論文 96-21-10-7

T*-트리:주기억 데이터베이스에서의 효율적인 색인기법

正會員 최 공 림*, 김 기 룡**, 김 경 창**

T*-tree: An Efficient Indexing Technique for Main Memory Database

Kong-Rim Choi*, Ki-Ryong Kim**, Kyung-Chang Kim** *Regular Members*

요 약

본 논문에서는 주기억 데이터베이스 시스템에서의 효율적인 데이터 처리를 위하여 T*-트리라는 새로운 색인구조를 제시한다. T*-트리 색인구조는 기존의 디스크를 기반으로 하는 색인기법과 달리 모든 데이터가 주기억장치에 적재되어 있는 시스템에서 보다 빠른 데이터 접근과 메모리 공간의 효율적인 사용을 위해 주기억 데이터베이스 시스템에서 주로 사용되고 있는 T-트리 색인구조의 장점은 그대로 계승하면서 단점을 보완한 인덱스 구조이다. 본 논문에서 제시하는 T*-트리는 데이터 아이템에 대한 검색과 저장공간의 활용면에서는 T-트리와 대동소이한 성능을 가지고 있으나, 범위 질의에서와 데이터의 삽입과 삭제시 중간노드에서의 노드간의 순회경로를 줄임으로써 보다 향상된 성능을 보여준다. 또한 T-트리와 스레드 이진트리를 조합하는 경우에는 순회경로가 다소 단축되지만 중간노드에서 자신보다 높은 레벨의 후속 노드로의 순회는 기존의 인오더 트리 순회에 의존하지만, T*-트리에서는 후위포인터를 이용하므로 직접순회가 가능하게 된다. 본 논문에서는 제안된 T*-트리의 구조와 T*-트리의 검색, 삽입 및 삭제 연산을 위한 알고리즘을 설명한 후, 기존의 T-트리와 성능분석을 실시하고 그 결과를 제시한다. 성능 분석결과 T*-트리는 데이터 검색의 경우 기존의 T-트리와 거의 동일한 성능을 보였으며, 삽입과 삭제등 색인구조의 변경시는 약간의 성능향상을 보였으나, 범위질의와 순차질의에서는 매우 향상된 성능을 나타냈다.

I. 서 론

최근 산업 자동화, 초고속 정보통신등의 여러분야에서 실시간 응용시스템의 요구가 증대되고 있으며, 컴퓨터 하드웨어 기술의 혁신으로 해마다 대용량의 메모리 칩이 계속 개발되고 있어, 가까운 장래에 수기가바이트의 용량을 갖는 주기억장치를 기대할 수 있게 되었다. 이러한 주기억장치 용량의 증가는 특정의 응용분야에서 전체 데이터베이스를 주기억 장치내에 상주시킨 채로 다양한 연산을 수행하고자 하는 사용자의 요구를 충족시키게 될 것이다. 따라서 실시간처리를 위한 주기억 데이터베이스 시스템에 관한 많은 연구가 진행되고 있으며 연구 결과도 다수 소개

*한국공항공단 전산실
**홍익대학교 전자계산학과
Dept. of Computer Science, Hong-Ik Univ.
論文番號:96186-0624
接受日字:1996年 6月 24日

되고 있다.[2, 4, 8, 9, 10, 11, 12] 기존의 디스크를 기반으로한 데이터베이스에서의 데이터구조가 디스크내 저장공간의 효율적인 사용과 디스크로의 접근을 최소화하는 것이 주된 목적인 반면에, 주기억 데이터베이스 시스템에서는 CPU사이클과 주기억 공간의 효율적인 사용을 위한 데이터 저장 및 색인구조가 주요한 문제로 제기된다. 주기억 데이터베이스에서는 데이터베이스가 주기억장치내에 상주하게 되므로 기존의 색인구조를 그대로 적용하는 것이 부적합할 수도 있다. 주기억장치를 기반으로 하는 색인구조의 목적은 가능한 메모리공간을 적게 사용하면서 CPU사용시간을 최소로 하는 것이며, 이러한 색인구조의 연구가 절실히 요구된다. 본 논문에서는 주기억 데이터베이스의 여러가지 관점 중, 주기억장치 내에 전체 데이터베이스를 상주시키는 것으로 가정하고, 이 상황에 적합한 보다 향상된 색인기법을 제안한다.

본 논문의 구성은, 2절에서는 관련연구로서 기존의 색인 구조인 T-트리를 설명하고, 3절에서는 본 논문에서 제안하는 T*-트리의 구조와 특성을 설명하고, 이의 연산을 위한 알고리즘을 제시하며, 각각의 연산에 대하여 예를 들어 설명하고, 4절에서는 기존의 색인구조인 T-트리와 T*-트리에 대한 성능비교를 제시한다. 그리고 마지막으로 5절에서 결론을 내린다.

Ⅱ. 관련연구

기존의 디스크 기반 데이터베이스 시스템에서의 트랜잭션 처리는 디스크에 저장된 데이터 접근을 위한 입출력 동작으로 인하여 전반적으로 실시간 응용분야의 요구를 충족시키기에 그 수행속도가 늦고 처리시간의 예측에도 많은 어려움이 있었다. 따라서 시스템의 높은 수행능력을 얻기 위한 접근 방식의 하나로 주기억장치내에 데이터베이스를 적재하려는 시도가 제기되었다. [1, 4, 5] 이제까지 연구된 주기억 데이터베이스는 주로 프로토타입 시스템이 소개되고 있으며 아직은 개념정립 단계로서 상용화된 제품은 없는 실정이다. 소개된 프로토타입의 시스템 중에서 Starburst, FLASH 및 Dali[2, 9, 10]등에서는 고정길이의 데이터아이템을 지원하는 관계형 데이터모델이 적용되고 있으며, 아직 주기억 데이터베이스 시스템에 적합한 데이터모델에 관한 연구가 미흡한 실정이다. 또한 기존의 디스크를 기반으로한 색인구조에서와는 달리 주기억 데이터베이스에 적합한 색인구조와 메모리 저장구조 및 질의처리의 효율화를 위한 임시릴레이션 및 질의처리 전략 등에 관한 연구가 절실히 요구되고 있다. 본 절에서는 기존의 디스크를 기반으로 한 색인구조들 중, 주기억장치 색인구조로서 널리 사용되고 있는 T-트리의 장점과 단점을 상세히 검토하고자 한다.

2.1 T-트리

새로운 데이터 구조인 T-트리[1]는 AVL-트리와 B-트리의 특성을 결합해서 새롭게 변형되어 나온 트리 구조로서 Lehman이 특히 주기억 데이터베이스의 효율적 운용을 위하여 제안하였다. T-트리는 한 노드안의 여러개의 데이터 아이템들을 가지는 B-트리의 특성과 이진검색의 AVL-트리 특성을 결합하여 만든 색인구조이다. [그림1]은 T-트리의 한 노드의 구조와 T-트리의 구성을 보여주고 있다. 한 노드에는 색인화일을 구성하는 유일한 키 값인 데이터 값과 그 아이템이 속한 아이템의 주소를 표시하는 포인터의 쌍으로 구성되는 여러개의 엘리먼트(element)들이 존재한다. 노드내의 모든 엘리먼트는 키 값의 크기에 따라 정렬되어 있고 가상 왼편의 엘리먼트가 가장 적은 키값을 가지며, 가장 오른편의 엘리먼트가 가장 큰 값을 갖게 된다. [그림2]는 2개의 서브트리를 가지고 있는 T-트리의 한 노드에서의 바운딩(bounding)관계를 보여주고 있다. 노드 내에서 가상 작은 아이템 보다 작은 값을 갖는 아이템으로 구성되는 왼쪽 서브트리와 가상 큰 값보다 큰 값을 가지는 아이템들로 구성되는 오른쪽 서브트리를 가진다. 이때 왼쪽 서브트리에서

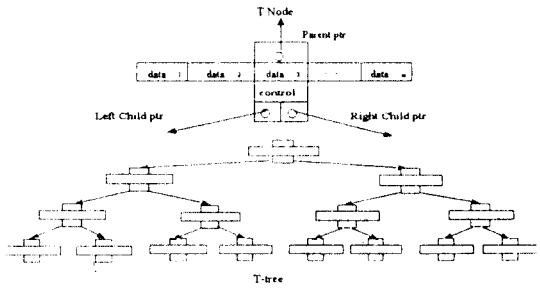


그림 1. T-트리

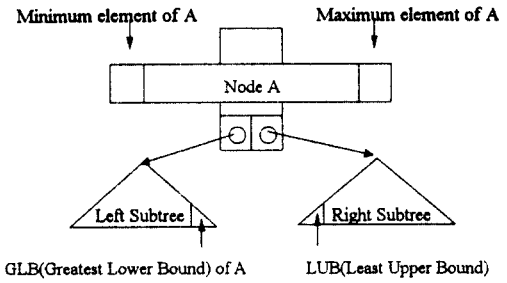


그림 2. 노드에서의 바운드

가장 큰 값을 GLB(the greatest lower bound)라 하고, 오른쪽 서브트리에서 가장 작은 값을 LUB(the least upper bound)라 한다. 모든 노드의 삽입과 삭제는 단말노드에서 만 이루어지며 이때 불균형 상태가 되면 AVL-트리처럼 로테이션연산을 수행해 균형 상태로 만든다.

T-트리는 주기억 데이터베이스 시스템에서의 성능 시험 결과[1, 2]는 저장공간의 효율성 및 정렬된 데이터의 처리시 다른 색인구조에 비하여 대체로 우수한 성능을 보여주고 있다. 그러나 T-트리 색인구조에서는 트리내에서의 노드간 순회를 인오더 순회에 의하므로, 범위질의나 순차질의의 수행시 접근하고자 하는 아이템을 가지지 않은 노드를 불필요하게 순회하게 된다. 또한 T-트리 내의 중간노드에서 데이터의 삽입(삭제)으로 인한 오버플로우(언더플로우)가 발생하게 되면, 오버플로우(언더플로우)된 최소아이템(최대아이템)을 인오더에의하여 GLB로(로부터) 이동시키게 된다. [그림3]은 범위질의시 인오더 순회로 인하여 검색할 데이터를 갖지않는 노드를 순회하는 모습

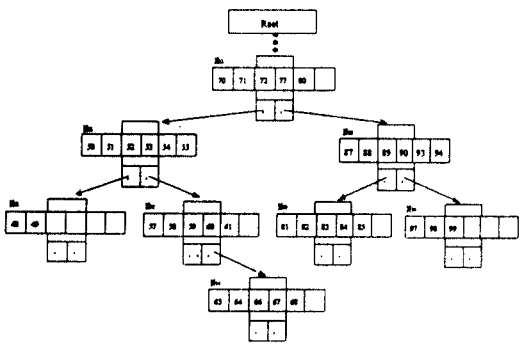


그림 3. T-트리에서 범위질의의 예

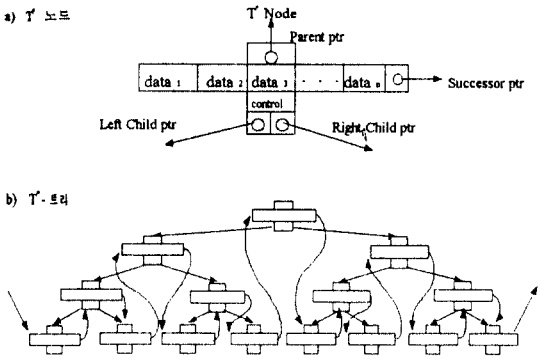


그림 4. T*-트리

을보여준다. 키값이 67과 82사이에 있는 모든 아이템을 검색하는 경우, 순회하는 노드는 첫 번째 N44로부터 N32, N21, N11, N22 및 N33로써, 여기에서 N32, N21과 N22노드는 검색 아이템을 보유하지 않은 노드들이다. 따라서 본 논문에서는 T-트리의 장점은 그대로 계승하면서 범위질의등에서의 단점을 보완할 수 있는 새로운 자료구조인 T*-트리를 제안한다.

Ⅲ. T*-트리 색인기법

3.1 T*-트리의 구조

T*-노드와 T*-트리의 구조는 [그림4]에서와 같다. T*-노드는 그림에서와 같이 T-노드에 하나의 후위포인터를 추가하여 차상위 크기의 아이템으로 구성된 후위노드를 지시하도록 하므로서 노드간의 크기순에 따른 단순연결리스트가 구성되도록하였다. 또한 노드내의 데이터 아이템의 삽입시 T-트리에서와는 달리 우측부터 차례로 저장되므로 빈공간은 항상 왼편에 있게 된다. 노드내의 각 엘리먼트들은 유일한 키값과 키값이 속한 데이터아이템의 주소로 구성되며 노드내는 아이템의 크기에 따라 사전 정렬된다. [그림4]에서의 T*-트리 구성도를 보면 추가된 후위포인터가 차상위 값들로 구성된 후위노드를 가르키고 있음을 알 수 있으며, 따라서 T-트리의 인오더 트리순회를 개선, 다음노드로 직접순회가 가능하게 되어 순회경로가 단축된다. 또한, 노드의 생성시에는 첫 번째 아이템이 최대값으로 우측부터 차례로 저장되도록 함으로서, 데이터의 삽입으로 인한 오버플로우의 경우 오버플로우 된 최대값을 LUB노드의 빈공간 중에

서 제일 오른편에 저장하게 되며, 삭제로 인한 언더플로우의 경우에도 LUB 노드로부터 최소값인 가장 왼편의 아이템을 바로우(borrow)하므로 노드내에서 추가적인 데이터 정렬과 이동이 생기지 않으며, 후위포인터를 이용하여 직접 LUB노드로 접근하게 되어 검색할 데이터가 없는 중간노드를 거치지 않도록 T*-트리의 구조 및 처리 알고리즘을 개선하였다. 아울러 T*-트리는 저장 공간의 효율적 이용측면에서도 기존의 T-트리와 비교하여 기존의 T-노드의 구조에 후위포인터가 추가되었을 뿐, 동일한 내부구조를 가지고 한 노드내에 많은 데이터 아이템이 저장되므로, 데이터아이템 당 포인터의 비율은 크게 변하지 않아 T-트리의 저장공간의 효율성에 대한 장점이 그대로 유지된다.

3.2 T*-트리의 기본 연산

3.2.1 T*-트리의 검색 알고리즘

T*-트리에서 단일키값에 의한 검색연산은 T-트리와 동일하며, 이진 트리에서의 검색과 유사하지만, 다른점은 이진 트리에서 한개의 값으로 비교하는 반면, T*-트리에서는 노드내의 최소값과 최대값을 가지고 비교한다는 점이다.

T*-트리에서 데이터 아이템 X를 검색하는 알고리즘은 다음과 같다.

(1)검색은 항상 트리의 루트(root)부터 시작한다.

(2)***if*** (X가 노드의 가장 작은 값보다 작으면) ***then*** 왼쪽 서브트리로 이동, 계속 검색

else if (X가 노드의 가장 큰 값보다 크면) ***then*** 오른쪽 서브 트리로 이동, 검색

else 현재 노드내에서 이진탐색으로 X를 찾는다.

T*-트리 내의 아이템들은 정렬되어 있어 이진탐색이 가능하며, 노드내에서 아이템을 찾지 못하거나, 검색 값을 바운딩하는 노드를 찾지 못하게 되면 검색을 실패하게 된다.

3.2.2 T*-트리의 삽입 알고리즘

T*-트리 삽입 알고리즘은 아이템의 삽입에 따라 새로운 노드가 생성되면 후위포인터가 변경, 추가 되는 것 외에는 T-트리와 대동소이하나, T-트리에서는 노드내의 데이터 아이템들이 우측에서 부터 차례로 입력되고, 중간노드에서 오버플로우가 발생하면 노드의 최소아이템이 인오더에 의한 트리순회를 하는데 반하여, T*-트리에서는 노드내의 데이터 아이템들이 좌측부터 입력되며 오버플로우가 발생하면 최대값이 후위포인터를 이용하여 직접 LUB노드로 이동, 최소값으로 저장되므로 순회경로의 단축이 가능하게 된다. 삽입연산은 삽입할 아이템의 값을 바운딩하는 노드를 검색하면서 시작되며, 새로운 아이템이 바운딩 노드에 삽입이 끝나면 트리의 균형을 검사하고, 불균형이 발생할 경우 재균형연산을 시행한다.

T*트리에서 데이터 아이템 X를 삽입하는 알고리즘은 다음과 같다:

(1)아이템을 삽입할 위치인 바운딩 노드를 검색한다.

(2)***if*** (노드내에 삽입 공간이 있으면) ***then*** 아이템 X를 적정위치에 삽입하고 종결

else 노드내에서 가장 큰 아이템을 제거하여 임시 기억장소에 저장하고 아이템 X를 노드내의 적절한 위치에 삽입한 후, 후위포인터를 따라 LUB노드로 이동, 임시장소의 값을 최소값으로 삽입한다.

(3)트리전체를 점검하고도 아이템 X의 바운딩노드를 찾지 못한 채 단말노드를 만나는 경우

else if (검색경로상의 마지막 노드인 단말노드에 삽입공간이 있으면)

then 단말노드의 최소값으로 저장한다.

else 새로운 단말노드를 생성하고, 후위포인터를 적절한 노드를 지시하도록 변경 또는 추가한 후, 삽입 아이템 X를 새로운 노드의 가장 오른쪽 공간에 최소값으로 저장한다.

(4)***if*** (새로운 단말노드가 첨가) ***then***

while (단말노드부터 루트까지의 경로에 대하여) ***do*** {

트리의 균형을 검사하여

if (깊이의 차이가 1보다 크게 되어 불균형 발생)

then 로테이션 연산을 수행

}

3.2.3 T*-트리의 삭제 알고리즘

삭제 알고리즘은 삽입에서와 같이 먼저 삭제될 아이템을 검색한 후, 삭제연산을 수행한다. 다만 아이템

의 삭제로 인하여 노드의 삭제가 발생하는 경우, 노드간 인오더 관계가 지속되도록 후위포인터를 변경하게 되며, 단말노드로부터 루트까지의 경로에 대한 트리의 균형상태를 점검하여 필요시 재균형을 위한 로테이션 연산을 수행하여 균형을 유지토록 한다. 또한 중간노드에서 아이템이 삭제되어 노드내 아이템 수가 최소수 보다 작아지게 되면 언더플로우 상태가 되어 LUB노드로부터 최소값을 빌려와 최대값으로 저장하게 되므로 노드내에서의 아이템의 재정리가 필요치 않으며, 후위포인터를 이용하여 직접 GLB노드로 이동하게 되므로 노드의 순회경로가 단축되어 처리속도가 향상된다.

T*-트리에서의 데이터 아이템 X를 삭제하는 연산 알고리즘은 다음과 같다.

(1)삭제할 값을 바운딩한 노드를 검색한다.
(2)*if* (노드안에 삭제할 값이 없다면) *then* 오류(error)를 출력하고 종결한다.
else if (삭제 연산이 언더플로우를 야기 시키지 않는다면)
then 그 값을 삭제하고 종결한다.
else { /* 언더플로우가 야기 */
if (현노드가 내부노드인 경우)
then 아이템을 삭제하고 후위포인터가 지시하는 LUB노드로 부터 최소값을 가지고 오며
else if (단말노드인 경우) *then* 아이템을 삭제한다.
(3)*if* (반단말노드이면서 다른 단말 노드와 결합시킬 수 있다면)
then 두개의 노드를 한 개의 노드로 병합하고 다른 한 노드는 버린다.
후위 포인터 또한 병합한다.
단계 (5)의 균형 점검을 실시한다.
(4)*if* (단말노드가 삭제로 인하여 비지 않으면) *then* 종결한다.
else 빈노드를 소거하고 후위포인터를 변경시키고 트리의 재균형을 위하여 단계 (5)을 처리한다.
(5)*if* (단말노드가 삭제) *then*
while (단말노드부터 루트까지의 경로에 대하여) *do* {
트리의 균형을 검사하여
if (깊이의 차이가 1보다 크게 되어 불균형 발생)
then 로테이션 연산을 수행
}

Ⅳ. T*-트리의 성능분석 및 평가

색인구조간의 성능비교는 주로 각각의 색인구조에 대한 저장공간의 효율적인 운용측면과 데이터베이스에 대한 질의처리를 위한 다양한 연산 중에서 색인구조를 이용한 데이터 아이템의 검색, 삽입 또는 삭제하는 기본적인 연산시간의 비교등의 실질적 환경에서의 성능 평가로서 비교된다.

4.1 성능분석 환경

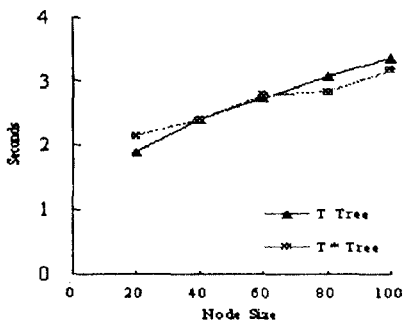
본 논문에서는 기존의 T-트리와 제안된 T*-트리 두 가지의 색인구조를 구현하여 성능 평가를 실시하였다. 성능평가를 위한 시험환경은 32메가바이트의 메모리를 가진 SUN-SPARC-10 시스템에서 각각의 색인구조를 C언어로 구현하고, 색인구조를 구성하는 데이터아이템을 랜덤수발생기(random number generator)에 의하여 사전에 생성 수개의 화일을 구성한 후 동일한 환경에서 같은 파일을 사용하여 두 색인구조의 처리시간을 산출하였다.

색인구조의 성능시험을 위한 연산형태는 색인구조를 이용한 데이터베이스의 다양한 연산 중에서, 먼저 데이터베이스의 구조를 변경시키는 삽입, 삭제 및 수정연산 중 기본이되는 삽입과 삭제연산과, 여러 가지의 질의형태 중 기본이 되는 단항연산 중 단순질의인 검색연산과 범위질의 연산에 의하여 실시한다. 시뮬레이션 방법은, 먼저 일정한 연산량에 노드의 크기를 변경시키면서 처리시간의 변화를 조사 하며, 두 색인구조에 대하여 상호비교한다.

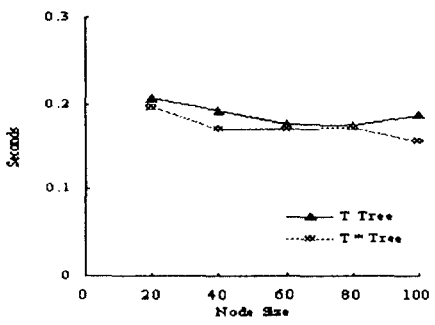
4.2. 성능분석 결과

[그래프1]에서 부터 [그래프4] 까지는 각각의 색인구조에 노드의 크기를 20, 40, 60, 80 및 100으로 변경시키면서 수행한 시험결과를 나타내고 있다. [그래프1]은 랜덤수발생기에 의하여 사전에 만들어진 중복된 색인 값이 없는 유일한 색인파일을 차례로 삽입시킴

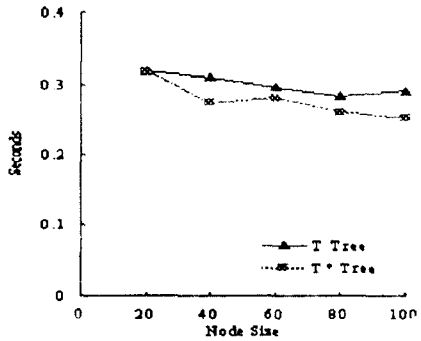
으로서 각각의 트리구조를 구축하는 경우로서, 시험결과 처리시간이 노드의 크기나 삽입되는 아이템의 수에 관계없이 거의 비슷한 처리시간을 보였다. 이는 아이템의 삽입시 T*-트리가 새로운 노드의 생성시마다 후위포인터를 추가 변경하는 작업이 필요한 반면, 오버플로우가 발생하는 경우에는 후위포인터를 이용하여 LUB노드로 직접 순회가 가능하므로 처리경로를 단축시킬 수 있기 때문이다. [그래프2]에서는 생성된 T*-트리와 T-트리로부터 노드의 크기를 변경시키면서 2000개의 데이터 아이템을 삭제처리하는 시간을 비교한 것이다. 시험결과 각트리에서의 처리시간은 약간의 차이는 있으나 거의 비슷하게 나타났다. 이는 T*-트리가 노드의 삭제시 후위포인터를 변경하는 오버로드가 발생하는 대신에 아이템의 삭제로 인한 언더플로우가 발생하는 경우 LUB로의 순회경로가 후위포인터의 사용으로 단축되기 때문으로 생각된다. [그래프3]은 삽입으로 구성된 각각의 색인구조로 부터의 검색시간을 비교한 것으로 그래프에서 나타난 바와 같이 거의 동일한 결과를 보여준다. 이는 각각의 색인구조에서의 검색방법은 동일한 알고리즘에의하여 수행되기 때문이다. 마지막으로 [그래프4]에서는 순차검색 및 범위질의에 대한 성능의 비교를 위하여 각 경우에서 연속된 데이터 아이템 100개를 검색하는데 소요되는 시간을 측정한 것으로서 그래프에서 보는 바와 같이 트리의 크기 및 처리되는 아이템의 수에 따라 다소 차이는 있으나 T-트리에 비하여 85%에서 95%정도의 연산시간 감소효과를 보여주었다. 이상의 성능시험 결과, 각각의 그래프에서 보는 바와 같이 T*-트리는 데이터 검색의 경우 기존의 T-트리와 거의 동일한 성능을 보였으며, 삽입과 삭제등 색인구조의 변경시는 미세한 성능향상을 보였으나, 범위질의와 순차질의에서는 T-트리의 연산 시간에 비하여 약 90%정도의 연산시간 감소 효과를 보여, 매우 향상된 성능을 보여주고 있다.



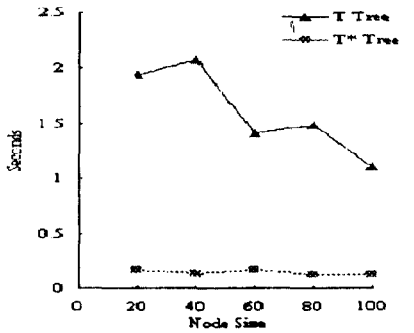
[그래프 1] Insert 10,000 Items (Vary Node Size)



[그래프 2] Delete 2,000 Items (Vary Node Size)



[그래프 3] Search 4,000 Items (Vary Node Size)



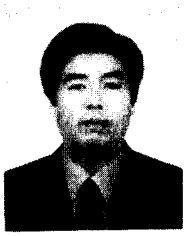
[그래프 4] Range Search 100 Data Items

V. 결 론

본 논문에서는 주기억 데이터베이스(MMDB) 시스템에서의 효율적인 데이터 처리를 위하여 T*-트리라고 하는 새로운 인덱스 구조를 제시하였다. T*-트리 색인구조는 기존의 디스크를 기반으로 하는 색인기법과 달리 모든 데이터가 주기억장치내에 적재되어 있는 시스템에서 보다 빠른데이터 접근과 메모리공간의 효율적 활용을 위하여 고안된 색인 구조로서, 이제까지 주기억 데이터베이스시스템에서 주로 사용되고 있는 T-트리 색인구조의 장점을 그대로 계승하면서 범위질의등의 단점을 보완한 것이다. 본 논문에서는 제안된 T*-트리의 구조와 T*-트리의 검색, 삽입 및 삭제 연산을 위한 알고리즘을 제시하고, 기존의 T-트리와 논리적인 성능비교와 이의 입증을 위한 성능분석을 실시하였다. 성능 분석결과 T*-트리는 데이터 검색의 경우 기존의 T 트리와 거의 동일한 성능을, 삽입과 삭제등 색인구조의 변경시는 약간의 성능향상을, 범위질의와 순차질의에서는 매우 향상된 성능을 나타내고 있슴이 입증되었다. 이는 T*-트리가 T-트리 구조에 후위포인터를 추가하여 빠르게 다음 순서의 노드에 접근할 수 있도록 노드내에서의 데이터 아이템의 저장순서를 개선하고, 중간노드에서 데이터 아이템의 삽입으로 인한 오버플로우나 삭제로 인한 언더플로우시 이동 대상 아이템을 변경하여 추가된 후위포인터를 최대한 활용가능케 함으로서, 순차연산과 범위질의에서의 성능향상은 물론 삽입과 삭제연산시의 처리속도를 개선하였기 때문이다.

참 고 문 헌

1. Tobin J. Lehman, Michael J. Carey "A Study of Index Structures for Main Memory Database Management Systems", in Proceedings 12th Int'l. Conf. on Very Large Databases, Kyoto, Aug. 1986, pp. 294-303
2. Tobin J. Lehman, Eugene J. Shekita, Luis-Felipe Carera "An Evaluation of Starburst's Memory Resident Storage Component", IEEE Trans. on knowledge and Data Eng, vol. 4, december 1992.
3. M. Leland and W. Roome, "The Silicon Database Machine", Proc, 4th Int'l. Workshop on Database Machines, Grand Bahama Island, March 1985.
4. Tobin J. Lehman, Michael J. Carey "Query Processing in Main Memory Database Management systems", Proc. ACM SIGMOD Conf., May 1986. pp239-250.
5. David J. Randy H. Katz, Frank Olken, Leonard D. Shapiro, Micheal R. Stonebraker, David Wood "Implementation Techniques for Main Memory Database Systems", In Proc. of ACM SIGMOD, Boston, 1984.
6. A. Aho, J. Hopcroft and D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley Publishing Company, 1974.
7. D. Comer, "The Ubiquitous B-tree", Computing surveys 11, 2 (June 1979)
8. Margarte H. Eich "Main Memory Database Research Directions" Technical Report 88-CAE-35
9. H. V. Jagadish, Daniel Lieuwen, Rajeev Rastogi, Avi Silberschatz "Dali:A High Performance Main Memory Storage Mnager", Proc. of the 20th VLDB Conf., Santiago, Chile 1994, pp48-59
10. Kaist "FLASH:A Main Memory Storage System", KAIST Tech. Report, 94-257-7-1.
11. Alfns Kemper, Guido Moerkotte, "Physical Object Management", Next-Generation Database Technology, chapter 9.
12. Eugene J. Shekita, Michael Carey "A Performance Evaluation of Pointer-Based joins", Proc. ACM SIGMOD Int'l Conf., Atiantic city, NJ, May 1990, pp300-311
13. Perlis, A. and Thornton, C., "Symbol Manipulation by Threaded Lists", CACM, Vol. 3, No. 4, April 1960, pp195-204
14. Leonard D. Shapiro, "Join Processing in Database Systems with Large Memories", ACM Transactions on Database Systems, Vol. 11, No. 3, September, 1986, pp239-264
15. A. Amman, M. Hanrahan and R. Kishnamurthy, "Design of memory Resident DBMS", Proc. IEEE COMPCON, San Francisco, Feb 1985.



최 공 림(Kong-Rim Choi) 정회원

1870년 3월~1974년 2월 : 연세대학교 전자공학과 졸업(공학사)

1985년 3월~1987년 8월 : 연세대학교 전자계산공학과(공학석사)

1992년 3월~1995년 2월 : 홍익대학교 대학원 박사과정 전자계산전공 수료

1980년 5월~현재 : 한국공항공단 전산실 재직중

김 기 룡(Ki-Ryong Kim) 정회원

제21권 7호 참조

김 경 창(Kyung-Chang Kim) 정회원

제21권 7호 참조