Hyper-TH : 실시간 주기억장치 데이터베이스 시스템을 위한 색인기법

민 영 수* · 신 재 룡* · 이 병 엽** · 유 재 수***

Hyper-TH : An Index Mechanism for Real-Time Main Memory Database Systems

Young Soo Min* · Jae Ryong Shin* · Byoung Yeop Lee** · Jae Soo Yoo***

Abstract

In this paper, we propose an efficient index mechanism for real-time main memory database systems. Existing main memory index structures based on the tree can effectively support range searches. However, it doesn't guarantee the real-time characteristic because difference between the access time of a node and an average access time can be high. The index structures based on the hash have always a regular random access time on the simple searches and that speed is very fast. However they do not support range searches. To solve such problems, we propose a new index mechanism called Hyper Tree-Hash (Hyper-TH) that combines ECBH (Extendible Chained Bucket Hashing) and T*-tree. ECBH can be dynamically extended and has a very fast access time. T*-tree effectively supports the range searches. We show through our experiments that the proposed mechanism outperforms existing other index structures.

* 충북대학교 정보통신공학과
** 대우정보시스템 e-솔루션사업팀 차장
*** 충북대학교 전기전자 및 컴퓨터공학부 부교수

1. 서 론

핵발전소 제어, 공정 제어, 항공기 제어, 무기 체계, 우주선의 운항 및 유도와 같은 실시간 응용들에 사용되는 실시간 시스템은 계산의 논리적 정확성만을 다루는 기존의 시스템과는 달리 실시간 제약 조건인 종료시한(deadline)을 만족시켜야 한다. 실시간 시스템은 빠른 수행 능력뿐만 아니라 실시간 응용들을 만족시키기 위해 시간 요구 조건을 만족시키면서 종료시한 내에 결과를 만들어 내는 시기적절함(timeliness), 시스템 명세에 정의된 고장이나 작업 부하 조건에서 작업의 종료시한을 보장하는 예상가능성(predictability)과 같은 실시간 특성을 지원해야 한다[1]. 빠른 수행 능력은 실시간 시스템에 있어서 엄격한 시간 제약 조건을 만족시키는데 도움을 주기는 하지만 빠른 수행 능력을 가진다고 해서 반드시 실시간 특성을 보장할 수 있는 것은 아니다. 결과적으로 실시간 시스템은 빠른 수행 능력보다 실시간 특성을 보장할 수 있는 능력이 더욱 중요하다고 볼 수 있다.

최근 실시간 응용들은 더욱 복잡해지고 방대한 양의 데이터 접근을 요구하고 있으며, 데이터에 대한 조직적 관리를 필요로 하고 있다. 데이터베이스 관리 시스템은 이러한 데이터의 조직적 관리 수단을 제공할 수 있기 때문에 실시간 시스템과 자연스러운 통합이 이루어졌다. 이렇게 통합된 실시간 데이터베이스 시스템은 다양한 응용들을 위해 실시간 제약조건을 포함한 데이터베이스 연산들을 제공한다. 하지만 기존의 디스크 기반 데이터베이스 시스템은 디스크에 저장되어 있는 자료의 입출력 시간으로 인해 실시간 시스템에서 요구하는 엄격한 시간 제약 조건을 만족시키기에 역부족이다. 따라서 빠른 응답 시간과 높은 트랜잭션 처리율을 제공하는 주기억장치 데이터베이스 시스템이 필요하게 되었다. 최근 하드웨어 기술의 비약적인 발전으로 반도체 메모리의 밀도가 높아지고, 가격이 낮아지면서 대용량 데이터를 주기억장치에 모두 적재하는 것이 가능하게 되었다. 따라서 이러한 하드웨어 기술을 기반으로 대용량 데이터를 조직적으로 관리하고 실시간 특성에 맞는 데이터베이스 연산을 지원하는 주기억장치 데이터베이스 시스템의 필요성이 높아졌다[2].

주기억장치 데이터베이스 시스템에서 사용할 주기억장치 기반 색인 구조는 기존의 디스크 기반 색인 구조와 다르다. 색인 구조면에서 디스크 접근 회수를 줄이고 디스크 저장공간을 효율적으로 사용하는 측면이 중요시되는 디스크 기반 색인 구조와 달리 주기억장치 기반 색인 구조는 주기억장치와 CPU를 효과적으로 사용하면서 전체 계산 시간을 줄이는 것에 중점을 두고 있다. 또한 주기억장치 기반 색인 구조는 색인되는 데이터 자체를 저장하지 않고 그 데이터가 저장되어 있는 위치 정보를 저장하기 때문에 가변길이 데이터에 대한 저장 문제를 해결할 수 있고 저장공간을 효율적으로 사용할 수 있는 장점을 갖는다[3][4].

실시간 주기억장치 기반 색인 구조는 최악의 수행 시간이 항상 특정 시간 내에 놓여질 수 있도록 예상가능성을 보장해야 하며 일반적인 데이터베이스 연산 처리가 가능해야 한다. 본 논문에서는 실시간 특성을 보장하기 위해 해시 기반 색인 구조인 ECBH와 범위 검색을 효과적으로 지원하기 위해 트리 기반 색인 구조인 T*-트리를 상호보완적으로 결합시킴으로써 실시간 주기억장치 데이터베이스 시스템을 위한 효율적인 색인 기법을 제시하고 성능 평가를 통해 실시간 특성을 지원하면서 기존의 다른 색인 구조보다 성능이 우수함을 증명하고자 한다.

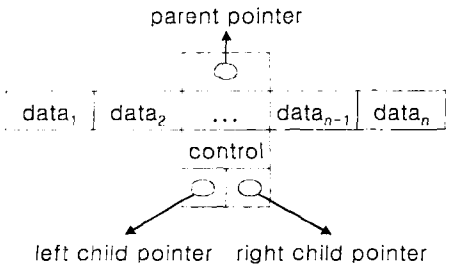
본 논문의 구성은 다음과 같다. 2장에서는 관

련 연구로써 기존 주기억장치 상주형 색인 구조들의 특성 및 문제점을 제시하고, 3장에서는 Hyper-TH의 기본 구조와 검색, 범위 검색, 삽입, 삭제연산 알고리즘을 설명한다. 4장에서는 제안하는 색인 구조와 기존의 색인 구조인 T-트리, T*-트리의 성능을 평가하고 분석한다. 마지막 5장에서는 결론을 맺고 향후 연구 방향을 제시한다.

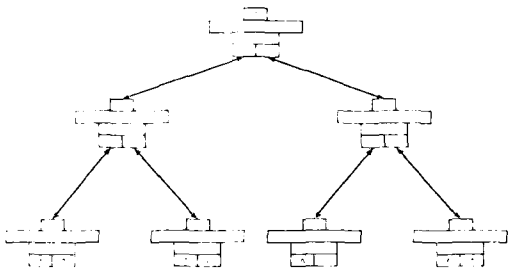
2. 관련 연구

2.1 T-Trees

T.J. Lehman과 M.J. Carey에 의해 제안된 T-트리는 AVL-트리[5]와 B-트리[7]를 발전시킨 데이터 구조로서 하나의 노드가 여러 엘리먼트를 갖는 이진 트리이다[3]. 따라서 T-트리는 (그림 1)과 같이 하나의 노드에 많은 엘리먼트를 포함하므로 저장공간 활용률이 좋다 또한, 한 노드 안에 있는 엘리먼트들이 정렬되어 있기 때문에 AVL-트리의 고유한 이진 검색 기능처럼 단순히 최소 값 엘리먼트와 최대 값 엘리먼트의 비교를 통해 검색이 이루어질 수 있다. 일반적인 트리에서 삽입과 삭제는 데이터의 이동을 필요로 하지만 T-트리에서는 대부분 단일 노드 내에서만 이동되므로 삽입, 삭제에 좋은 성능을 갖는다. 재균형 연산은 AVL-트리와 유사한 회전을 사용한다. 하나의 노드에 많은 엘리먼트가 있으므로 AVL-트리보다 훨씬 적은 재균형 연산이 일어나지만, AVL-트리의 재균형 연산을 적용할 때 T-트리의 내부 노드가 가득 차지 않는 경우가 발생할 수 있다. 이러한 문제를 해결하기 위해 T-트리에서는 특별 재균형 연산을 수행한다. 특별 재균형 연산은 회전을 하기 전에 일부 데이터를 이동시켜서, 회전 후 T-트리의 내부 노드가 가득 차도록 만든다.



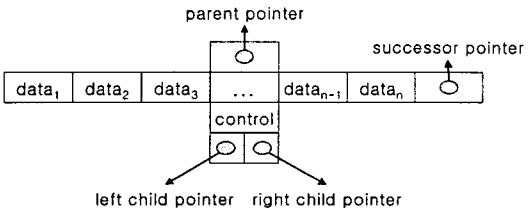
(그림 1) T-노드의 구조



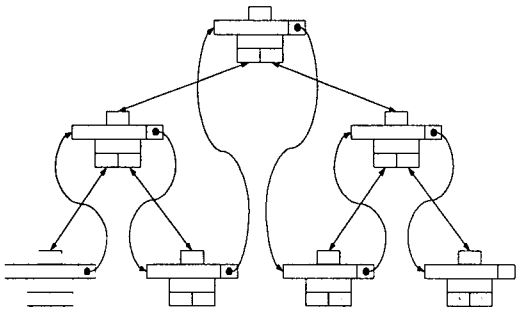
(그림 2) T-트리의 구조

2.2 T*-trees

K.R. Choi와 K.C. Kim은 중위 순회를 사용해서 범위 검색을 수행하는 T-트리의 성능을 연결 리스트를 통한 순차 검색을 수행해서 향상시킨 T*-트리를 제안하였다[8]. T*-트리는 T-트리와 거의 동일하다고 볼 수 있지만 범위 질의나 순차 검색과 같은 질의에 대해 트리를 순회하기보다는 간단한 연결 리스트를 통해 순차 검색이 가능하도록 (그림 3)과 같이 각 노드의 끝에 후임자 노드에 대한 참조 포인터를 갖도록 설계되었다. 또한 삽입에 있어서 삽입되는 위치의 노드가 가득 찬 경우에, T-트리는 한 노드의 가장 작은 값을 하위 노드에 재 삽입하는 반면에, T*-트리는 한 노드의 가장 큰 값을 하위 노드에 재 삽입한다.



(그림 3) T*-노드의 구조



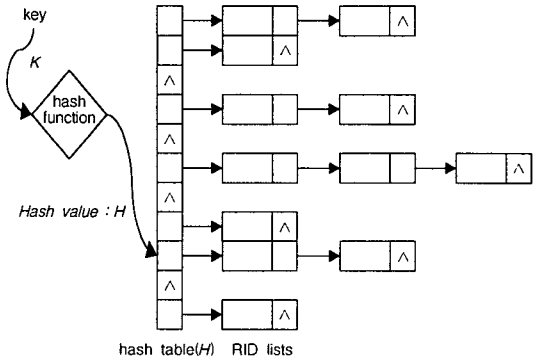
(그림 4) T*-트리의 구조

2.3 CBH(Chained Bucket Hashing)

CBH는 메모리에 저장된 정적 파일에 대해 가장 빠른 임의 접근을 제공하는 것으로 알려져 있다[9]. 또한 CBH는 메모리와 디스크에서 모두 사용되는 정적 구조로 해시 테이블의 크기가 동적으로 변하지 않는 정적인 환경에서는 매우 빠른 성능을 갖는다. 그러나 해시 테이블의 크기가 미리 결정되어져야 하는 정적인 구조이므로 동적인 환경에는 적합하지 못하다.

(그림 5)와 같이 해시 함수를 이용해서 키 값을 해시 테이블 범위의 주소 값으로 변환하게 되고 그 주소는 해시 테이블의 엔트리를 찾는데 사용되고, 이 엔트리는 같은 해시 주소를 가지는 식별자들의 리스트를 가리킨다. 그리고 주어진 키 값에 대한 해시 값 계산 시간 및 해시 테이블의 엔트리를 찾아내는 시간은 항상 일정하다.

균일 해싱 함수일 때, 해시 테이블의 크기를 D, 삽입되는 키의 수를 m이라 하면 CBH의 체인 길이는 $1+m/D$로 D가 고정되어 있으면 탐색 비용은 m의 증가에 따라 선형적으로 증가하게 된다. 이러한 선형적인 증가는 해시 기반 접근 방법의 주요한 목적인 O(1)의 탐색 시간을 위반한다. 뿐만 아니라 균일 해싱 함수를 충분히 만족하지 못하는 경우에는 최악의 경우 O(m)의 탐색 시간이 걸리게 된다. 따라서 테이블이 너무 작으면 성능이 나빠질 수 있고 반대로 너무 크면 공간의 낭비를 초래하게 된다.



(그림 5) CBH의 구조

2.4 ECBH(Extendible Chained Bucket Hashing)

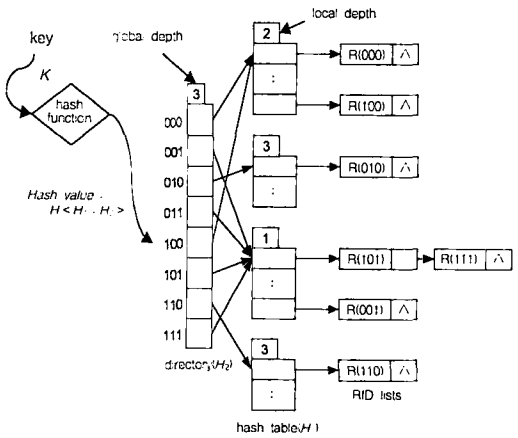
ECBH는 주기억장치 데이터베이스 시스템의 동적 파일을 위해 CBH와 EH[9]를 상호보완적으로 통합한 것이다[10]. ECBH는 CBH의 높은 성능과 EH의 점진적인 확장 가능성을 동시에 상속받은 우수한 성능을 나타내는 방법이다.

임의의 키 K의 해시 값 $H<H_1 : H_2>$는 해시 테이블의 엔트리를 할당하기 위해 사용하는 H_1과 디렉토리의 엔트리를 할당하기 위해 사용하는 H_2로 구성된다. 이때 H_1을 위해 확보되는 비트의 수는 $\lceil \log C \rceil$이다. C는 해시 테이블의 크기이다. 그리고 삽입되는 키의 수가 m이고 선행하는 RID_i의 체인 수가 L_{RIDi}일 때 평균 체인 길이 L_{avg}는 식 (1)과 같다.

$$L_{avg} = \frac{1}{m} \sum_{i=1}^{m} L_{RIDi} \quad (1)$$

Global depth가 g, local depth가 d일 때 최대 global depth인 g_{max}는 식 (2)와 같으며, 이때 h는 H의 비트 수를 나타낸다. 또한 ECBH의 평균 체인 길이를 제어하기 위한 변수는 l이다. 평균 체인 길이가 l보다 커지면 분할이 수행된다.

$$g_{max} = h - \log C \quad (2)$$



(그림 6) ECBH의 구조

2.5 Hybrid Tree-Hash(Hybrid-TH)

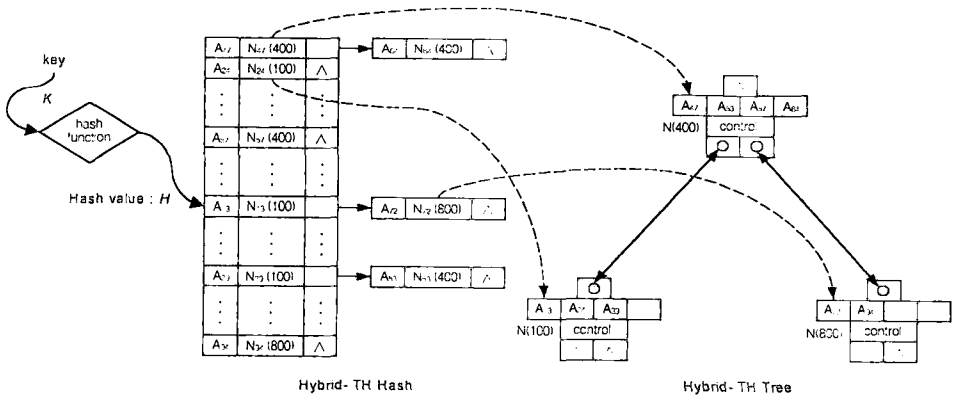
Hybrid-TH는 실시간 주기억장치 데이터베이스 시스템에서 예상 가능성을 보장하기 위해 해시 기반 색인 구조인 CBH와 트리 기반 색인 구조인 T-트리를 상호보완적으로 결합한 색인 기법이다[11]. Hybrid-TH는 CBH의 빠른 검색 성능과 예상 가능한 접근시간, T-트리의 범위 검색을 지원하지만, CBH가 해시 테이블 크기를 미리 결정해야 하는 정적인 구조이므로 동적인 환경에는 적합하지 못하다. 또한 범위 검색에서는 T-트리의 중위 순회를 사용하므로 T*-트리의 연결 리스트를 사용한 순차 범위 검색보다 성능이 좋지 못하다.

3. 제안하는 실시간 색인 구조

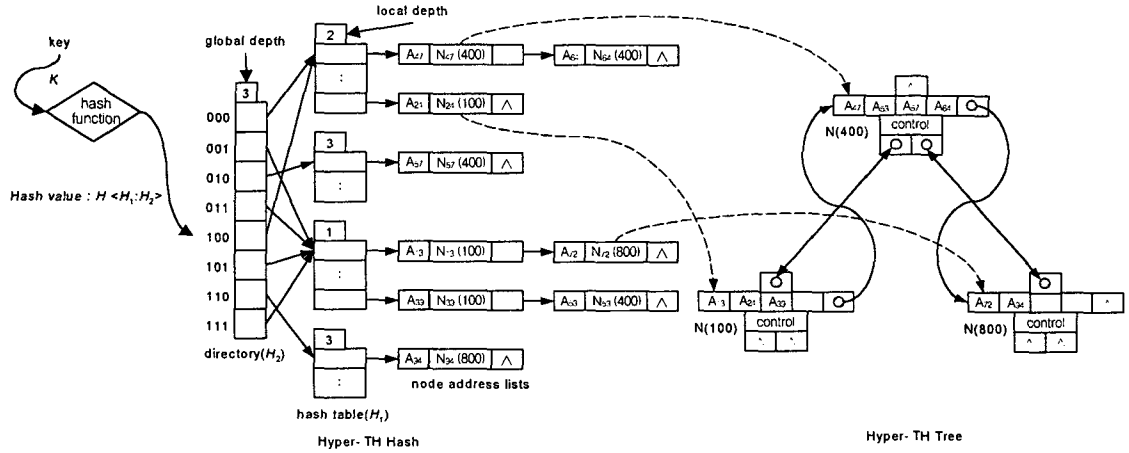
3.1 Hyper-TH의 기본 구조

Hyper-TH는 실시간 주기억장치 데이터베이스 시스템에서 실시간 특성을 보장하기 위해 범위 검색에 효과적인 T*-트리 색인 구조와 동적 확장이 가능하고 데이터에 대한 접근시간이 빠른 ECBH 색인 구조를 상호보완적으로 결합한 형태를 취한다.

Hyper-TH 트리는 기존의 T*-트리 색인 구조와 동일하고, Hyper-TH 해시는 기존의 ECBH 색인 구조와 유사하지만 데이터 아이템



(그림 7) Hybrid-TH의 구조



(그림 8) Hyper-TH의 구조

의 주기억장치 주소만 저장되어 있던 연결 리스트에 데이터 아이템의 주기억장치 주소가 저장되어 있는 T*-노드 주소를 추가적으로 저장하는 것이 다르다. Hyper-TH 색인 기법의 구조는 (그림 8)와 같다.

(그림 8)에서 A_{47}은 ECBH에서 RID와 유사하며 데이터 아이템 47이 저장되어 있는 주기억장치의 주소이다. $N_{47}(400)$은 A_{47}이 저장되어 있는 T*-노드의 주기억장치 주소가 400이라는 것을 나타낸다.

3.2 연산 방식

3.2.1 검색

간단한 검색 연산은 검색하고자 하는 데이터 아이템을 해시 함수로 변환시킨 후, 변환된 해시 값에 따라 해시테이블 검색을 수행한다. 검색하고자 하는 데이터 아이템의 주기억장치 주소가 해시테이블에 존재하는 경우에는 그 주소를 가지고 직접 접근할 수 있으며, 존재하지 않는 경우에는 그 데이터 아이템이 데이터베이스에 존재하지 않음을 나타낸다. 이와 같은 방법으로 간단한 검색 연산을 수행하면 빠른 검색이 가능하고, 검색 시간을 예상할 수 있다.

3.2.2 범위 검색

간단한 검색 연산은 해시 색인 기법만을 사용하지만 범위 검색 연산은 해시 색인 기법과 트리 색인 기법을 함께 사용한다. Hyper-TH 색인 기법에 사용된 T*-트리는 하나의 노드에 많은 엘리먼트를 포함하며 후임자 포인터를 가지고 간단한 순차검색을 수행하므로 범위 검색에 있어서 기존의 트리 색인 기법 중에 가장 성능이 우수하다. K_{min}과 K_{max} 사이의 데이터 아이템 범위 검색은 다음과 같은 순서로 수행한다.

(1) K_{min}을 해시 함수로 변환시킨 후, 변환된 해시 값 H_{min}<H_{min1} : H_{min2}>에 따라 해시 테이블 검색을 수행한다.

(2) K_{min}의 주기억장치 주소가 해시테이블에 존재하면 해시테이블의 연결리스트에서 K_{min}의 주기억장치 주소가 저장되어있는 T*-노드의 주소를 얻는다.

(3) K_{min}의 주기억장치 주소가 해시테이블에 존재하지 않으면 Hyper-TH 트리를 검색해

서 K_{min}과 가장 근접한 큰 데이터 아이템을 포함하는 T*-노드의 주소를 얻는다.

(4) 얻어진 T*-노드에서 K_{min}보다 크거나 같은 데이터 아이템 값의 위치를 찾은 다음, 그 위치부터 검색을 시작하여 K_{max}보다 크거나 같은 데이터 아이템 값을 만날 때까지 후임자 포인터(successor pointer)를 따라 순차 검색을 수행한다.

하나의 T*-노드는 정렬이 되어있고 그 노드에서 가장 큰 데이터 아이템 값을 쉽게 알 수 있으므로 한 노드에 대한 순차검색의 경우, 우선적으로 K_{max}와 그 노드에서 가장 큰 데이터 아이템 값을 비교해서 K_{max}가 더 크면 그 노드의 데이터 아이템 값들은 범위에 포함되는 것이므로 K_{max}와 비교 연산을 수행하지 않는다.

3.2.3 삽입

삽입 연산은 크게 트리 삽입과 해시 삽입으로 이루어진다. 트리 삽입은 삽입하고자 하는 데이터 아이템의 주기억장치 주소만을 Hyper-TH 트리에 삽입한다. 결과로서 삽입되어진 T*-노드의 주소를 얻을 수 있다. 해시 삽입은 삽입하고자 하는 데이터 아이템의 주기억장치 주소와 결과로서 얻은 T*-노드의 주소를 Hyper-TH 해시에 삽입한다. 데이터 아이템 K의 삽입은 다음과 같은 순서로 수행한다.

(1) K를 해시 함수로 변환시킨 후, 변환된 해시 값 $H<H_1 : H_2>$에 따라 해시 테이블 검색을 수행한다.

(2) 해시테이블에 존재하면 종료하고, 존재하지 않으면 Hyper-TH 트리에 K의 주기억장치 주소를 삽입한다. T*-트리 삽입 연산과 조건에 따라 특별 재균형 연산 또는 수정된 특별 재균형 연산이 Hyper-TH 트리 삽입에 적용이 된다. 삽입되는 T*-노드에 오버플로가 발생하면 K로 인해 하위 노드로 이동하게 되는 K'가 발생할 수 있다.

(3) 오버플로와 특별 재균형 연산이 발생하지 않은 삽입의 경우에는 K의 주기억장치 주소와 삽입된 T*-노드의 주소를 Hyper-TH 해시에 삽입한다.

오버플로가 발생하고 특별 재균형 연산이 발생하지 않은 삽입의 경우에는 Hyper-TH 해시에 K의 주기억장치 주소, K'의 주기억장치 주소와 각각 삽입된 T*-노드의 주소를 삽입하면 된다. Hyper-TH 해시에서 K는 새로운 공간을 할당받아 삽입하는 것이고 K'는 이미 삽입되어 있는 값을 변경하는 것이다.

특별 재균형 연산이 발생하는 삽입의 경우에는 조건에 따라 특별 재균형 연산 또는 수정된 특별 재균형 연산이 변경시킨 노드에 두 개의 키 K와 K'가 존재할 것이다. K는 Hyper-TH 해시에 삽입하고 K'는 Hyper-TH 해시에 이미 삽입되어 있는 T*-노드 주소를 변경한다.

(4) Hyper-TH 해시의 L_{avg}를 검사해서 1을 초과하면 그 해시 테이블의 local depth를 1만큼 증가시키고, 만일 local depth가 global depth보다 크면 디렉토리의 크기를 2배로 키우고 디렉토리의 엔트리들을 재 설정한다.

3.2.4 삭제

삭제 연산은 크게 트리 삭제와 해시 삭제로 이루어진다. 트리 삭제는 삭제하고자 하는 데이터 아이템이 저장되어있는 T*-노드의 주소를 얻기 위해 Hyper-TH 해시를 검색하고, 얻어진 T*-노드를 검색하여 그 데이터 아이템을 삭제한다. 해시 삭제는 삭제하고자 하는 데이터 아

이템의 주기억장치 주소를 Hyper-TH 해시에서 삭제한다. 데이터 아이템 K의 삭제는 다음과 같은 순서로 수행한다.

(1) K를 해시 함수로 변환시킨 후, 변환된 해시 값 $H<H_1 : H_2>$에 따라 해시 테이블 검색을 수행한다.
(2) 해시테이블에 존재하지 않으면 종료하고, 존재하면 해시테이블로부터 T*-노드의 주소를 얻어온다.
(3) 얻어온 T*-노드의 데이터 아이템들을 검색하여 K의 주기억장치 주소를 삭제한다. T*-트리 삭제 연산과 재균형 연산, 특별 재균형 연산 또는 수정된 특별 재균형 연산이 Hyper-TH 트리 삭제에 적용이 된다. 삭제 후에 노드들의 병합이나 제거가 발생하면 추가적인 데이터 아이템의 이동과 재균형 연산이 이루어지며 이동된 데이터 아이템의 T*-노드 주소를 변경해야 한다.
(4) 노드들의 병합이나 제거가 발생하지 않는 삭제의 경우에는 Hyper-TH 해시에서 K를 삭제한다.

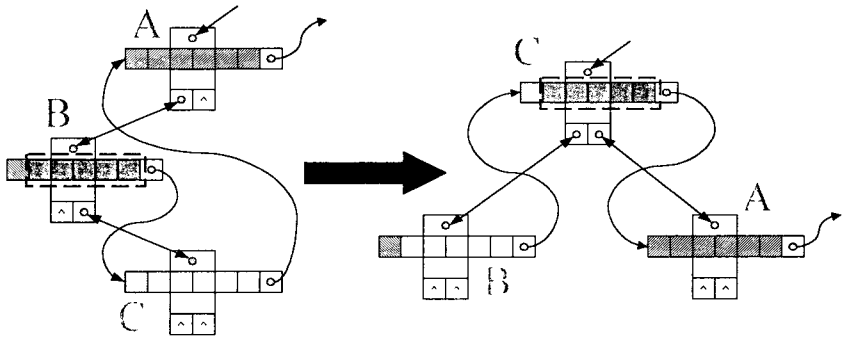
병합이나 제거가 발생하는 삭제의 경우에는 Hyper-TH 해시에서 K를 삭제하고, 추가적으로 발생한 데이터 아이템의 이동에 대해서 Hyper-TH 해시에 이미 삽입되어 있는 해당 데이터 아이템들의 T*-노드 주소를 변경한다.

(5) 삭제로 인해 그 해시 테이블이 공백이 되면, 그 해시 테이블의 H_2와 $(d-1)$비트가 같은 형제 해시 테이블과 병합하고 병합한 해시 테이블의 local depth를 1만큼 감소시킨다. 모든 해시 테이블의 local depth가 global depth보다 1만큼 작으면 디렉토리를 반으로 줄이고 디렉토리의 모든 엔트리들을 재 설정한다.

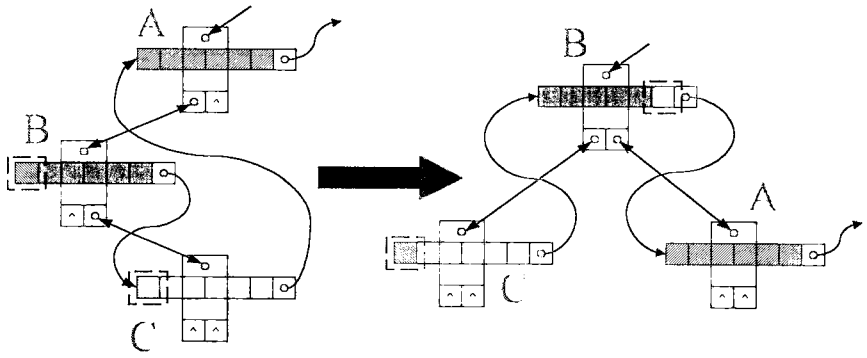
3.2.5 수정된 특별 재균형 연산

특별 재균형 연산은 T-트리에서 트리의 균형을 유지하게 위해 수행하는 재균형 연산의 특별한 경우로써 T*-트리에서 그대로 사용되었다. (그림 9)에서처럼 C 노드는 회전을 수행한 후에는 내부 노드가 된다. 내부 노드는 가득 채워져야 한다는 규칙에 따라 회전을 하기 전에 B 노드의 일부 데이터가 C 노드로 이동을 한다.

Hyper-TH에서는 Hyper-TH 트리의 데이터 이동에 따라 Hyper-TH 해시에 저장되어있는 Hyper-TH 트리 노드의 주소를 변경해야하기 때문에 Hyper-TH 트리의 데이터 이동이 증가함에 따라 Hyper-TH 해시의 변경도 증가하게 된다. 특히, 특별 재균형 연산은 Hyper-TH 트리에서 많은 데이터 이동을 요구하며 Hyper-



(그림 9) T*-트리의 특별 재균형 연산



(그림 10) 수정된 특별 재균형 연산

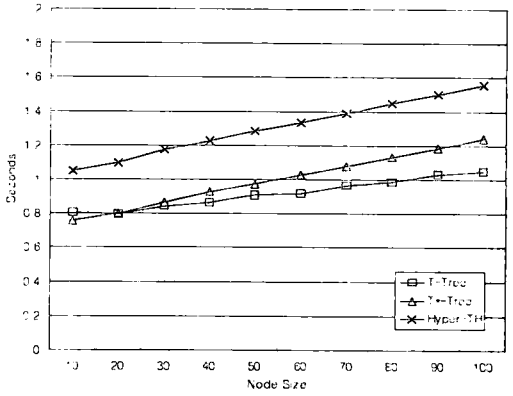
TH의 성능을 저하시키는 요인이 된다. 특별 재균형 연산을 보완하기 위한 방법으로써 수정된 특별 재균형 연산이 추가되었다. Hyper-TH에서는 데이터의 이동이 적은 조건에 따라 T*-트리의 특별 재균형 연산과 수정된 특별 재균형 연산을 사용하여 Hyper-TH 트리의 데이터 이동에 따른 Hyper-TH 해시의 변경을 최소화했다. 수정된 특별 재균형 연산은 (그림 10)처럼 회전 후에 C 노드를 내부 노드로 만들지 않고 B 노드를 내부 노드로 만드는 방법이다.

Hyper-TH의 특별 재균형 연산은 삽입에 따른 특별 재균형 연산과 삭제에 따른 특별 재균형 연산으로 구분할 수 있다. 삽입에 따른 특별 재균형 연산은 수정된 특별 재균형 연산에 따라 수행한다. 삭제에 따른 특별 재균형 연산은 C 노드에 있는 데이터의 양에 따라 T*-트리의 특별 재균형 연산과 수정된 특별 재균형 연산 중에 데이터의 이동이 적게 발생하는 것을 선택하여 수행한다.

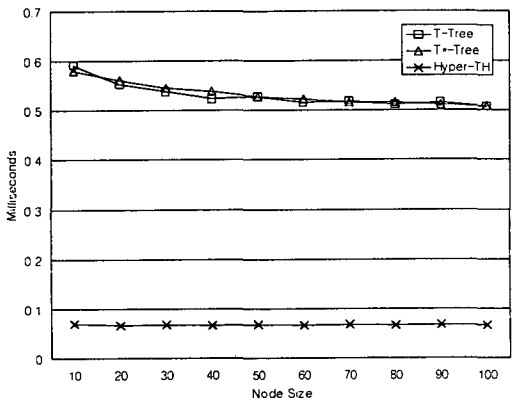
4. 성능 평가

4.1 구현 및 성능 평가 환경

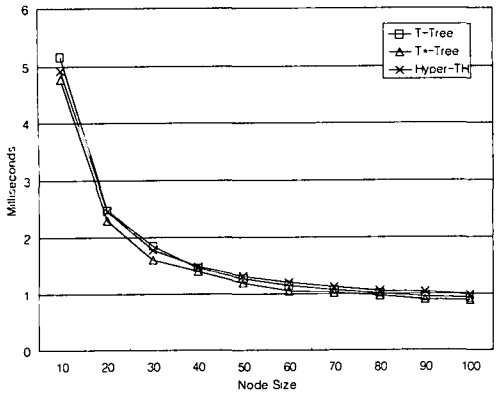
T-트리와 T*-트리, 그리고 Hyper-TH에 대한 검색, 범위 검색, 스캔, 삽입 연산을 주기억장치 형태로 구현하였으며 C 언어를 사용하였다. 플랫폼은 1 기가바이트의 주기억장치를 가지는 Sun Enterprise 250이며, OS는 Solaris 2.7이다. 컴파일러는 gcc 2.8을 사용하였다. 데이터는 난수 발생기를 사용하여 10,000보다 크고 1,000,000,000보다 작은 중복되지 않는 임의의 정수를 100,000개씩 20번 생성하고, 이 데이터로 트리 색인 구조의 노드 크기를 10부터 100까지 증가시키면서 삽입, 검색, 범위 검색, 스캔을 수행하였다. Hyper-TH의 해시테이블 크기는 1024로 했다. 범위 검색은 10,000개부터 70,000개까지 범위를 변화시키면서 테스트 했으며 40,000개의 범위 검색에 대한 성능만을 나타내었다. 테스트 결과는 (그림 11)~(그림 14)와 같다.



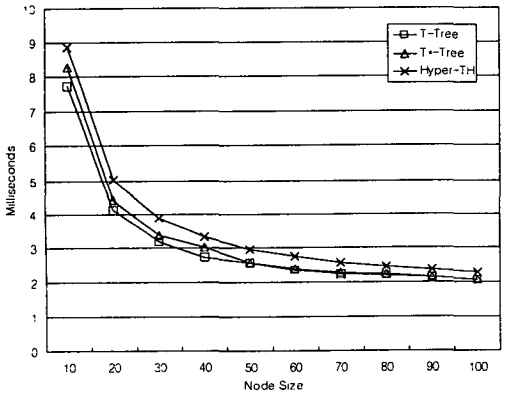
(그림 11) 삽 입



(그림 12) 검 색



(그림 13) 범위 검색



(그림 14) 스 캔

4.2 결과 분석

Hyper-TH의 성능평가 대상으로 T-트리와 T*-트리를 선택한 이유는 크게 두 가지로 볼 수 있다. 첫째는 색인 구조가 범위 검색과 스캔에 대한 연산을 지원할 수 있어야 하는 데 해시 색인 구조는 이러한 연산을 지원하기 어려우므로 비교 대상에서 제외되었다. 둘째는 다루고자 하는 데이터의 양을 미리 알 수 없는 상황에서 동적으로 색인을 구성할 수 있어야 하며 메모리 공간을 효율적으로 사용할 수 있어야 하는데 Hybrid-TH의 경우는 예로 들고 있는 해시 색인 구조가 정적 구조인 CBH이므로 비교 대상에서 제외되었다.

4.2.1 삽입

Hyper-TH의 삽입 연산 시간은 T-트리와 T*-트리의 삽입 연산 시간보다 해시 삽입 시간만큼, T*-트리의 삽입 연산 시간은 T-트리의 삽입 연산 시간보다 후임자 노드에 대한 참조 포인터를 기록하는 시간만큼 증가한다. 결과적으로, Hyper-TH의 삽입 연산 시간 차이는 T-트리와 T*-트리가 가지는 시간 차이와 트리의 노드 주소를 저장하면서 발생하는 추가적인 연산이다.

기존의 Hybrid-TH 삽입 연산의 경우에 해시 테이블의 크기를 상당히 크게 설정하면 Hyper-TH와 비슷한 결과를 가지지만 CBH가 동적 파일에 대한 지원을 하지 못한다는 것 때문에 비교 대상에서 제외가 되었다. Hybrid-TH의 경우에 CBH의 해시테이블 크기를 상당히 크게 설정하면 저장 공간 낭비가 심하고, 작게 설정하면 연결 리스트 길이의 증가에 따른 삽입과 검색 시간 증가와 수행 시간을 예상하는 것이 힘들어진다.

4.2.2 검색

Hyper-TH의 검색 연산 시간은 해시만을 사용하므로 T-트리와 T*-트리의 검색 연산 시간보다 훨씬 빠르다. 또한 해시테이블에서 연결 리스트의 길이를 일정 크기가 넘지 않도록 설정해서 검색 연산 시간을 예상할 수 있으므로 실

시간 특성을 보장하는 것이 가능하다.

4.2.3 범위 검색

범위 검색 연산은 범위 검색 시작 데이터 아이템을 검색하는 부분과 범위 검색 종료 데이터 아이템을 검색하는 부분이 다를 뿐 스캔 연산과 비슷하다. 따라서 스캔 연산과 비슷한 성능을 나타낸다. 하지만 중위 순회를 사용하는 경우, 비교 연산이 증가하므로 후임자 노드 참조 포인터를 사용하는 것이 다소 좋은 성능을 보인다. Hyper-TH는 범위 검색 연산에서 범위 검색 시작 데이터 아이템이 해시에 존재하는 경우에 가장 좋은 성능을 나타낸다. 그 이유는 범위 검색 시작 데이터 아이템을 해시에서 찾으면 트리에서 범위 검색 시작 데이터 아이템을 찾지 않아도 되며, 그 데이터 아이템이 저장되어 있는 T*-노드의 주소를 가지고 스캔 연산을 수행할 수 있다. 또한 이 경우에 수행 시간을 예상하는 것이 가능해진다.

4.2.4 스캔

Hyper-TH와 T*-트리는 후임자 노드 참조 포인터를 따라 스캔한다. 이론상으로 후임자 노드 참조 포인터를 따라 스캔하는 것이 중위 순회를 통해 스캔하는 것보다 훨씬 빠를 것으로 예상되지만 실제적으로는 비슷한 비교 연산 횟수로 인해 예상과는 달리 약간 빠른 성능을 보인다.

5. 결 론

본 논문에서는 실시간 주기억장치 상주형 데이터베이스 시스템을 위한 효율적인 색인 기법을 제안하였으며, 구현 및 성능평가를 통해 기존의 색인 구조들과 비교함으로써 실시간 특성 보장 및 성능 향상을 증명하였다. Hyper-TH는 T*-트리의 주기억장치 상주형 트리 색인 구조 특성과 향상된 범위 질의 처리 특성, 그리고 ECBH의 결정론적인 검색시간 특성과 동적 확장 특성을 적절히 결합하여 시기 적절함과 예상 가능성과 같은 실시간 특성을 만족시키면서 향상된 성능을 보장할 수 있다.

향후, 실시간 색인 기법에 대한 동시성 제어를 연구할 계획이며, 제안된 색인 기법을 적용시킨 실시간 주기억장치 데이터베이스 시스템을 개발해서 테스트를 통해 얻어진 문제점을 보완하고 성능 향상 및 안정화에 힘쓰고자 한다.

참 고 문 헌

[1] Kim, Y.K., S.H. Son, "Predictability and Consistency in Real-Time Database Systems," in Advances in Real-Time Systems(S. H. Son, ed.), chap. 21, pp.509-531, Prentice Hall, 1995.

[2] Garcia-Molina, H., K. Salem, "Main Memory Database Systems : An Overview," IEEE Transactions on Knowledge and Data Engineering, Vol.4, No.6, Dec. 1992, pp.509-516.

[3] Tobin J. Lehman, Michael J. Carey, "A Study of Index Structures for Main Memory Database Management Systems," in Proceedings 12th Int. Conf. On Very Large Databases, Kyoto, Aug. 1986, pp.294-303

[4] Tobin J. Lehman, Michael J. Carey, "Query Processing in Main Memory Database Management Systems," Proc. ACM SIGMOD Conf., May. 1986, pp.239-250

[5] Aho, A., J. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Company, 1974.

[6] Perlis, A., Thornton, C., Symbol Manipulation by Threaded Lists, CACM, Vol.3, No.4, April 1960, pp.195-204

[7] Comer, D., "The Ubiquitous B-tree," Computing surveys 11, 2 (June 1979)

[8] Choi, K.R., K.C. Kim, "T*-tree : A Main Memory Database Index Structure for Real Time Applications," Proceeding of RTCSA '96 IEEE, 1996.

[9] Fagin, R., Nievergelt, J., Pippenger, N., Strong, H., Extendible Hashing A Fast Access Method for Dynamic Files. ACM Trans. On Database Systems, Vol.4, No.3, pp.315-344, 1979

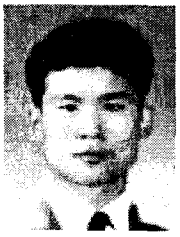
[10] Kim, P.C., K.U. Lim, J.P. Hong, "Extendible Chained Bucket Hashing for Main Memory Databases," International Journal of Applied Software Technology, Vol. 1, No.2, 1995, pp.123-135.

[11] Ryu, C.H., E.M. Song, B.S. Jun, Y.K. Kim, S.I. Jin, "Hybrid-TH : a Hybrid Access Mechanism for Real-Time Memory-Resident Database Systems," Proceeding of RTCSA '98 IEEE, 1998.

저자소개

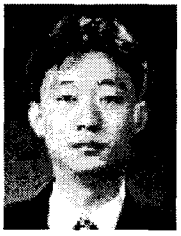
민 영 수

충북대학교 정보통신공학과에서 공학사, 충북대학교 대학원 정보통신공학과에서 공학석사학위를 취득하였으며, 현재 충북대학교 정보통신공학과에서 박사과정에 있다. 주요관심분야는 데이터베이스 시스템, 실시간 시스템, XML 등이다.



신 재 룡

충북대학교 정보통신공학과에서 공학사, 충북대학교 정보통신공학과에서 공학석사학위를 취득하였고, 박사과정을 수료하였다. 주요관심분야는 데이터베이스 시스템, 실시간 시스템, 멀티미디어 데이터베이스 등이다.



이 병 엽

한국과학기술원 전산학과에서 공학사, 공학석사학위를 취득하였고, 한국과학기술원 경영정보학과에서 공학박사학위를 취득하였다. 현재 대우정보시스템 e-솔루션사업팀 차장으로 재직중이며, 주요관심분야는 데이터마이닝, XML, 멀티미디어 정보처리 등이다.

유 재 수

전북대학교 공과대학 컴퓨터공학과에서 공학사, 한국과학기술원 전산학과에서 공학석사·박사학위를 취득하였으며, 목포대학교 전산통계학과 전임강사를 역임하였고, 현재 충북대학교 전기전자 및 컴퓨터공학부 부교수로 재직중이다. 주요관심분야는 데이터베이스 시스템, 정보검색, 멀티미디어 데이터베이스, 분산 객체 컴퓨팅 등이다.