동적 주기억 데이터베이스를 위한 색인 구조의 성능 평가*

박정규 전흥석 노삼혁
홍익대학교 컴퓨터공학과

Performance Evaluation of an Index Structure for Dynamic Main Memory Database

Park Jung Kyu H. Seok Jeon Sam H. Noh
Dept. of Computer Engineering, Hong-Ik University

요 약

주기억 데이터베이스에서 효율적인 성능을 위해서 제안된 색인 구조 중 T-트리가 있다. 이 색인 구조는 삽입 삭제가 많은 동적 주기억 데이터베이스에서 빈번한 노드 생성 및 삭제에 따르는 오버헤드(overhead)로 효율적이지 못하다. 이 문제를 극복하기 위해서 T^2-트리가 제안되었다. T^2-트리는 T-트리의 단점인 범위 질의의 비효율성의 해결과 삽입 삭제가 빈번한 동적 주기억 데이터베이스 시스템을 위해 억제된 노드 생성 및 삭제 기법과 스레드 이진 트리의 특징을 가지고 있다. 이 논문에서는 리눅스에서 주기억 데이터베이스 프로그램인 FastDB에 사용된 T-트리 인덱싱 구조를 새롭게 제안된 T^2-트리로 수정하여 두 가지 인덱싱 구조를 비교 실험한 결과를 보여주고 있다. 실험결과에 의하면 T-트리에 비해서 T^2-트리가 동적인 주기억 데이터베이스 시스템에서 효율적인 구조임을 알 수 있다.

1. 서론

주기억 데이터베이스 시스템은 주기억장치에 데이터베이스 시스템 전체를 상주시켜 데이터베이스 연산을 수행하는 시스템을 말한다. 기존에 디스크 기반의 데이터베이스에서는 디스크 접근 횟수와 디스크 공간을 줄이는 것이 목적이었지만, 주기억 데이터베이스는 중앙처리 장치의 계산횟수와 주기억장치에 차지하는 공간을 줄이는 것이 목적이다. 최근까지 연구된 색인구조들 중에 대부분은 디스크 기반 데이터베이스를 위한 색인구조들 이었다. 이에 반해 주기억 데이터 데이터베이스를 위한 색인 구조 중에는 T-트리[1]와 이의 단점을 개선한 T^2-트리[2]가 있다.

이 논문에서는 리눅스에서 구현된 FastDB [3] 주기억 데이터베이스 프로그램의 T-트리를 주기억 데이터베이스의 색인 구조로 새롭게 제안된 T^2-트리로 수정하여 기존의 T-트리와 비교 실험하였다. 실험 결과 T^2-트리가 동적 주기억 데이터베이스에서 효율적 보여줌을 실험 결과를 통하여 보여준다.

본 논문의 구성은 다음과 같다. 제 2 절에서는 기존의 주기억 데이터베이스 색인기법인 T-트리와 T^2-트리에 대해 살펴본다. 제 3 절에서는 실제 주기억 데이터베이스 프로그램인 FastDB에 대한 내용과 T^2-트리 구현 내역에 대해 설명하며, 제 4 절에서 T^2-트리의 성능 평가를 하고 마지막으로 제 5 절에서 결론을 맺는다.

2. Index Structure for Main Memory Database

2.1 T-트리

T-트리는 AVL 트리와 B-트리의 특성을 결합해서 새롭게 변형된 트리 구조로서 Lehman이 주기억 데이터베이스의 효율적 운용을 위하여 제안하였다. T-트리는 한 노드 안의 여러 개의 데이터 아이템들을 가지는 B-트리[4]의 특성과 이진 검색의 AVL 트리[5] 특성을 결합하여 만든 색인 구조이다. T-트리의 구조는 그림 1과 같다.

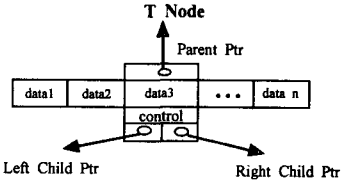

그림 1. T-트리의 노드 구조

T-트리는 3가지의 단점을 가진다. 첫 번째, 노드간 순회를 중위 순회로 하므로 범위 질의 수행 시 불필요한 순회를 하게 된다. 두 번째, 중간 노드에서 데이터 아이템의 삽입과 삭제 시 점유 범위 만족을 위해서 Greatest Lower Bound (GLB) 접근이 발생한다. 이 때 불필요한 중간 노드까지 반드시 거쳐야만 접근 가능하고, GLB 노드만을 살펴보기 때문에 새로운 노드 생성 또는 삭제 가능성이 높다. 세 번째, 노드 생성과 삭제 시 트리의 불균형 상태를 알아보기 위해서 항상 트리 깊이를 계산해 주어야 하고, 불균형 상태가 되면 회전 연산을 수행해야 한다는 단점이 있다.

2.2 T^2-트리

앞서 소개한 T-트리의 단점을 보완하기 위해서 만들어진 T^2-트리는 억제된 노드 생성 및 삭제와 단말 노드와 단말 노드가 하나의 자식 노드를 가지는 반 단말 노드의 스레드 포인터를 이용한 새로운 색인 구조이다. T^2-트리의 구조는 그림 2와 같다.

*본 연구는 한국 과학 재단 특정기초연구과제(과제번호:98-0102-09-01-3)의 지원을 받았음.

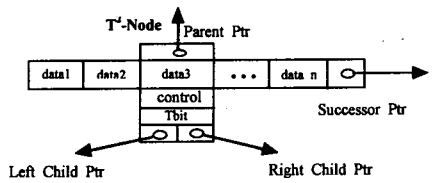

그림 2. T^2-트리의 노드 구조

T^2-트리는 후위 포인터[6]와 스레드 이진 트리[7]의 특성을 결합한 색인 구조이다. T^2-트리는 왼쪽 자식포인터에 대하여 스레드 포인터를 적용하고 있다. 컨트롤 박스에 있는 Tbit는 왼쪽 자식 포인터의 스레드 여부를 구별하기 위해 사용되는 비트이다. Tbit 가 0 값을 가지면 실제 왼쪽 자식을 가리키는 포인터이며, 1 값을 가지면 스레드 포인터가 되어 전위 포인터(predecessor pointer)가 된다.

후위 포인터는 범위 질의 외에도 데이터 아이템의 삽입과 삭제 시에도 사용이 된다. 이러한 접근방식의 효과는 T-트리에 비해서 데이터 아이템의 삽입과 삭제 시 발생되는 노드 생성과 삭제를 억제시킬 수 있으며 자연스럽게 트리 균형을 맞추기 위한 오버헤드도 줄어들게 된다. 또한 오버플로(overflow)를 해결하기 위해서 GLB 노드와 Least Upper Bound(LUB)노드 양쪽을 살펴보기 때문에 트리 공간 활용도도 높아져 검색 시에도 성능을 높일 수 있다.

3. 구현

3.1 FastDB

FastDB는 국내 리눅스 배포본에 패키지 형태로 들어 있는 주기억 데이터베이스 시스템으로서, 현재 널리 알려진 주기억 데이터베이스 시스템이다. 현재 2.11 버전까지 나와 있고 Unix, Linux, Windows NT 환경에서 동작하는 버전이 무료로 제공된다.

FastDB의 T-트리는 Lehman이 제안한 T-트리의 개념에 따라 작성되었지만 차이점이 존재한다. FastDB에서 제안한 T-트리에서 차이점은 다음과 같다. Lehman의 논문에서는 T-트리 내 노드에 실제 데이터가 삽입이 되어 있으나 FastDB에 T-트리는 실제 데이터가 아닌 메모리 내에 순서대로 저장되어 있는 데이터 레코드의 번호를 가지고 있다. 이 데이터 레코드에 접근하기 위해서는 데이터 레코드의 번호를 FastDB에서 제공하는 함수를 사용하여 실제 주소로 변환하여 데이터에 접근한다.

3.2 T^2-tree의 구현

그림 3은 FastDB에서 인덱싱 구조로 사용되고 있는 T-트리를 T^2-트리로 수정한 그림 3이다. FastDB에서 T-트리를 수정하기 위해 우선 기존에 제공되는 소스의 자료 구조부터 수정을 하였다. T^2-트리에 Parent 포인터, Successor 포인터 변수와 정수형 Tbit 변수를 dbTreeNode 클래스에 추가하였다. 위의 추가된 변수를 이용하여 T-트리를 T^2-트리로 수정하기 위해 변경된 함수들은 다음과 같다.

데이터를 검색하는 함수 find()는 기존의 T-트리와 크게 변경된 부분은 없다. 범위 검색을 위해 함수를 중복정의 하여 범위 검색시 초기 값과 마지막 값을 입력할 수 있도록 하였다. 노드를 새로 할당하는 함수 allocate()에는 인자로 Successor 포인터 변수를 추가하였다. 또한 T^2-트리는 T-트리와 다르게 노드 내 데이터를 노드 뒤에서부터 추가 할 수 있도록 하였다. 데이터를 삽입하는 함수 insert()에서는 노드에 삽입시 새로운 노드를 생성할 때 Tbit 값을 초기화 시켜주도록 하였고 스레드 포인터의 연결 코드를 추가하였다.

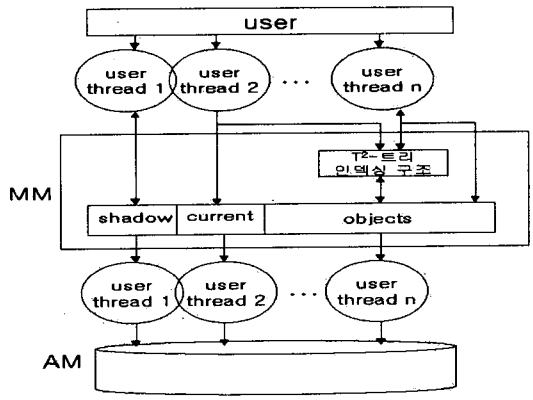

그림 3. 수정한 FastDB 구조도

또한 데이터 삽입시 오버플로(overflow)가 발생하면 새로운 노드를 추가하기 위해서 T-트리에서는 GLB 노드만 확인 하지만 T^2-트리에서는 LUB 노드도 확인하여 노드를 생성할 수 있도록 코드를 추가하였다. 데이터를 삭제하는 함수 delete()에서는 삽입의 경우와 유사하게 노드의 삭제시 스레드 포인터 연결 코드를 추가하였다. 데이터의 삭제시 언더플로가 발생하면 T-트리는 GLB노드만을 확인하여 처리하지만 T^2-트리에서는 LUB 노드도 확인하여 노드가 삭제될 수 있도록 코드를 추가하였다.

4. 실험 및 성능 평가

주기억 데이터베이스 색인 구조인 T-트리와 T^2-트리를 성능평가하기 위한 실험 환경은 표 1과 같다.

표 1. 실험 환경

CPU	Pentium III 550 MHz
메모리	128MB
기억장치	6.4GB
운영체제	LINUX 6.1

실험에 사용한 데이터 파일은 실험전 데이터 파일 생성 프로그램을 통하여 순차파일과 랜덤수발생기를 파일로 만들었다. 실험은 삽입 삭제 검색으로 크게 나눌 수 있으며 자세한 내용은 다음과 같고 실험 결과 그림 4 와 같다.

삽입 (순차파일)

삽입 실험은 데이터 파일 생성프로그램으로 만든 이진 순차파일 파일을 사용하여 실험을 하였다. 순차파일을 이용하여 100,000개의 테이터를 삽입한 결과는 T-트리에 비해서 T^2-트리가 시간이 더 많이 걸렸다. 그 내용으로 데이터 삽입시 노드 내에서 데이터 이동에 따른 시간이 T-트리가 T^2-트리에 비해서 좋은 결과를 나타냈다. 또한 노드 할당 및 회전 시간도 T-트리가 T^2-트리에 비해 3.5% 빠른 결과를 나타냈다.

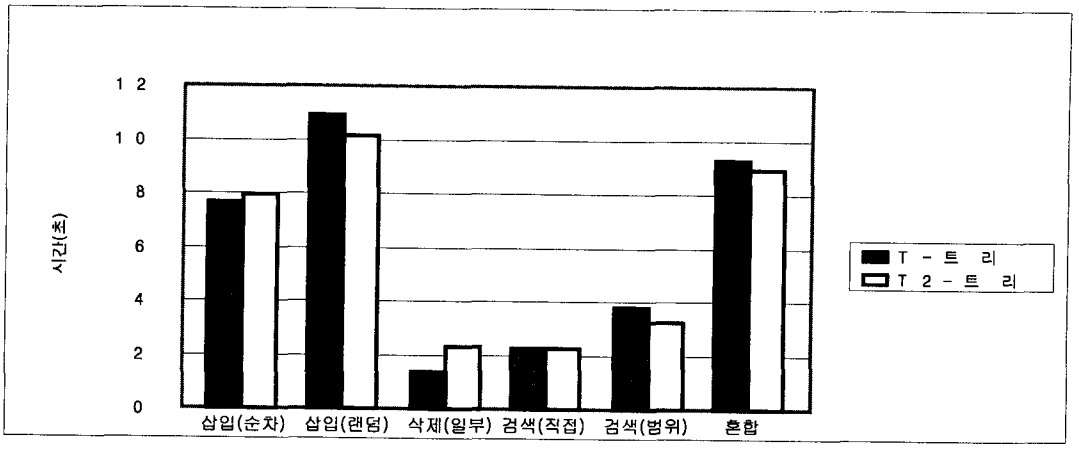

그림 4. T-트리와 T^2-트리 비교 실험 결과

삽입 (랜덤파일)

랜덤파일을 이용하여 100,000개의 데이터를 삽입한 실험에서는 T^2-트리가 T-트리에 비해서 좋은 결과를 보여주고 있다. 랜덤 데이터를 삽입할 때 노드 분열이 T-트리에 비해서 T^2-트리가 더 적게 발생함에 따라 시간이 7% 더 적게 걸림을 알 수 있다. 또한 이에 따라 노드 분열이 적게 발생하고 그에 따른 회전 시간이 T-트리에 비해서 T^2-트리가 좋음을 알 수 있었다.

삭제

삭제 실험은 100,000개의 데이터를 삽입 후 10,000 개의 데이터를 삭제하는 실험을 하였다. 이 실험에서는 삭제 후 스레드 포인터와 후위 포인터 연결에 드는 시간 때문에 T-트리 보다 T^2-트리가 성능이 25% 떨어진 것으로 나타났다.

검색

검색 실험은 데이터 삽입 후 100개 테이터를 직접 검색하는 실험과 범위 검색 실험을 하였다. 직접 검색의 경우 T-트리와 T^2-트리가 별 차이를 보이지 않았지만, 범위 검색의 경우는 후위 포인터를 사용하는 T^2-트리가 T-트리 보다 12.5% 빠른 검색 결과를 보임을 알 수 있다.

혼합

이 논문에서 가장 중요한 실험으로 50,000개의 데이터를 삽입 후 삽입과 삭제를 각 25,000개 씩 랜덤 하게 실행하였다. 실험 결과 T^2-트리가 T-트리에 비해서 4.13% 좋음을 알 수 있다. 이 실험으로 삽입과 삭제가 많은 동적 데이터베이스 환경에서 T^2-트리가 T-트리에 비해 효율적임을 알 수 있다.

4. 결론

본 실험에서는 기존의 주기억 데이터베이스 색인 구조인 T-트리와 T^2-트리를 주기억 데이터베이스 프로그램인 FastDB에 적용하여 실험하였다. 삽입 실험 경우 순차파일을 사용한 경우는 T-트리가 T^2-트리 보다 3.5% 더 좋은 결과를 나타내고 있으나 랜덤파일을 이용한 경우는 T^2-트리가 7% 좋은 결과를 보여 주고 있다. 이것은 T^2-트리가 노드 내 실제 데이터 삽입시간과 노드의 회전시간이 T-트리에 비해 좋은 결과를 나타내기 때문이다.

삭제 실험은 T^2-트리가 T-트리에 비해 성능이 떨어짐을 알 수 있다. 이것은 T^2-트리의 스레드 포인터와 후위 포인터의 연결에 드는 오버헤드 때문이다.

검색 실험의 경우는 직접 검색의 경우 T-트리와 T^2-트리가 거의 같은 시간을 보이지만, 범위 질의의 경우 T^2-트리의 목적에 부합되게 T-트리에 비해 12.5% 좋은 결과를 보이고 있다.

마지막 혼합 실험과 같이 삽입 삭제가 많은 경우 T^2-트리가 T-트리에 비해서 4.13% 좋은 결과를 가져왔다. 이러한 결과로 보면 전체적으로 동적 데이터베이스 환경에서 T^2-트리가 T-트리에 비해서 효율적인 색인 구조임을 알 수 있다.

참고 문헌

[1] T. J. Lehman and M. J. Carey, A Study of Index Structures for Main Memory Database Management Systems, In Proceedings of the *12th VLDB Conference*, Kyoto, August, pp. 294-303, 1986

[2] 김태진, T^2-트리: 동적 주기억 데이터베이스를 위한 효율적 색인 구조, 정보과학회, Vol. 26. No. 2, pp.258-260, 1999

[3] http://www.ispras.ru/~knizhnik

[4] R. Bayer and E. M. McCreight, Organization and Maintenance of Large Ordered Indexes, Acta Informatica 1, pp.173-189, 1972

[5] G. M. Adelson-Velskii and E. M. Landis, An Algorithm for the Organization of Information, Soviet Math. Doclady 3, pp.1259-1263, 1962

[6] Kong-Rim Choi and Kyung-Chang Kim, T*-Tree: A Main Memory Database Index Structure for Real Time Application, *In Proceedings of the Third International Workshop on Real Time Computing System Application*, pp.81-88, 1996

[7] A. J. Perlis and C. Thornton, "Symbol Manipulation by Threaded List", CACM, Vol. 3, No. 4, pp.173.-189 April, 1960