2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

실시간 MMDBMS를 위한 효율적인 색인 기법의 설계 및 성능평가

민영수, 신재룡, 유재수
충북대학교 정보통신공학과
E-mail : {minys, jrshin, yjs}@pretty.chungbuk.ac.kr

Design and Performance Evaluation of an Efficient Index Mechanism for Real-Time MMDBMS

Young Soo Min, Jae Ryong Shin, Jae Soo Yoo
Dept. of Computer & Communication Eng., Chungbuk National University

요 약

본 논문에서는 실시간 MMDBMS 를 위한 효율적인 색인 기법을 제안한다. 기존의 주기억장치 트리 기반 색인 구조는 범위 검색을 효과적으로 지원할 수 있지만 한 노드에 대한 접근 시간과 평균 접근 시간의 차이가 클 수 있기 때문에 실시간 특성을 보장하지 못하는 단점이 있다. 또한 해시 기반 색인 구조는 간단한 검색에서 접근 시간이 매우 빠르고 일정하지만 범위 검색을 지원하지 못하는 단점이 있다. 이러한 두 색인 구조의 단점을 해결하기 위해 본 논문에서는 동적 확장이 가능하며 검색 시간이 빠르고 실시간 특성을 지원할 수 있는 ECBH(Extendible Chained Bucket Hashing)와 범위 검색에 더욱 효과적인 T*-트리를 상호보완적으로 결합하여 Hyper-TH(Hyper Tree-Hash)라는 실시간 MMDBMS 에 적합한 새로운 색인 기법을 제안하고 구현한다. 그리고 성능 평가를 통해 제안하는 색인 기법의 우수성을 증명한다.

1. 서론

핵발전소 제어, 공정 제어, 항공기 제어와 같은 실시간 응용들에 사용되는 실시간 시스템은 계산의 논리적 정확성만을 다루는 기존의 시스템과는 달리 실시간 제약 조건인 종료시한(deadline)을 만족시켜야 한다. 실시간 시스템은 실시간 응용들을 만족시키기 위해 시간 요구 조건을 만족시키면서 종료시한 내에 결과를 만들어 내는 시기 적절함(timeliness), 시스템 명세에 정의된 고장이나 작업 부하 조건에서 작업의 종료시한을 보장하는 예상가능성(predictability)과 같은 실시간 특성과 빠른 수행 능력을 가져야 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.

*본 연구는 과학재단 특정기초과제(과제번호: 1999-1-303-007-3) 연구비지원에 의하여 수행되었음.

최근 실시간 응용들은 더욱 복잡해지고 방대한 양의 데이터 접근을 요구하고 있으며, 데이터에 대한 조직적 관리를 필요로 하고 있다. 데이터베이스관리시스템은 이러한 데이터의 조직적 관리 수단을 제공할 수 있기 때문에 실시간 시스템과 자연스러운 통합이 이루어졌다. 이렇게 통합된 실시간 데이터베이스관리시스템은 다양한 응용들을 위해 실시간 제약조건을 포함한 데이터베이스 연산들을 제공한다. 하지만 기존의 디스크 기반 데이터베이스관리 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 데이터베이스관리시스템(MMDMS)이 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 데

이터를 조직적으로 관리하고 실시간 특성에 맞는 데이터베이스 연산을 지원하는 MMDMS 의 필요성이 높아졌다[2].

MMDBMS 에서 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 CPU 를 효과적으로 사용하면서 전체 계산 시간을 줄이는 것에 중점을 두고 있다. 또한 주기억장치 기반의 색인 구조는 색인되는 데이터 자체를 저장하지 않고 그 데이터가 저장되어 있는 위치 정보를 저장하기 때문에 가변길이 데이터에 대한 저장 문제를 해결할 수 있고 저장 공간을 효율적으로 사용할 수 있는 장점을 갖는다[3].

실시간 주기억장치 기반 색인 구조는 최악의 수행시간이 항상 특정 시간 내에 놓여질 수 있도록 예상 가능성을 보장해야 하며 일반적인 데이터베이스 연산 처리가 가능해야 한다. 본 논문에서는 실시간 특성을 보장하기 위한 해시 기반 색인 구조와 범위 검색을 효과적으로 지원하기 위한 트리 기반 색인 구조를 상호보완적으로 결합시킴으로써 실시간 MMDBMS 을 위한 효율적인 색인 기법을 제시하고 성능 평가를 통해 기존의 다른 색인 구조보다 성능이 우수함을 증명하고자 한다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구로써 기존의 주기억장치 색인 구조들의 특성 및 문제점을 제시한다. 3 장에서는 제안하는 색인 기법에 대한 기본 구조를 설명한다. 4 장에서는 검색, 범위 검색, 삽입, 삭제 알고리즘을 설명한다. 5 장에서는 제안하는 색인 기법의 성능을 기존 색인 구조인 T-트리, T*-트리와 비교하고 분석한다. 마지막 6 장에서는 결론을 맺고 향후 연구방향을 제시한다.

2. 관련 연구

2.1. T-트리

T. J. Lehman 과 M. J. Carey 에 의해 제안된 T-트리는 AVL-트리와 B-트리를 발전시킨 데이터 구조이다[3]. T-트리는 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 하위 노드로 이동시켜서 회전 후 T-트리의 내부 노드가 가득 차도록 만든다.

2.2. T*-트리

K. R. Choi 와 K. C. Kim 은 중위 순회를 사용하여 범위 검색을 수행하는 T-트리의 성능을 향상시키기 위해 연결 리스트를 통한 순차 검색을 수행하는 T*-트리를 제안하였다[5]. T*-트리는 T-트리와 거의 동일하다고 볼 수 있지만 범위 질의나 순차 검색과 같은 질의에 대해 트리를 순회하기 보다는 간단한 연결 리스트를 통해 순차 검색이 가능하도록 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 재삽입하는 반면에, T*-트리는 한 노드의 가장 큰 값을 하위 노드에 재삽입 한다.

2.3. ECBH

ECBH 는 주기억장치 데이터베이스의 동적 파일을 위해 CBH 의 빠른 접근 성능과 EH 의 점진적인 확장성을 적절히 결합한 방법이다[6].

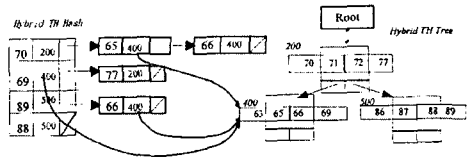
임의의 키 K 의 해시 값 $H<H1 : H2>$는 해시테이블의 엔트리를 할당하기 위해 사용하는 $H1$ 과 디렉토리의 엔트리를 할당하기 위해 사용하는 $H2$ 로 구성된다. 이때 $H1$ 을 위해 확보되는 비트의 수는 $\lceil \log C \rceil$이다. 그리고 삽입되는 키의 수가 m 이고 선행하는 RID_i 의 체인 수가 L_{RIDi} 일 때 평균 체인 길이 L_{avg} 는 수식 1 과 같다.

$$L_{avg} = \frac{1}{m}\sum_{i=1}^{m} L_{RID_i} \qquad \text{.....(수식 1)}$$

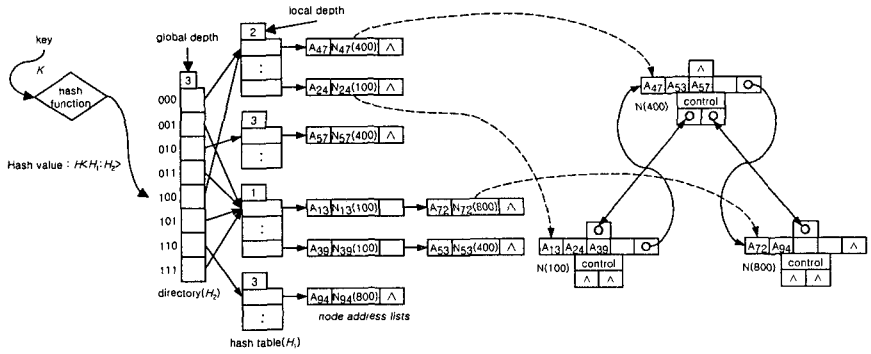
Global depth 가 g, local depth 가 d 일 때 최대 global depth 인 g_{max} 는 h-logC 이다. 이때 h 는 H 의 비트 수를 나타낸다. 또한 ECBH 의 평균 체인 길이를 제어하기 위한 변수는 l 이다. 평균 체인 길이가 l 보다 커지면 분할이 수행된다.

2.4. Hybrid-TH(Hybrid Tree-Hash)

Hybrid-TH 는 실시간 MMDBMS 에서 예상 가능성을 보장하기 위해 해시 기반 색인 구조인 CBH 와 트리 기반 색인 구조인 T-트리를 상호보완적으로 결합한 색인 기법이다[4]. Hybrid-TH 는 CBH 의 빠른 검색 성능과 예상 가능한 접근시간, T-트리의 범위 검색을 지원하지만, CBH 가 해시테이블 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.



(그림 1) Hybrid-TH 의 구조



(그림 2) Hyper-TH 의 구조

3. 제안하는 실시간 색인 기법

제안하는 Hyper-TH 는 실시간 MMDMS 에서 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 는 기존의 T*-트리 색인 구조와 동일하고, Hyper-TH 해시는 기존의 ECBH 색인 구조와 유사하지만 데이터 아이템의 주기억장치 주소만 저장되어 있던 연결 리스트에 데이터 아이템의 주기억장치 주소가 저장되어 있는 Hyper-TH 트리 노드 주소를 추가적으로 저장하는 것이 다르다.

그림 2 는 Hyper-TH 구조의 한 예를 나타낸다. A_{47} 은 ECBH 에서 RID 와 동일하며 데이터 아이템 47 이 저장되어 있는 주기억장치의 주소이다. 그리고 $N_{47}(400)$은 A_{47} 이 저장되어 있는 T*-노드의 주기억장치 주소가 400 이라는 것을 나타낸다.

4. Hyper-TH 의 연산 알고리즘

4.1. 검색연산

간단한 검색연산은 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 검색연산을 수행하면 빠른 검색이 가능하고, 검색 시간을 예상할 수 있다.

4.2. 범위 검색연산

K_{min} 과 K_{max} 사이의 데이터 아이템 범위 검색은 다음과 같은 순서로 수행한다.

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

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

(3) K_{min} 의 주소가 해시테이블에 존재하지 않으면 Hyper-TH 트리를 검색해서 K_{min} 과 가장 근접한 큰 데이터 아이템을 포함하는 T*-노드의 주소를 얻는다.

(4) 얻어진 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 포인터를 따라 순차검색을 수행한다.

4.3. 삽입연산

삽입연산은 크게 트리 삽입과 해시 삽입으로 이루어진다. 트리 삽입은 삽입하고자 하는 데이터 아이템의 주기억장치 주소만을 Hyper-TH 트리에 삽입한다. 결과로서 삽입되어진 T*-노드의 주소를 데이터 아이템의 주기억장치 주소와 함께 해시에 삽입한다. 데이터 아이템 K 의 삽입은 다음과 같은 순서로 수행한다.

(1) K 를 해시 함수로 변환시킨 후, 변환된 해시 값 H<$H1$: $H2$>에 따라 해시테이블 검색을 수행한다.

(2) 해시테이블에 존재하면 종료하고, 그렇지 않으면 Hyper-TH 트리에 K 의 주소를 삽입한다.

(3) 오버플로우와 특별 재균형 연산이 발생하지 않는 삽입의 경우에는 K 의 주기억장치 주소와 삽입된 T*-노드의 주소를 Hyper-TH 해시에 삽입한다.

(4) 오버플로우가 발생하고 특별 재균형 연산이 발생하지 않는 삽입의 경우에는 Hyper-TH 해시에 K 의 주기억장치 주소, K'의 주기억장치 주소와 각각 삽입된 T*-노드의 주소를 삽입한다.

(5) Hyper-TH 해시의 L_{avg} 를 검사해서 l 을 초과하면 그 해시테이블의 local depth 를 1 만큼 증가시키고, local depth 가 global depth 보다 크면 디렉토리의 크기를 2 배로 키우고 디렉토리의 엔트리들을 재설정한다.

4.4. 삭제연산

삭제연산은 크게 트리 삭제와 해시 삭제로 이루어진다. 트리 삭제는 삭제하고자 하는 데이터 아이템이 저장되어 있는 T*-노드의 주소를 얻기 위해 Hyper-TH 해시를 검색하고, 얻어진 T*-노드를 검색하여 그 데이터 아이템을 삭제한다. 해시 삭제는 삭제하고자 하는 데이터 아이템의 주기억장치 주소를 Hyper-TH 해시에서 삭제한다. 데이터 아이템 K 의 삭제는 다음과 같은 순서로 수행한다.

(1) K 를 해시 함수로 변환시킨 후, 변환된 해시 값 $H<H1 : H2>$에 따라 해시테이블 검색을 수행한다.

(2) 해시테이블에 존재하지 않으면 종료하고, 존재하면 해시테이블로부터 T*-노드의 주소를 얻어온다.

(3) 얻어온 T*-노드의 데이터 아이템들을 검색하여 K 의 주기억장치 주소를 삭제한다.

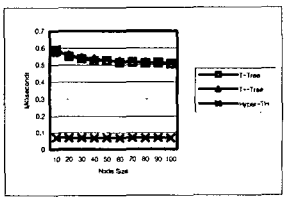
(4) 노드들의 병합이나 제거가 발생하지 않는 삭제의 경우에는 Hyper-TH 해시에서 K 를 삭제한다. 병합이나 제거가 발생하는 삭제의 경우에는 Hyper-TH 해시에서 K 를 삭제하고, 추가적으로 발생한 데이터 아이템의 이동에 대해서 Hyper-TH 해시에 이미 삽입되어 있는 해당 데이터 아이템들의 T*-노드 주소를 변경한다.

(5) 삭제로 인해 그 해시테이블이 공백이 되면 ECBH 의 방법과 동일하게 병합과 재설정을 해준다.

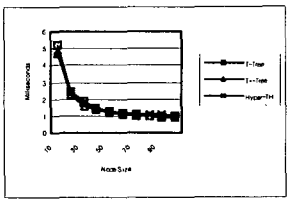
5. 성능 평가

T-트리, T*-트리 그리고 Hyper-TH 에 대한 알고리즘을 주기억장치 형태, 즉 색인에 키 값이 저장되어 있는 주기억장치 상의 주소를 저장하는 형태로 구현하였으며 C 언어를 사용하였다. 128Mbyte 의 주기억장치와 Intel Pentium-II 350 ㎒ CPU, OS 는 linux kernel 2.2.14 를 사용한 PC 에서 단일 사용자 모드로 테스트했다. 데이터는 난수 발생기를 사용하여 임의의 정수를 100,000 개 생성하고, 이 데이터로 트리 색인 구조의 노드 크기를 10 부터 100 까지 증가시키면서 각각 100 번씩 삽입, 검색, 범위 검색, 스캔을 수행하고 평균을 구했다. Hyper-TH 의 해시테이블 크기는 1024 로 했다.

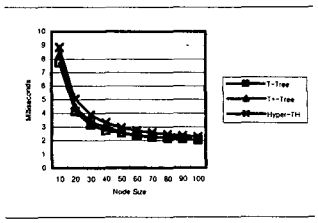
성능평가 결과 단순 검색은 해시의 사용으로 월등한 성능향상을 보이고 있으며 범위 검색과 스캔은 약간의 성능향상을 보였다 그러나 삽입은 해시와 트리에 삽입이 이루어지므로 기존의 방법보다 성능이 다소 떨어지지만 다중 프로세서 시스템을 적용하여 해결할 수 있다.



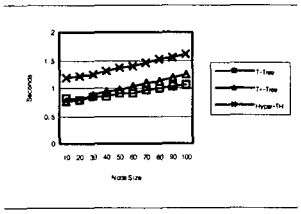
(그림 3) 검색



(그림 4) 범위 검색



(그림 5) 스캔



(그림 6) 삽입

6. 결론 및 향후 연구방향

본 논문에서는 주기억장치 색인 구조들에 대한 고찰을 통해 실시간 MMDBMS 을 위한 효율적인 색인 기법을 제안하였으며, 구현 및 성능평가를 통해 기존의 색인 구조들과 비교함으로써 실시간 특성 보장 및 성능 향상을 증명하였다.

Hyper-TH 는 T*-트리의 주기억장치 트리 색인 구조 특성과 향상된 범위 질의 처리 특성, 그리고 ECBH 의 결정론적인 검색시간 특성과 동적 확장 특성을 적절히 결합하여 시기 적절함, 예상 가능성과 같은 실시간 특성을 만족시키면서 향상된 성능을 보장할 수 있다. 향후, 삭제 연산의 구현과 동시성 제어 방법을 연구할 계획이다.

참고문헌

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

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

[6] P. C. Kim, K. U. Lim and J. P. Hong, "Extendible Chained Bucket Hashing for Main Memory Databases", International Journal of Applied Software Technology, Vol. 1, No. 2, 195, pp.123-135.