을 생성한 다음 이를 정해진 페이지크기 만큼 분할해 놓는다. 이 각 파일의 페이지들이 모여서 노드를 이룬다. 이미 설명한 대로 검색에 대한 응답시간은 CPU 시간, 디스크 접근 속도, 질의 전송 시간을 고려한다. 여기서 CPU 시간은 질의를 처리하기 위해서 계산하는 데 걸리는 비용이고 디스크 접근 속도는 디스크 접근 수와 입출력 I/O 시간의 곱으로 계산한다. 질의 전송시간은 서버와 서버사이의 통신시간과 서버와 클라이언트 사이의 통신시간이다.

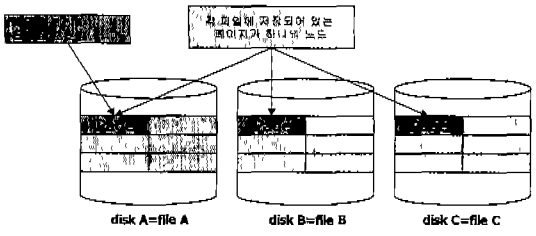

그림 15 디스크 구성

그림 16은 응답 시간을 계산하는 예이다. 병렬 처리의 효과를 얻기 위해서 각 서버의 동작을 순차적으로 실행하고 각 서버의 응답 시간을 계산해서 가장 큰 응답시간을 병렬로 서버를 실행했을 때 최종 응답시간으로 한다. 그림 16에서와 같이 각 서버를 수행해서 응답시간을 얻으면 주 서버의 응답시간 RT1은 4이고 부 서버 1의 응답 시간 RT2는 5, 부 서버 2는 3이 된다고 가정했을 때 병렬환경에서 최종 응답시간은 5가 된다.

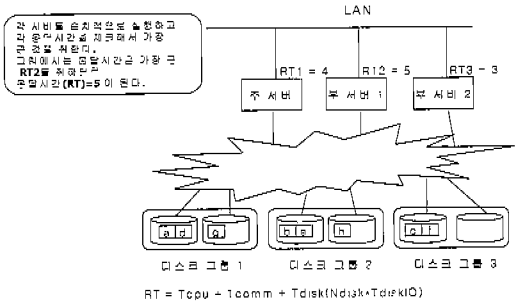

그림 16 응답 시간 계산

4.3 시뮬레이션 결과

이 실험에서는 매우 다양한 환경에서 실험을 수행했다. 실험결과를 차원을 변화시켰을 때, 디스크의 수를 변화시켰을 때, 페이지 크기를 변화 시켰을 때로 구분하여 정리했다. 그리고 제안하는 방법이 기존의 병렬 다차원 색인구조보다 우수하다는 것을 보여주기 위해서 MR-트리와 비교하였다. 다른 병렬 색인구조가 많이 있지만 MR-트리가 비교적 최근에 제안되었으며 높은 성능을 얻을 수 있는 nP-nD형태인 점을 고려하여 선택하였다.

4.3.1 차원 증가에 따른 검색성능

페이지 크기는 4K, 데이타는 10만개의 균등 데이타, 서버의 수는 3, 총 디스크 수는 15로 고정한 상태에서 차원을 10에서 80차원까지 증가시켜 가며 K-최근접 질의와 범위 질의의 디스크 접근수 및 응답시간을 측정하였다. 그림 17에서 그림 19까지는 차원에 따른 K-최근접 질의와 범위 질의의 디스크 접근수, 노드 접근 비율, 응답시간을 보여준다. 그림 17을 보면 차원의 수가 증가함에 따라서 디스크 접근수가 거의 선형적으로 증가한다. 하지만 노드접근 비율은 그림 18에서처럼 20차원을 기준으로 더 이상 증가하지 않고 일정한 것을 볼 수 있다.

각 질의 형태에 따른 디스크 접근수나 응답시간을 보면 KNN 1이 가장 성능이 나쁘고 KNN 2, 3은 거의 비슷하다. 이 이유를 살펴보면 KNN 1은 기존의 KNN 방법을 그대로 이용하는데 이 경우 이 논문에서 제안한 디스크 접근 방법을 이용할 수 없다. 반면 KNN 2와 KNN 3은 범위 질의를 이용해 질의를 수행하므로 디스크 접근수를 효과적으로 줄일 수 있다. 그리고, KNN1은 질의를 수행하기 위해 매번 노드를 접근할 때 마다 엔트리들을 정렬해야 하며 전체적으로 범위질의에 비해 보다 복잡한 계산이 필요하다. 이 계산의 복잡성은 차원이 증가할수록 또 접근하는 노드의 수가 많을수록 더해지게 되어 CPU를 오랫동안 이용해야 한다. 이외에도 디스크 접근수가 더 많다는 것이 종합적으로 작용해서 KNN1 방법이 다른 KNN 방법에 비해서 응답시간 측면에서 훨씬 더 나빠지게 된다.

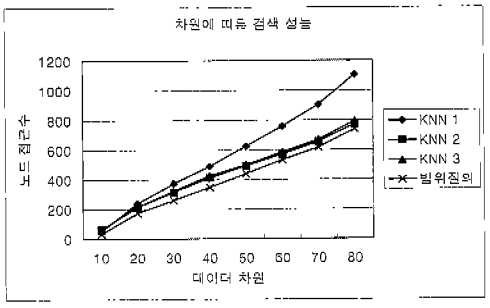

그림 17 차원 vs. 노드 접근 수

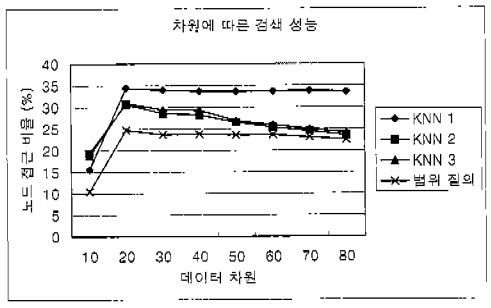

그림 18 차원 vs. 노드 접근 비율

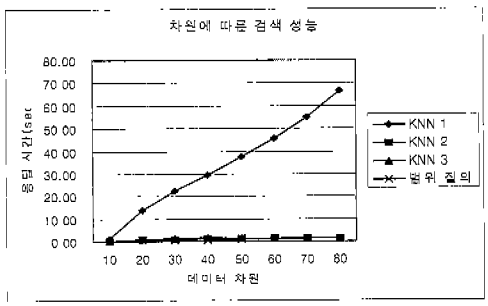

그림 19 차원 vs. 응답 시간

4.3.2 디스크 수에 따른 검색성능

페이지 크기는 4K, 데이타는 10만개의 균등분포 데이타, 차원은 20, 서버의 수는 3으로 고정하고 디스크 수를 3-18개까지 증가시키며 K-최근접 질의와 범위 질의의 처리 성능을 평가하였다. 그림 20에서 그림 22까지를 보면 전체적으로 디스크의 수가 많을수록 디스크 접근수나 응답시간측면에서 성능이 향상되는 것을 볼 수 있다. 제안하는 방법에서 디스크의 수는 노드의 크기를 결정하고 디스크 I/O의 병렬성을 향상시키므로 당연한 결과라 할 수 있다. 특이한 점이라면 그림 22의 응답시간에 대한 비교에서 역시 KNN1의 응답시간이 가장 높게 나왔다. 그리고 KNN 2, KNN 3, 범위질의는 디스크수의 증가에 따라서 응답시간이 약간 줄어들거나 거의 일정한 것을 볼 수 있는데 디스크 I/O 측면에서 병렬성을 얻은 반면 노드의 크기도 커져서 CPU 시간이 길어지기 때문이다. KNN 1에 대해서도 디스크 수의 증가에 따라 응답시간이 감소하다가 9-12개 지점에서 감소 속도가 점점 완만해지는데 이 이유도 역시 CPU 시간에서 찾아볼 수 있을 것이다.

4.3.3 페이지 크기에 따른 검색성능

데이타는 10만개의 균등분포 데이타, 차원수는 20, 서버의 수는 3, 디스크 수는 15로 하고 페이지 크기를 2-48Kbytes로 변화시키며 각 질의 방법의 처리 성능을 측정하였다. 그림 23에서 그림 26까지를 보면 페이지 크기가 증가할수록 디스크 접근수나 응답시간측면의 성능은 향상됨을 볼 수 있다. 특이한 점은 그림 25와 그림 26의 응답시간 그래프이다. 둘 다 응답시간을 보이는 그림인데 그림 26은 KNN 1을 제외하고 다른 검색 방법의 성능 변화를 좀더 자세히 보기 위해서 나타내었다. 그림 26을 보면 페이지 크기가 증가하면서 응답시간이 감소하는 추세를 보이다가 32K 지점에서부터 응답시간이 다시

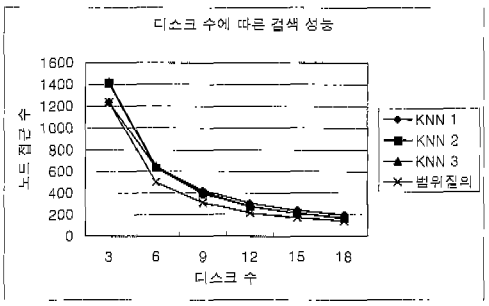

그림 20 디스크 수 vs. 노드 접근 수

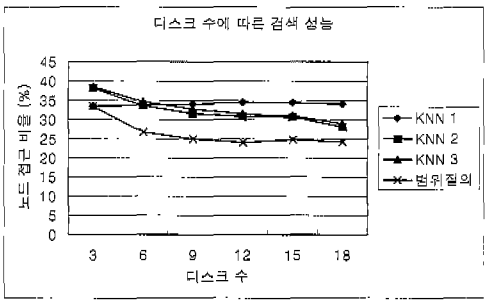

그림 21 디스크 vs. 노드 접근 비율

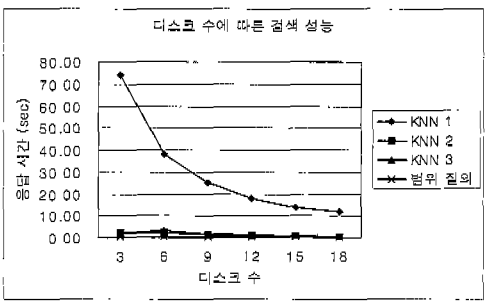

그림 22 디스크 수 vs. 응답 시간

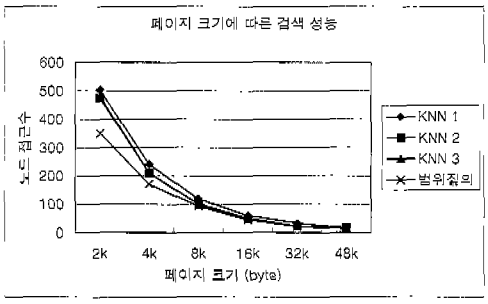

그림 23 페이지 크기 vs. 노드 접근 수

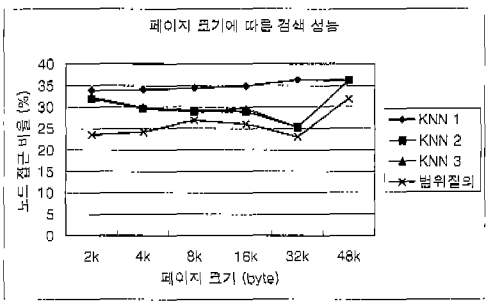

그림 24 페이지 크기 vs. 노드 접근 비율

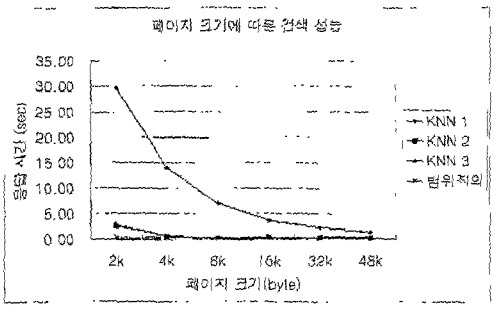

그림 25 페이지 크기 vs. 응답 시간

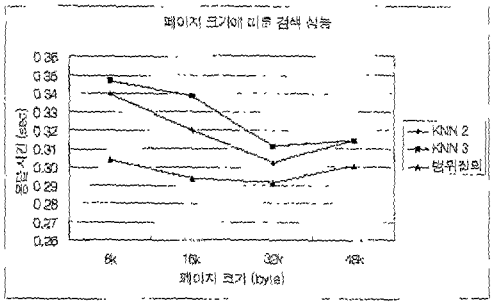

그림 26 페이지 크기 vs. 응답 시간

증가하는 것을 볼 수 있다. 원인은 페이지가 너무 커지게 되면 노드의 크기가 너무 커져서 색인구조로서의 성능을 제대로 발휘 할 수 없게 되고 디스크 접근수는 줄어들지만 계산량이 증가하여 CPU 시간이 길어지기 때문이다. 실험에서는 디스크 개수가 15일 때 약 16-32 Kbytes의 페이지 크기가 최적이었다.

4.3.4 MR-트리와의 검색 성능 비교

기존에 제안된 병렬 다차원 색인구조인 MR-트리와 성능비교를 하였다. 페이지 크기는 4K로 하고 데이타는 10만개의 균등분포와 9차원의 실데이타, 서버 수는 3, 디스크 수는 3, 6, 15로 하고 차원은 균등분포에 대해서는 10-80차원까지 측정하고 실데이타는 9차원으로 고정하고 성능평가를 수행하였다. 그림 27에서 그림 29까지는 균등분포 데이타에 대해서 응답시간과 디스크 접근수를 측정한 것이다. MR-트리는 각 서버가 여러 디스크에 트리를 구성하는 형태가 아니다. 즉, 하나의 서버에는 하나의 디스크가 할당되어 전체적인 색인구조를 구성한다. 하지만 제안하는 방법은 하나의 서버가 임의 수의 디스크를 가질 수 있어서 서버 수보다 디스크의 수가 많을 때 훨씬 유리하다. 그래서 이 실험에서는 서버의 수를 3으로 고정하는 대신 디스크의 수를 3, 6, 15로 변화시키면서 성능을 측정하여 MR-트리와 비교하였다.

그림 27에서 그림 29까지에 나타난 실험 결과를 보면 전체적으로 제안하는 방법이 응답시간이나 디스크 접근수 측면에서 우수하다. 디스크 수가 3인 경우에는 MR-트리와 디스크 접근수나 응답시간이 크게 차이가 나지 않는 것을 볼 수 있다. 하지만 디스크의 수가 6, 15일 때는 평균적으로 2-3배정도 성능이 향상되는 것을 볼 수 있다. 이런 관점에서 볼 때 제안하는 방법은 한 서버가 여러 디스크를 접근할 수 있는 병렬 환경에서 훨씬 좋은 성능을 발휘한다는 것을 알 수 있다. 또한, 디스크의 수가 15일 때 MR 트리와 제안하는 방법의 디스크 접근수의 차이가 응답시간의 차이보다 훨씬 더 큰 것을 볼 수 있다. 이것은 제안하는 범위 질의가 여러 노드를 보다 작은 수의 디스크 I/O를 통해 읽어내는 방법을 이용해 디스크 접근수를 줄이고 있지만 계산해야 하는 양은 MR-트리와 비교해서 크게 다르지 않기 때문이다. 즉, 디스크 I/O 횟수는 줄지만 CPU 시간은 줄지 않으므로 응답시간의 차이가 디스크 접근수 차이에 비해서 더 작게 되는 것이다.

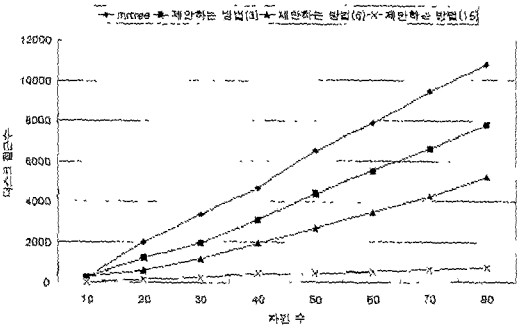

그림 27 차원 vs. 노드 접근 수(범위 질의 성능)

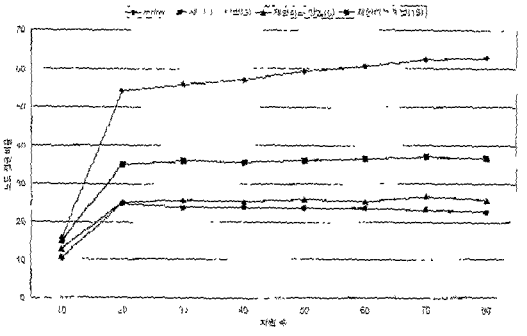

그림 28 차원 vs. 노드 접근 비율(범위 질의 성능

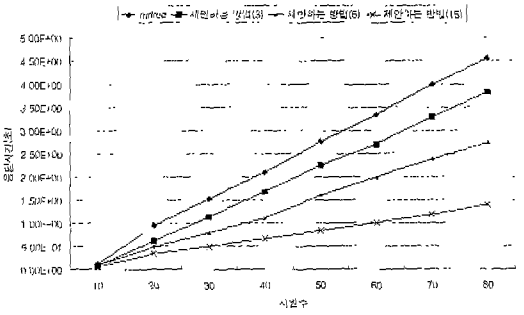

그림 29 차원 vs. 응답시간(범위 질의 성능)

그림 30과 그림 31은 9차원의 실 데이타를 가지고 실험한 결과를 보여준다. MR-트리와의 형평성을 맞추기 위해 서버 개수는 3이고 디스크의 수 역시 3으로 고정하였다. MR-트리에서는 범위질의와 최근접 질의에 대해서 제안하는 방법에서는 범위질의와 3개의 최근접 질의 방법 모두에 대해서 디스크 접근수와 응답시간을 측정하였다. MR-트리가 제안된 논문에서는 K-최근접 질의에 대한 언급은 없었지만 이 논문에서 실험을 위해 기본적인 K-최근접 질의를 구현하여 실험하였다. 그 결과 MR-트리나 제안하는 방법 모두 실데이타로 측정한 결과가 균등분포 데이타로 측정한 결과보다 좋게 나왔다. 일반적으로 다차원 색인구조에서 분포가 편중될수록 질의 성능이 좀더 좋게 나타나는 것과 일치하는 현상이다. 두 방법의 비교 결과는 대체적으로 제안하는 방법이 우수하였다. 특히, 응답시간 측면에서 MR-트리의 K-최근접 질의가 기타 다른 질의 방법에 비해서 2-3배 정도 나쁘게 나왔다. K-최근접 질의는 보통 깊이 우선 방식으로 트리를 순회하게 되어 있다. MR-트리에서는 매번 단말 노드를 접근할 때 마다 서버 간에 통신이 필요하다. 따라서 응답시간에 통신시간이 차지하는 비중이 커져서 응답시간 저하가 더 두드러진다고 해석해 볼 수 있다.

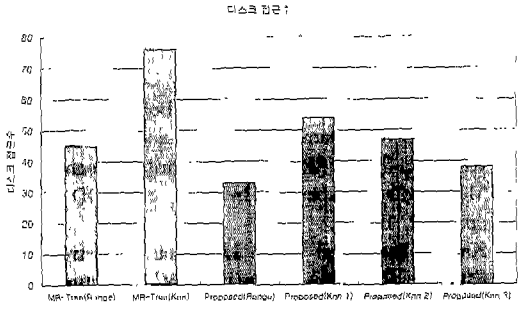

그림 30 제안하는 방법과 MR-트리의 디스크 접근수

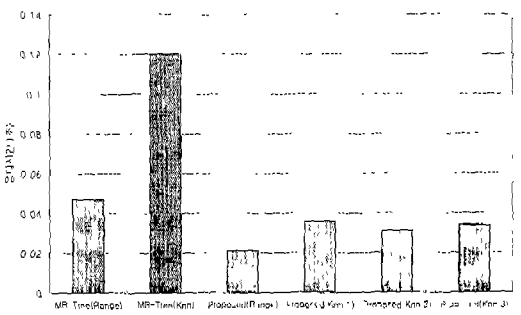

그림 31 제안하는 방법과 MR-트리의 응답시간

5. SAN에서의 구현

제안하는 병렬 고차원 색인 구조는 무공유(Shared Nothing)와 디스크 공유(Shared Disk) 환경 모두에 적용 가능하다. 무공유 환경에서는 각 서버가 다중 디스크를 갖는 환경에 쉽게 적용할 수 있고 디스크 공유 환경에서는 그림 2에서와 같이 각 서버가 접근할 수 있는 디스크를 미리 할당하여 적용할 수 있다. SAN은 구조적으로 공유디스크 병렬 환경과 매우 유사하다. 저장장치들은 특별한 네트워크(SAN:Storage Area Network)에 부착되어 있고 컴퓨터들 역시 이 네트워크에 부착되어 SAN에 부착된 디스크들을 자유롭게 접근할 수 있다.

하지만 제안하는 색인구조를 SAN 위에서 구현하기 위해서는 몇 가지 고려해야 할 점들이 있다. 일반적으로 SAN 운용 소프트웨어는 여러 디스크들을 마치 하나의 커다란 디스크처럼 보이게 하는 가상 디스크 시스템을 제공한다. 사용자에게는 SAN에 어떤 디스크들이 어떻게 부착되어 있는지가 투명하다. 제안하는 색인 구조 역시 이 가상 디스크 위에서 구현이 된다. 문제는 가상 디스크 시스템이 사용자가 요구하는 저장공간 할당요청을 투명하게 처리하기 때문에 물리적으로 원하는 위치에 데이타를 기록할 수 없다는 것이다. 따라서, 물리적으로 원하는 위치에 원하는 만큼 공간을 할당받기 위해서는 기존의 가상 디스크 인터페이스와는 좀 다른 인터페이스가 필요하다.

일반적으로 가상 디스크시스템에서는 원하는 양의 공간을 요청하면 어딘가에서 적절히 해당 양만큼의 공간을 할당하고 사용자는 그 할당된 공간의 논리주소를 가지고 읽기/쓰기를 수행한다. 하지만, 제안하는 색인구조를 구현하기 위해서는 원하는 공간의 크기 외에도 원하는 장치 이름을 같이 줄 수 있어야 한다. 그리고, 가상 디스크 내부에서도 이런 요청을 받아서 해당 장치로부터 원하는 크기의 디스크 공간을 할당하고 논리 주소를

부여하는 기능이 추가적으로 구현되어야 한다.

6. 결 론

이 논문에서는 병렬환경의 장점을 최대한 이용하는 병렬 고차원 색인 구조를 설계하고 이에 대한 성능 평가를 수행하였다. 제안한 병렬 고차원 색인 구조에서 각 서버는 디스크 그룹을 관리하고, 각 디스크 그룹에 엔트리를 할당하는 방법은 가장 간단한 라운드 로빈 방법을 사용한다. 하나의 노드를 여러 디스크에 분산시켜 팬-아웃을 증가시키고 트리의 높이를 줄임으로서 검색 성능을 향상시킨다. 각 디스크 그룹 내에서 노드 분할 방법은 각 레벨에서 페이지의 수가 가장 작은 디스크에 할당을 해서 노드를 다른 디스크에 분산시켜 병렬성을 최대화한다. 질의 처리 시 범위 질의 방법은 여러 노드의 병렬 입출력을 수행하는 방법을 제안한다. 그리고 K-최근접 질의 방법은 각 서버에서 K-최근접 질의를 수행하는 방법과, 범위 질의의 병렬성을 이용하여 K-최근접 질의의 성능을 향상시키기 위한 방법을 제안했다. 실험 결과 두 번째와 세 번째 방법이 첫 번째 방법보다 좋은 성능을 나타내는 것을 볼 수 있었다. 제안된 방법의 성능을 평가하기 위해 시뮬레이션을 하였다. 시뮬레이션 결과 기존 병렬 색인 구조(MR-트리)보다 제안된 병렬 고차원 색인 구조가 성능이 우수함을 보였고, 고차원의 데이타에 대해서도 효율적으로 색인할 수 있음을 보였다. 향후에는 디스크와 서버가 추가/삭제되는 경우 온라인으로 이를 처리하는 방법에 대해서 연구를 진행한다.

참 고 문 헌

[1] K.I. Lin, H. Jagadish, and C. Faloutsos, "The TV-tree : An Index Structure for High Dimensional Data," VLDB Journal, Vol 3, pp. 517-542, 1994.

[2] S. Berchtold, D. A. Keim and H-P. Kriegel, "The X-tree : An Index Structure for High-Dimensional Data," In Proc. 22nd VLDB Conf. pp. 28-39, 1996.

[3] D. A. White and R. Jain, "Similarity Indexing with the SS-tree," In Proc. ICDE, New Orleans, pp. 516-523, 1996.

[4] N. Katayama and S. Satoh, "The SR-Tree : An index structure for high dimensional nearest neighbor queries," In Proc. SIGMOD conf., pp. 369-380 1997.

[5] 이석희, 유재수, 조기형, 허대영, "CIR-Tree : 효율적인 고차원 색인기법", 한국정보과학회 논문지(B), 한국정보과학회 제26권 제6호, pp. 724-734, Jun 1999.

[6] J.T. Robinson, "The K-D-B-tree: A search structure for large multidimensional dynamic indexed." In Proc. ACM SIGMOD Conf., pp. 10-18, 1981.

[7] Lomet D. and Salzberg B, "The hB-Tree: A Robust Multiattribute Search Structure," In Proc. ICDE Conf., pp. 296-304, 1989.

[8] A. Henrich, H.-W. Six and P. Widmayer, "The LSD-tree: spatial access to multidimensional point and non-point objects," In Proc. VLDB Conf., pp. 45-53, 1989.

[9] M. Freeston, "The BANG file: a new kind of grid file," In Proc. VLDB conf., pp. 260-269, 1987.

[10] J. Nievergelt, H. Hinterberger, and K. Sevcik, "The grid file: An adaptable, symmetric multikey file structure." ACM Transactions on Database Systems(TODS), 1984.

[11] K. Chakrabarti and S. Mehrotra, "The Hybrid Tree : An Index Structure for High-Dimensional Feature Spaces," In Proc. ICDE conf., pp. 440-447, 1999.

[12] Aristides Gionis, Piotr Indyk and Rajeev Motwani, "Similarity Search in High Dimensions via Hashing," In Proc. VLDB Conf., pp. 518-529, 1999.

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

[14] Berchtold S., "Independent Quantization: An Index Compression Technique for High-Dimensional Data Spaces," In Proc. ICDE Conf., pp. 577-588, 2000.

[15] Ning An, Liujian Quian, Anand Sivasubramaniam and Tom Keefe, "Evaluating Parallel R-Tree Implementations on a Network of Workstations," In Proc. ACM GIS Conf., pp. 159-160, 1998.

[16] 조성훈, 김성주, 이준호, 이주영, 박석천, "SAN의 구조와 기술 요소", 정보처리학회지 제8권 제4호, pp. 19-28, 2001

[17] I. Kamel and C. Faloutsos, "Parallel R-trees," CS-TR-2820, UMIACS-TR-92-1, Computer Science Technical Report Series, University of Maryland, Collage Park, MD, 1992

[18] Stefan Berchtold, Christian Bohm, Bernhard Braunmuller, Daniel A.Keim and Hans-Peter Kriegel, "Fast Parallel Similarity Search in Multimedia Databases," In Proc. SIGMOD Conf., pp. 1-12, 1997.

[19] Kap S. Bang and Huizu Lu, "The PML-tree: An Efficient Parallel Spatial Index Structure for Spatial Databases," In Proc. ACM Conf., pp. 79-88, 1996.

[20] N. Koudas, C. Faloutsos and I. Kamel, "Declustering R-trees on Multi-Computer Architectures," Technical Research Report ISR 1994.
[21] Bernad Scnnitzer and Scott T.Leutenegger, "Master-Client R-trees: A New Parallel R-tree Architecture," In Proc. SSDBM Conf. pp. 68-77, 1999.
[22] Botao Wang, Hiroyuki Horinokuchi, Kunihiko Kaneko and Akifumi Makinouchi, "Parallel R-tree Search Algorithm on DSVM," In Proc. DASFAA Conf., pp. 237-245, 1999.
[23] Xiaodong Fu, Dingxing Wang, Weimin Zheng and Mciming Sheng, "GPR-Tree: A Global Parallel Index Structure for Multiattribute Declustering on Cluster of Workstations," IEEE, 1997.
[24] Roger Weber, "Parallel VA-File," VLDB, 1999.
[25] http://www.metastor.com/products/sans/E4400_datasheet.pdf.
[26] N. Beckmann, H.P. Kornacker, R. Schneider and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," In Proc. ACM SIGMOD Conf., pp. 322-331, 1990.
[27] A. Guttman, "R-Trees: A dynamic index structure for spatial searching," In Proc. ACM SIGMOD Conf., pp. 47-57, 1984.
[28] K. Lin, H. V. Jagadish, and C. Faloutsos, "The TV-Tree an index structure for high dimensional data," VLDB Journal, pp. 517-542, 1994.
[29] A. Henrich, "Improving the performance of multi-dimensional access structures based on k-d-trees," In Proc. Information and Knowledge Management Conf., pp. 68-75, 1996.
[30] T.Sellis, N. Roussopoulos and C. Faloutsos, "The R--Tree: a dynamic index for multi-dimensional objects," In Proc. VLDB Conf., pp. 507-518, 1987.
[31] Ibrahim Kamel and Christos Faloutsos, "Parallel R-trees," In Proc. ACM SIGMOD Conf., pp. 195-204, 1992.
[32] Stefan Berchtold, "Improving the Query performance of High-Dimensional Index Structure by Bulk Load Operation," In Proc. EDBT Conf., pp. 216-230, 1998.

박 춘 서

1999년 충북대학교 정보통신공학과(공학사) 2001sus 충북대학교 정보통신공학과(공학설가. 2001년 ~ 현재 한국전자통신연구원 연구원. 관심분야는 동시성제어, 저장 시스템, 멀티미디어 정보검색, 병렬처리 등.

송 석 일

1988년 충북대학교 공과대학 정보통신공학과(공학사). 2000년 충북대학교 정보통신공학과(공학석사). 2002년 충북대학교 정보통신공학과(박사과정 수료). 관심분야는 고차원 데이타 색인, 정보검색프로토콜(Z39.50), SGML, OODBMS, 저장 시스템, 회복기법, 동시성제어

신 재 룡

1996년 충북대학교 정보통신공학과(공학사). 1998년 충북대학교 정보통신공학과(공학석사). 2001년 충북대학교 정보통신공학과(박사과정 수료). 관심분야는 데이타베이스 시스템, 실시간 시스템, 멀티미디어 데이타베이스 등

유 재 수

1989년 전북대학교 공과대학 컴퓨터공학과(학사). 1991년 한국과학기술원 전산학과(공학석사). 1995년 한국과학기술원 전산학과(공학박사). 1995년 ~ 1996년 목포대학교 전산통계학과 전임강사. 1996년 ~ 현재 충북대학교 전기전자 및 컴퓨터공학부 부교수. 관심분야는 데이타베이스 시스템, 정보검색, 멀티미디어 데이타베이스, 분산 객체 컴퓨팅