부분키를 사용한 캐쉬 인식 B^+ 트리의 성능 평가[7)]

Performance Evaluation of Cache Sensitive B^+-tree

김원식, 한욱신
경북대학교 컴퓨터공학과

Won-Sik Kim, Wook-Shin Han
Kyungpook National University

요약

부분키를 사용한 캐쉬 인식 B^+ 트리는 키 압축과 포인터 압축 방법을 동시에 적용한 캐쉬 인식 트리이다. 기존의 캐쉬 인식 트리들은 키 압축과 포인터 압축을 따로 고려하였다. 이에 반해 부분키를 사용한 캐쉬 인식 B^+ 트리는 키와 포인터를 동시에 압축하여 캐쉬 활용도를 높였다. 본 논문은 기 발표된 부분키를 사용한 캐쉬 인식 B^+ 트리의 벌크로드와 검색 알고리즘을 구현하여 성능 평가를 수행하였다. 그리고 B^+-트리와 Simple Prefix B^+-트리의 성능비교를 통하여 부분키 캐쉬 인식 B^+ 트리의 성능의 우수함을 확인 하였다.

Abstract

Cache sensitive B+-trees with partial keys is cache sensitive tree using both key compression and pointer compression. Although conventional cache sensitive trees consider individuallykey compression and pointer compression, cache sensitive B^+-trees with partial keys make more cache utilization by compressing both key and pointer. We implement bulkload and search algorithms of cache sensitive B^+-tree with partial key. And out performance studies show that cache sensitive B+-tree with partial key is better than B^+-tree and Simple Prefix B^+-tree.

Ⅰ. 서 론

주기억 장치로 사용되는 DRAM 가격의 하락으로 인하여 주기억 장치의 용량이 증가함에 따라 데이터베이스 인덱스가 주기억장치에 상주하는 것이 가능하게 되었다. 그러나 CPU의 처리속도와 주기억 장치의 속도의 차이 때문에 캐쉬 메모리의 활용여부가 성능에 직접적인 영향을 준다. 그러므로 CPU와 주기억 장치 사이의 캐쉬 활용도를 높여주는 캐쉬 인식 트리에 대한 연구가 활발하게 진행 중이다 [1, 2, 3].

기존의 캐쉬 인식 트리에 대한 연구는 압축 대상에 따라 1) 포인터 제거에 기반한 캐쉬 인식 트리와 2) 키 압축에 기반한 캐쉬 인식 트리로 나눌 수 있다. 기존의 캐쉬 인식 트리들은 키와 포인터 압축을 따로 고려했지만 부분키를 사용한 캐쉬 인식 B^+ 트리는 2가지 압축 방법을 통합하고 개선하여 캐쉬 활용도를 더 높인다. 본 논문은 기 발표된 부분키를 사용한 캐쉬 인식 B^+ 트리[5]의 벌크로드와 검색 알고리즘을 구현하여 성능평가를 수행하였다. 본 논문의 공헌은 다음과 같이 요약된다.

- 부분키를 사용한 캐쉬 인식 B^+ 트리의 벌크로드와 검색 알고리즘을 구현하고 성능평가를 수행하였다.
- B^+-트리와 Simple Prefix B^+-트리의 성능 비교를 통해 부분키를 사용한 캐쉬 인식 B^+ 트리의

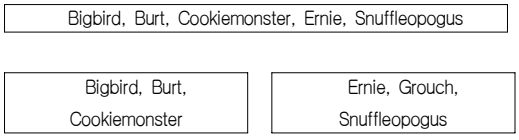
이 논문은 2003년도 한국학술진행재단의 지원에 의하여 연구되었음 (KRF-2003-003-D00347).

성능이 우수함을 확인하였다.

본 논문의 구성은 다음과 같다. 제2절에서는 Simple Prefix B^+-트리를 살펴본다. 제3절에서는 기존의 방법들을 통합, 개선한 부분키를 사용한 캐쉬 인식 B^+ 트리를 살펴보고 제4절에서는 성능평가를 수행하여 B^+-트리와 Simple Prefix B^+-트리와 성능 비교를 수행한다. 마지막 5절에서는 본 논문의 결론을 내린다.

Ⅱ. Simple Prefix B-tree [4]

검색 알고리즘에서 비단말 노드의 키는 주어진 검색 키에 대해서 어느 서브 트리를 탐색 할 것인지를 결정해 주는 역할을 한다. 예를 들어서 아래와 같이 현재 가득 찬 상태인 단말 노드가 있다고 가정하자. 이때 "Grouch"라는 키를 삽입하게 된다면 2개의 단말 노드는 분할될 것이다.



▶▶ 그림 1. 분할 전 단말 노드(상)와 분할 후 2개의 단말 노드(하).

"Ernie"이라는 키를 상위 레벨의 비단말 노드에 저장하는 대신에 "D"혹은 "E"를 저장한다면 검색 시 주어진 검색 키에 대해서 어느 서브 트리를 탐색 할 것인지 결정하는데 충분하다. 이와 같이 "Cookiemonster"와 "Ernie"를 구별해 줄 수 있는 스트링을 구분자 (separator)라고 부른다. 구분자 들의 후보군중에서 가장 작인 길이의 구분자를 선택하는 것이 비단말 노드의 차수 (degree)를 높혀 준다.

Simple Prefix B-tree는 비단말 노드의 키를 가변길이 구분자로 교체한 B^*-트리이다. 비단말 노드의 차수를 높음으로 Simple Prefix B-tree의 높이는 낮아지게 된다. 그러므로 검색시 B-트리 보다 더 좋은 성능을 얻을 수 있다.

Ⅲ. 부분키를 사용한 캐쉬 인식 B^+ 트리 [5][8)]

부분키를 사용한 캐쉬 인식 B^+ 트리는 키 압축 방법과 포인터 압축 방법을 동시에 적용한 CSB^+-트리 [3]의 변형으로서, 정의 1에서 정형적으로 정의한다.

정의 1. 차수(order) k인 부분키-CSB^+ 트리는 다음과 같이 정의된다.

1. 모든 비단말 노드는 다음과 같이 구성된다.
   a. 오름차순으로 정렬된 n개의 키 K_i의 부분키 PK_i $\left(\frac{k}{2} \leqq n < 2\right)$
   b. 첫 번째 자식 노드에 대한 포인터
   c. n개의 부분키 PKi ($1 \leq i \leq n$)가 저장된 비단말 노드의 j번째 자식 노드에 속한 모든 키 X의 범위는 다음과 같다.
      $K_{j-1} \leq X < K_j$ ($1 < j \leq n$); $X < K_j$ ($j=1$); $K_{j-1} < X$ ($j=n+1$)
2. 모든 단말 노드는 다음과 같이 구성된다.[9)]
   a. 오름차순으로 정렬된 n개의 키 K_i의 부분키 PK_i $\left(\frac{k}{2} \leqq n < 2\right)$
3. 모든 단말 노드는 같은 레벨에 존재한다.
4. n개의 부분키가 저장된 비단말 노드는 n+1개의 자식노드를 가지는데, 이 자식 노드들은 주기억장치 내에 차례대로 연속하여 저장되며 하나의 노드 그룹을 형성한다.
5. 노드 N에 저장된 각각의 키의 베이스 키의 위치는 다음과 같다.

a. N에 속한 키들 중에서 가장 왼쪽 키를 제외한

8) 아래의 부분키를 사용한 캐쉬 인식 B+트리의 정의는 기 발표된 논문[5]의 정의를 인용하였습니다.

9) 부분키의 ref로 데이터 레코드에 대한 포인터가 저장되므로 단말 노드에서 다시 데이터 레코드에 대한 포인터를 저장할 필요가 없다.

모든 키의 베이스 키는 각각 그 왼쪽 이웃 키가 된다.

b. N에 속한 가장 왼쪽 키(leftmost key)의 베이스 키는 부모 노드에 속한, N의 가장 왼쪽 키(leftmost key)보다 작거나 같은 키들 중에서 가장 큰 키가 된다.

c. 노드 N이 그 부모 노드의 첫번째 자식 노드인 경우, N의 가장 왼쪽키의 베이스 키는 그 부모 노드의 가장 왼쪽 키의 베이스 키와 동일하다.

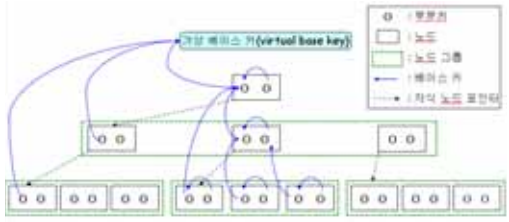
d. 트리의 각 레벨에서 가장 왼쪽 노드(leftmost node)의 가장 왼쪽 키의 베이스 키는 키의 정의역의 최소값으로 하며 이를 가상 베이스 키(virtual base key)라 부른다.

정의 1에서 부분 키는 pkB-trees[1] 에서 사용한부분 키와 동일한 구조를 가진다. 그러나 트리 각 레벨의 가장 왼쪽 노드의 가장 왼쪽 키는 트리내에 그 베이스 키가 존재하지 않게 되는 문제점을 가지며 논문[1]에서는 이에 대한 언급이 없다. 이를 해결하기 위해 이러한 키들은 트리마다 유일하게(unique) 정해진 공통의 베이스 키를 가지도록 하였고, 이 키를 가상 베이스 키라고 부른다. 가상 베이스 키는 저장할 필요 없이 키의 정의역의 최소값으로 해석한다.

그림 2는 부분키-CSB$^+$ 트리의 예를 나타내고 있다. 그림에서 실선 사각형은 노드를, 점선 사각형은 노드 그룹을 나타내고 있다. 그리고 실선 화살표는 주어진 키에 대한 베이스 키를 표현한다. 정의에서 설명하였듯이 각 레벨의 맨 왼쪽 노드의 맨 왼쪽 부분키의 베이스 키는 가상 베이스 키가 된다. 그리고 CSB$^+$-트리와 같이 임의 노드의 자식 노드들은 하나의 노드 그룹으로 연속적으로 주기억 장치 내에 할당되어 있다.

Ⅳ. 성능 평가

본 절에서는 부분 키를 사용한 캐쉬 인식 트리의 성능 평가를 통하여 B+-트리,Simple Prefix B$^+$-tree와 성능 비교를 설명한다. 제 5.1절에서는 성능 평가를 수행한 실험 환경에 대해서 설명하고 제 5.2절에서는 실험결과를 설명한다.



▶▶ 그림 2. 부분키-CSB$^+$ 트리의 예.

1. 실험 환경

본 실험은 Intel Pentium Ⅲ M 850MHz CPU, 448MB RAM, Windows XP professional 환경에서 수행하였다. Pentium Ⅲ M CPU는 2레벨의 캐쉬를 가지고 있다. L1 캐쉬는 총 16KB로서 32 byte의 캐쉬 라인들로 구성되어 있다. L2 캐쉬는 총 256KB로서 32 byte의 캐쉬 라인으로 구성되어 있다.

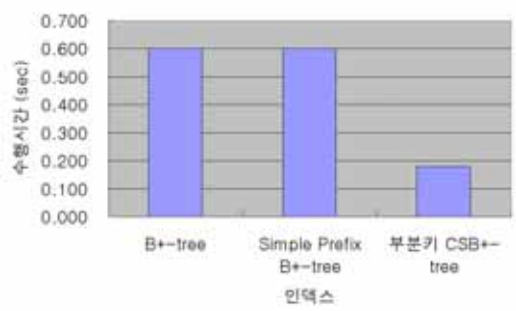
본 실험에서 사용하는 데이터, 즉 로이터 통신 데이터와 랜덤으로 생성된 데이터를 사용한다. 두 데이터의 키들의 길이는 1부터 10사이고 중복값이 없는 유일한 키로 구성되어 있다. 로이터 통신 데이터의 30만개에 대해서 실험을 하였다. 랜덤으로 생성된 데이터는 총 100만개에 대해서 실험을 수행하였다.

본 실험에서는 성능평가 요소로써 수행시간를 측정하였다. 실험은 3번 실시하여 최소값을 선택하였다. 각 데이터에 대해서 벌크로드 수행시간과 검색 수행시간을 측정하였다. 노드크기는 128byte로 고정하였다.

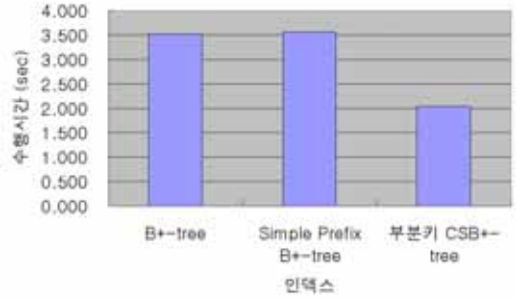
2. 실험 결과

벌크로드 수행시간은 그림3과 그림 5를 통하여 부분키 CSB$^+$-tree가 B$^+$-tree와 Simple Prefix B$^+$-tree 보다 빠르다는 것을 확인할 수 있다. 검색 수행시간은 그림 4와 그림 5를 통하여 부분키

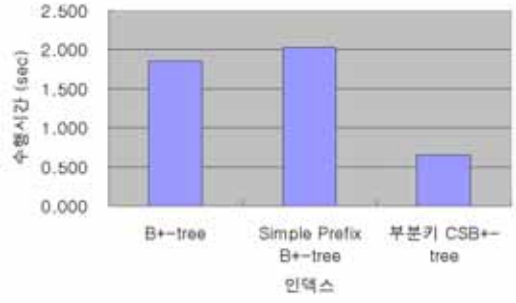
CSB$^+$-tree가 B$^+$-tree와 Simple Prefix B$^+$-tree 보다 빠르다는 것을 확인할 수 있다. 이는 아래와 같이 분석될 수 있다. 부분키 CSB$^+$-tree가 키와 포인트를 둘다 압축하므로 노드의 블락킹 요소를 향상시켜서 검색 성능을 향상시켰다고 분석 할 수 있다.



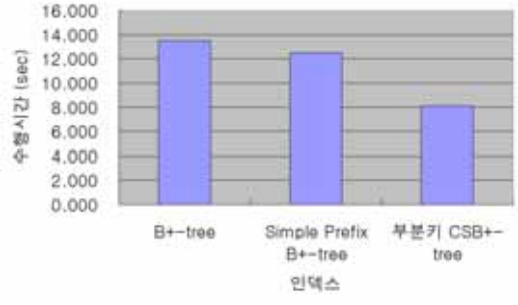
▶▶ 그림 3. 로이터 통신 데이터의 벌크로드 수행시간.



▶▶ 그림 4. 로이터 통신 데이터의 검색 수행시간.



▶▶ 그림 5. 랜덤 데이터의 벌크로드 수행시간.



▶▶ 그림 6. 랜덤 데이터의 검색 수행시간.

2개의 데이터에 대한 벌크로드와 검색 수행 시간을 통해 부분키 CSB$^+$-tree가 B$^+$-tree와 Simple Prefix B$^+$-tree보다 성능의 우수하다는 것을 확인 할 수 있다.

Ⅴ. 결론 및 향후 연구과제

본 논문을 기발된 부분키를 사용한 캐쉬 인식 B$^+$ 트리의 벌크로드와 검색 알고리즘을 구현하여 성능 평가를 수행하였다. 부분키를 사용한 캐쉬 인식 B$^+$ 트리는 키 압축과 포인터 압축 방법을 동시에 적용한 캐쉬 인식 트리이다. 키와 포인터를 동시에 압축으로써 부분키를 사용한 캐쉬 인식 B+ 트리가 B$^+$-tree와 Simple Prefix B$^+$-tree보다 성능이 우수하다는 것을 실험을 통해서 확인 할 수 있었다.

향후 연구 과제로써 아직 구현하지 않은 삽입과 삭제 알고리즘을 구현하여 B$^+$-tree와 Simple Prefix B$^+$-tree와 성능 비교를 수행할 예정이다.

▌참고문헌▐

[1] P. Bohannon, P. Mcilroy, R. Rastogi. "Main-Memory Index Structures with Fixed-Size Partial Keys. "In ACM SIGMOD 2001.

[2] J.Rao and K.A. Ross. "Cache conscious indexing for decision-support in main memory," In VLDB 99.

[3] J. Rao and K.A. Ross. "Making B$^+$-trees cache

conscious in main memory," In ACM SIGMOD 2000
[4] Bayer, R., and Unterauer, K. "Prefix B-trees, "ACM Trans. Datab. Syst, 1977
[5] 이동민, 김원식, 한욱신, "부분키를 사용한 캐쉬 인식 B^+ 트리," 한국 정보 과학회 춘계 학술대회, 2004.