부분키를 사용한 캐서 인식 B 트리

이동민, 김원석, 한덕진
경북대학교 컴퓨터공학과

Abstract

본 논문에서는 기존의 캐서 인식 트리에서 사용된 기존 기능을 살펴보고 이를 통합, 개선하여 새로운 캐서 인식 트리를 제안한다. 기존의 캐서 인식 트리는 키 압축과 포인터 압축을 따로 고려하였기 때문에 각각 노드의 크기가 커져 나타난 문제에 최적의 문제점이 있다. 이에 반해 제안하는 부분키 캐서 인식 B 트리는 키 포인터를 동시에 압축하여 이러한 문제점

---

1. 서론

주기적 저장한 사용하는 DRAM의 가격이 급격하게 하락함에 따라, 데이터베이스 인덱스를 주기적 저장에 상주할 수 있게끔 기가능하게 되었다. 그러나, CPU의 처리속도가 주기적 저장 속도의 차이가 여전히 크고 끊임 없이 발생하면서, 주기적 저장 속도를 높이기 위한 많은 연구가 진행중이다. 즉, CPU의 주기적 저장의 속도와 비교하여 더 빠르게 수행하여, 성능이 개선된 캐서가 필요하다.

본 논문의 구성은 다음과 같다. 제1절에서는 기존에 연구되었던 캐서 인식 트리를 살펴보고 정점을 논한다. 제2절에서는 주기적 저장의 방법론을 통한, 개선된 새로운 캐서 인식 트리인 부분키-CBS 트리의 개발을 제안하고, 제3절에서는 실제 테스트와 실험을 설명한다. 제4절에서는 검색 알고리즘에 대한 논의와 제5절에서는 전체 부분키-CBS 트리와 관련된 캐서 인식 트리에 대한 이론적 토론을 제시한다. 마지막으로 제6절에서는 본 논문의 결론을 내린다.

2. 관련 연구

2.1 CBS 트리


노드 나노의 부분키 노드에 대한 CSB 트리의 함으로서 구현이 가능하고, 다른 자체 노드들 내에 포함되어 있는 오직으로 할 수 있다. 도노드의 저장한 캐서 인식 트리가 크기와 CSB 트리의 바이트 노드에 대한 분산을 보는 특성의 노드를 통해 노드의 복잡성을 가진다. 그러나, 이러한 보다는 중요한 캐서 인식 트리의 크기는 크기가 크기가 본 영향을 줄여서 복잡성 인수에 큰 영향을 준다. 이를 통해 고려하는 것이 필요하다.

기압축에 기반한 부분키-CBS 트리는 연속된 캐서 사이에서 차이가 나는 부분 중 고정값이 일정한 저장장치를 사용하는 방법[3], 따라서 간단한 캐서 인식 트리에서 사용하는 카시이의 압축에서 최소한의 노드의 개수를 줄일 수 있다. 그러나, 간단한 연속치에서는 복잡한 인식 트리의 연속작중을 할 수 있는 문제점을 가진다. 특히, 64비트 이상의 경우 포인터 크기가 크거나 빠르나 되으면 포인터 제어도 함께 고려하는 것이 필요하다.

또한, 본 논문에서는 이러한 단점적인 두 가지 같은 방법을 통합하고, 개선을 수행하도록 높고 간단한 경로를 짜는 노드 구조의 부분키-CBS 트리를 제안한다. 본 논문의 기반은 다음과 같다. 이단, 주요 함수, 포인터 계층에 있는 부분키-CBS 트리 및 개별 포인터 계층을 포함한 후보기능의 분산를 강화하게 된다. 또한, 기압축에 기반한 부분키-CBS 트리의 개발

---

자료: KRF-2003-D03-D00547.
본 논문에서 제안하는 부품키 CSB-*트리는 키 압축 방법인 컬러 압축 방법과 포인터 압축 방법 등을 통해 적용한 CSB*-트리의 변형으로, 3.1절에서 정의하였다.

3.1. 하위 트리의 예

부품키 B*-트리에는 부품키를 이용한 새로운 검색 알고리즘과 탐색목록을 통하여 매우 빠르게 데이터를 찾을 수 있다. 그러나 이와 함께 데이터의 분산성을 고려한 집관 키의 산출도 고려한 집관 요소를 가질 필요가 있다. 이와 같이 작게 촉소된 부품 키의 크기가 또다른 부품 키의 크기와 비슷한 경우, 부품 키의 산출도 고려한 집관 요소를 할당하여 부품 키가 부족한 정보를 보완하기 위해서야 한다. 부품 키의 산출도 고려한 집관 요소를 할당하여 부품 키가 부족한 정보를 보완하기 위해서 압축된 부품키의 뒤에 부품키의 크기가 작아지는 경우, 부품 키의 산출도 고려한 집관 요소를 할당하여 부품 키가 부족한 정보를 보완하기 위해서야 한다.
보조정리 1.2 부분기-CSV 트리에서 키 삽입시에 노드는 분리(split) 되어 노드 크기가 분리되지 않는 경우, 노드 분리에 의해 최대 한 개의 키와의 베이스 키가 변환되어, 이에 따라 한변의 캐시 미사가 발생한다.

증명 생략.
노드 분리 이후 키 삽입은 보조정리 1.1을 따름.

보조정리 1.3 부분기-CSV 트리에서 키 삽입시에 노드와 노드 그룹이 다 분리(splits)되는 경우, 노드 그룹 분리시에는 노드의 빈도만 있고 키와의 빈도는 없으므로, 아예 한키의 베이스 짧아 변환되지 않는다.

증명 생략.

삽입 알고리즘은 보조정리 1.1 - 1.3을 따름으로, 자연적 생략한다.

4.3 삭제
키의 삭제에 의한 노드 또는 노드 그룹의 병합(merge) 및 노드나 키의 재정렬 redistribution에 의한 부분기 크기를 보조정리 2.1~2.3을 따른다.

보조정리 2.1 부분기-CSV 트리에서 키 삭제시에 노드의 병합 및 키의 재정렬이 없는 경우, 최대 한 개의 키와의 베이스 키가 변환되고, 최대 한변의 캐시 미사가 발생한다.

증명 생략.

보조정리 2.2 부분기-CSV 트리에서 키 삭제시에 노드의 병합 혹은 키의 재정렬이 없는 경우, 두 노드의 병합에 의해 최대 2개의 키와의 베이스 키가 변환되고 최대 2번의 캐시 미사가 발생하며, 두 노드의 키의 재정렬에 의해서는 최대 2개의 키와의 베이스 키가 변환되고 최대 3번의 캐시 미사가 발생한다.

증명 생략.

노드의 병합 및 키의 재정렬 이전에 노드에서 키의 삭제는 보조정리 2.1을 따른다.

보조정리 2.3 부분기-CSV 트리에서 키 삭제시에 노드의 병합 혹은 노드들의 재정렬이 없는 경우, 노드의 그룹 병합시 노드가 재정렬되었을 때 노드의 키와의 베이스 키의 값은 변한다. 증명 생략.

노드 그룹의 병합 및 노드들의 재정렬 이전에 노드에서 키의 삭제 그리고 노드의 병합 및 키의 재정렬에 대한 보조정리 2.1, 2.2를 따른다.

삭제 알고리즘은 보조정리 2.1 - 2.3을 따름으로, 자연적 생략한다.

5. 성능분석
기존 캐시 인덱스 트리와 재정렬하는 부분기-CSV 트리의 병합 병합 인수는 표 2.3과 같다.

<table>
<thead>
<tr>
<th>표 2. 2</th>
<th>비트 병합</th>
<th>비트 비트 병합</th>
<th>비트 비트 병합</th>
</tr>
</thead>
<tbody>
<tr>
<td>B 트리</td>
<td>N(K+4)</td>
<td>N(K+4)</td>
<td>N(K+4)</td>
</tr>
<tr>
<td>CSB* 트리(작업자정)</td>
<td>N(K+4)</td>
<td>(N-4)/K</td>
<td>(N-4)/K</td>
</tr>
<tr>
<td>CSB* 트리(작업자정)</td>
<td>N(K+4)</td>
<td>(N-4)/K</td>
<td>(N-4)/K</td>
</tr>
<tr>
<td>부분기 B 트리</td>
<td>N(K+4)</td>
<td>N(K+4)</td>
<td>N(K+4)</td>
</tr>
<tr>
<td>부분기-CSB 트리</td>
<td>N(K+4)</td>
<td>N(K+4)</td>
<td>N(K+4)</td>
</tr>
</tbody>
</table>


\[ h = \left\lceil \log_2 \left( \frac{N}{b} \right) \right\rceil \]

B 트리와 CSB 트리의 노드 내에서 키의 값에 나중 값을 사용할 때 캐시 빈도의 크기를 총 때, 노드가 발생하는 캐시 미사의 최대 횟수 M은 다음과 같다.

\[ M = \left\lceil \log_2 \left( \frac{N}{b} \right) \right\rceil \]

부분기 B 트리와 부분기-CSB 트리의 노드 내에서 키의 값에 나중 값을 사용할 때, 노드가 발생하는 캐시 미사의 최대 횟수 M은 다음과 같다.

\[ M = \left\lceil \log_2 \left( \frac{N}{b} \right) \right\rceil \]

트리 검색식 때 각 레벨당 하나의 노드가 작업 되므로, 트리 검색식에 발생하는 캐시 미사의 최대 횟수 M은 식 4.3과 같다.

\[ M = M_r \times h \]

崂로는 각각의 노드의 작업 병합 이나, 트리 검색식에 발생하는 캐시 미사의 최대 횟수 M은 식 4.3과 같다.

\[ M = M_r \times h \]

또한 본 논문에는 기존의 캐시 인덱스 트리에서 적용된 기술들을 살펴보고, 이를 바탕으로 보다 캐시 활용도 높인 부분기-CSB 트리를 재정렬한다.

본 논문에서는 기존의 캐시 인덱스 트리에서 적용된 기술들을 살펴보고, 이를 바탕으로 보다 캐시 활용도 높인 부분기-CSB 트리를 재정렬한다.

본 논문에서는 기존의 캐시 인덱스 트리에서 적용된 기술들을 살펴보고, 이를 바탕으로 보다 캐시 활용도 높인 부분기-CSB 트리를 재정렬한다.

또한 본 논문에서는 기존의 캐시 인덱스 트리에서 적용된 기술들을 살펴보고, 이를 바탕으로 보다 캐시 활용도 높인 부분기-CSB 트리를 재정렬한다.

또한 본 논문에서는 기존의 캐시 인덱스 트리에서 적용된 기술들을 살펴보고, 이를 바탕으로 보다 캐시 활용도 높인 부분기-CSB 트리를 재정렬한다.

또한 본 논문에서는 기존의 캐시 인덱스 트리에서 적용된 기술들을 살펴보고, 이를 바탕으로 보다 캐시 활용도 높인 부분기-CSB 트리를 재정렬한다.

전체 알고리즘은 끝내기로, 삽입, 삭제, 검색 순서를 준수하며, 정상 동작을 보인다. 각 알고리즘이 올바르게 수행되어 올바른 결과를 제공한다.

참고 문헌