일괄구성과 확장된 지연삭제를 이용한 B$^+$-Tree의 동시성 제어 및 회복

김대일, 김성희, 조숙경, 배해영

인하대학교 전자계산공학과

g1991250@inhavision.inha.ac.kr

Concurrency Control and Recovery Method of B $^+$-Tree using Bulk Loading and Extended Lazy Deletion

Dae-Il Kim, Sung-Hee Kim, Sook-Kyoung Cho, Hae-Young Bae

Dept. of Computer Science, Inha University

요 약

B$^+$-Tree는 데이터베이스 관리 시스템에서 대용량의 데이터를 관리하기위해 가장 널리 사용되는 인덱스이다. 그러나 기존의B$^+$-Tree는 데이터베이스의 초기 구성 및 재구성시 많은 비용이 들고, 또한 삭제 연산의 빈번한 발생시 색인 구조 변경연산의 발생빈도가 높아져 동시성이 떨어진다는 단점이 있다. 이러한 문제점을 해결하기 위해서 기존 대부분의 데이터베이스 관리시스템에서는 일괄구성과 지연삭제를 이용하고 있으나, 동시성 및 회복에 대한 처리가 미흡하여 실제 시스템에 적용하기에는 문제가 있다. 따라서 본 논문에서는 일괄구성과 지연삭제 방법을 적용한B$^+$-Tree에서의 동시성 및 회복기법을 제안한다. 제안된 기법은 일괄구성 시에 잠금의 부하와 연속적인 철회(Cascade Rollback)가 없고, 또한 지연 삭제기법을 확장함으로써 빈 페이지 리스트 관리에 대한 부하가 없으며, 삭제 연산에 대한 회복 시 논리적 복귀(Logical Undo)가 빨라지고 구현이 간단해진다는 장점이 있다.

1. 서 론

B$^+$-Tree는 데이터베이스 관리 시스템에서 대용량의 데이터를 효율적으로 관리하기위해서 가장 널리 사용되는 인덱스이다. 하지만 기존의B$^+$-Tree는 데이터베이스를 처음 구성할 때나, 인덱스를 새로 구성할 시, 많은 DISK/IO가 발생하고, 삭제 연산이 빈번할 시에는 색인 구조 변경연산이 많이 발생하여 동시성이 떨어진다는 단점이 있다[4]. 이러한 문제점을 해결하기 위해서 대부분의 데이터베이스 시스템에서는 일괄구성과 지연삭제를 이용한다[4,5,7]. 하지만 일괄구성 및 지연(Lazy) 삭제 방법을 적용한B$^+$-Tree를 데이터베이스 관리시스템에 적용하려면 동시성 제어 및 회복 기법이 필수적으로 요구되지만, 이에 대한 연구가 미흡하여 실제 시스템에 적용하기에는 문제가 있다.

본 논문에서는 기존의B$^+$-Tree에서 빠른 삽입, 삭제를 지원하기 위해서 일괄구성과 확장된 지연(Lazy)삭제 방법을 이용하고, 일괄구성과 확장된 지연(Lazy) 삭제방법을 적용한B$^+$-Tree의 동시성 제어 및 회복기법을 제안한다.

제안하는 기법은 기존의B$^+$-Tree에 동시성과 회복에 대해 언급한 [3]에 기반하고, 일괄구성 시에 잠금의 부하와 연속적인 철회(Cascade Rollback)가 없고, 또한 지연 삭제기법을 확장함으로써 빈 페이지 리스트 관리에 대한 부하가 없으며, 삭제 연산에 대한 회복 시 논리적 취소(Logical Undo)가 빨라지고 구현이 간단해진다는 장점이 있다

본 논문의 구성은 다음과 같다. 2 장에서는 본 논문이 기반하고 있는 ARIES/IM과 B$^+$-Tree의 성능향상을 위해 사용된 일괄구성 및 지연삭제(Lazy Deletion) 에 대해 알아보고, 3 장에서는 일괄구성 및 확장된 지연삭제(Extended Lazy Deletion) 를 이용한B$^+$-Tree의 동시성 제어 및 회복기법을 설명하고, 4 장에서는 결론을 제시한다.

* 본 연구는 정보통신부의 대학 S/W 연구센터 지원사업의 연구 결과임.

2. 관련연구

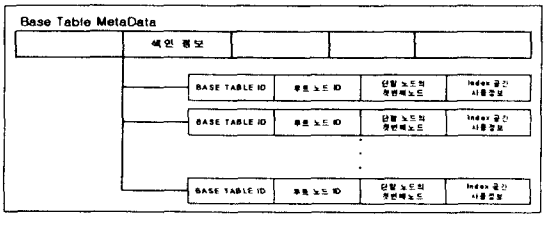
기존의 B⁺-Tree의 동시성과 회복기법으로는 ARIES을 기반으로 한 ARIES/IM이 대표적이다[2,3]. 이외에도 [5][6] 에서 제안한 기법이 있지만 회복기법에 대한 고려가 없어서 데이터베이스 관리시스템에 적용하기에는 부적절하다[3]. ARIES/IM 은 래치와 잠금을 사용하여 동시성 제어를 수행하고, 회복은 WAL(Write Ahead Logging) 과 페이지 지향 재수행(Page Oriented Redo), 그리고 논리적 복귀(Logical Undo)를 기반으로 수행된다[2,3]. 또한 동시성 성능을 향상시키기 위해서 모든 변경 연산을 내용 변경 연산과 트리 구조 변경연산으로 나누고, 트리 구조 변경 연산은 NTA(Nested Top Action)[3] 로 처리한다.

일괄구성은 데이터베이스를 처음 구성할 때나, 인덱스를 새로 구성할 시 DISK/IO 의 비용을 최소로 하기 위해서 상향식(Bottom-Up)으로 인덱스를 구성하는 것이며, 지연삭제는 빈번한 삭제시 색인구조변경 연산의 발생빈도를 줄이기 위해서 B⁺-Tree의 노드에 키의 개수가 0개가 될 때까지 색인 구조 변경연산을 지연하는 기법이다[4,5,7].

3. 개선된 B⁺-Tree의 동시성 제어 및 회복 기법

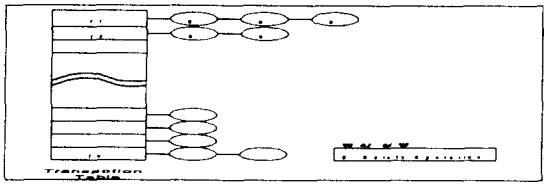
3.1. 자료구조

본 논문에서 제안하는 동시성 및 회복 기법을 위한 자료구조인 테이블과 인덱스의 메타데이타, Pending 리스트, 트랜잭션 테이블과, 시스템 Queue 가 있고 각각의 역할 및 구조는 다음과 같다.



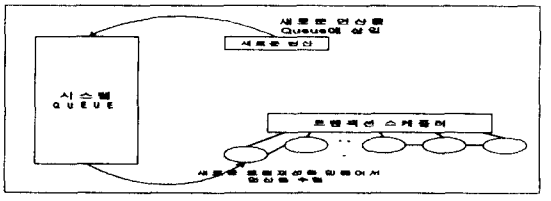
[그림 1] Meta Data

첫번째로 테이블과 인덱스 메타데이타는 각 시스템마다 다를 수 있기 때문에 본 논문에서는 필요한 인덱스에 관계된 데이터만을 언급하겠다. [그림 1] 에서 테이블은 자신에게 속해 있는 인덱스 정보를 가지고 있고, 인덱스는 루트 노드에 ID, 단말노드의 첫번째 노드 ID, 인덱스의 공간사용 정보를 가지고 있다.



[그림 2] 트랜잭션 Table 과 Pending List

두번째로 트랜잭션 테이블과 Pending 리스트는 완료시점까지 지연돼야되는 특정 연산을 위한 것으로서, 트랜잭션 테이블은 테이블 형태로 되어 있으며 각각의 엔트리에 연산 리스트인 Pending 리스트를 가진 [그림 2] 와 같은 구조를 가지고 있다. 각각의 트랜잭션이 완료시점이 되면 먼저 트랜잭션은 트랜잭션 테이블에서 자신의 엔트리를 찾아서 수행해야 할 연산이 있는지를 Pending 리스트에서 찾아서 수행한다.



[그림 3] 시스템 Queue

세번째로 시스템 Queue 는 일괄구성과 같이 수행시간이 많이 걸리는 연산이 사용자의 트랜잭션중에 수행될 때 사용자의 트랜잭션에 대한 응답시간이 길어진다. 따라서 [그림 3] 과 같이 수행시간이 긴 연산은 시스템 Queue 에 삽입하고 나중에 데이터 베이스 관리시스템의 트랜잭션 스케줄러가 주기적으로 시스템 Queue 를 검색해서 처리해야 될 작업이 있으면 별도의 시스템 트랜잭션으로 수행되도록 한다.

3.2 동시성과 회복을 고려한 일괄구성 알고리즘

일반적인 일괄구성의 동시성 제어는 베이스 테이블의 갱신연산을 방지하기위해 공유 잠금을 수행하여 테이블에 레코드의 삽입, 삭제가 허용되지 않는다.

본 제안기법은 일괄구성시에도 데이터 베이스 관리 시스템의 트랜잭션 격리 레벨이 No Phantom 까지 지원한다는 가정하에 베이스 테이블에 레코드의 삽입, 삭제가 가능하도록 하기 위하여 베이스 테이블에 공유잠금을 수행하지 않고 레코드 레벨의 잠금을 이용한다. 일괄구성시 레코드를 순차적으로 읽어가는데 이때 읽어간 데이터 이전에 삽입이나 삭제가 발생하지 않으면, 새로 구성된 색인은 베이스 테이블의 모든 데이터를 반영하고 있게 되므로 데이터베이스 관리 시스템의 일관성을 유지하게 된다. 그런데 베이스 테이블의 레코드의 개수가 많아져 요청하는 잠금의 개수가 많아질 경우 잠금테이블을 관리하는데 사용하는 자료구조인 해쉬(Hash)에 오버플로우가 빈번하게 발생하게 되어 잠금을 수행하는 비용이 커져 전체적인 데이터베이스 관리시스템의 성능저하를 가져온다.

이러한 잠금부하는 한 테이블에 대해 잠금의 개수가 커질 경우 상위 잠금으로 잠금을 상승시키는 잠금 승진(Lock Escalation) 기법을 수행하여 잠금의 부하를 줄일 수 있다. 그리고 일괄구성 후 바로 베이스 테이블의 색인 메타데이타의 갱신과 기존 색인의 삭제를 수행하면

다른 트랜잭션이 기존 인덱스의 접근이 배제됨으로서 재구성을 수행하는 동안 대기해야되기 때문에 동시성이 저하되고, 삭제된 객체의 페이지가 다른 객체에게 할당되어 연속적인 철회(Cascade Rollba ck)가 발생할 수 있다는 문제점이 있다. 따라서 메타데이타의 갱신과 기존 색인의 삭제는 트랜잭션의 완료시점까지 미루어져야 한다.

따라서 알고리즘은 다음과 같다. 1 번 라인에서는 새로운 객체를 만들고, 2 번 라인에서는 베이스 테이블의 레코드들을 차례로 읽으면서 External Sorting을 수행하고, 3번 라인에서는 일괄구성을 수행하고, 4 번 라인에서는 기존 색인의 삭제 및 메타 데이터의 갱신연산을 트랜잭션 Pending 리스트에 추가한다.

[알고리즘 1] 동시성과 회복을 고려한 일괄구성

```
Algorithm1: BulkLoadingBTree(IDBaseTable)
Input: IDBaseTable : 베이스 테이블 ID
01:    NewIndexID = CreateNewIndex
02:    PreOperation(IDBaseTable, NewIndex)
03:    BuildNewIndexByBulkLoading(IDBaseTable, NewIndex)
04:    AddTransPe  ndList(IDBaseTable, NewIndex)
End of Algorithm 1
```

3.3 확장된 지연 삭제 알고리즘

기존의 Lazy 삭제는 페이지의 키의 개수가 0 개가 될 때, 색인구조 변경 연산을 수행하는데 이런 색인구조변경연산은 빈 페이지 리스트를 관리해야 되는 부하와 동시성 저하라는 문제가 발생한다. 그래서 본 논문은 노드에 키의 개수가 0 개가 되더라도 색인 구조 변경연산을 수행하지 않는 확장된 지연 삭제를 수행하고자 한다. 그런데 확장된 지연 삭제를 수행하게 되면, 삭제연산이 삽입연산보다 많게 될 때, 키의 개수에 비해 B+-Tree의 높이가 높아져 검색 시에 여러 개의 페이지를 접근하게 되어 성능이 떨어진다는 문제가 있어서 인덱스의 공간 사용이 저하될 시에 다시 재구성을 수행하여야 한다[5]. 하지만 재구성을 사용자의 트랜잭션중에 수행하게 되면 사용자의 트랜잭션의 수행시간이 길어지게 된다는 문제점이 있어 본 논문에서는 [그림 2] 의 시스템 Queue 에 재구성 연산을 삽입하고, 데이터 베이스 관리시스템이 독립된 시스템 트랜잭션을 만들어서 처리하도록 하여 사용자의 트랜잭션의 응답시간을 빠르게 하였다.

회복은 ARIES/IM 과 동일하지만 삭제시에 색인변경연산을 고려할 필요가 없고, 또한 할당된 페이지의 반환연산이 없기 때문에 빈 페이지리스트에 대해 관리할 필요가 없다.

[알고리즘 2] 확장된 삭제 알고리즘

```
Algorithm 2: ExtendedLazyDeletion
Input: IDIndex: 인덱스 식별자, Key :, 키
01:      IM_DeleteKey( 인덱스 ID, Key)
02:    If CheckIndexMetaData(  인덱스 ID) then
03:      Add Operation to System Queue
04:    end if
End of Algorithm
```

[알고리즘 2] 는 삭제 알고리즘으로서 1 번 라인에서 인덱스에 대해서 삭제를 수행한 후 삭제한 인덱스에 대해서 공간 사용정보를 체크해서 재구성 연산을 시스템 Queue 에 추가한다.

4. 결 론

본 논문에서는 일괄구성과 지연삭제 방법을 적용한 B+-Tree를 실제 데이터베이스 시스템에 적용할 수 있도록 동시성 및 회복기법을 제안했다. 제안기법은 트랜잭션 Pending 리스트를 사용하여 연속적인 철회(Cascade Rollback) 가 발생하지 않고, 재구성시 베이스 테이블에 대한 삽입, 삭제를 가능하게 하여 동시성을 높였으며, 시스템 Queue를 사용하여 일괄구성 연산을 사용자의 트랜잭션에서 수행하지 않고 따로 시스템 트랜잭션으로 처리하여 사용자의 트랜잭션의 응답시간을 줄였다.

참고문헌

[1] Bayer, R., and Schkolnick, M. "Concurrency of operations on B-Trees,," Acta inf. 9(1977), 1-12.

[2] C. Mohan, D. Harderle, B. Lindsay, H. Pirahech, and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Lockin g and Partial Rollbacks Using Write Ahead Logging," ACM TODS, 17(1), pages 94-162, 1992.

[3] C. Mohan, D. Harderle, B. Lindsay, "ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging," In Proc. ACM SIGMOD Conf., page s 371-380, June 1992.

[4] Jan Jannink "Implementing Deletion in B +-Trees" SIGMOD RECORD, volume 24, number1, March 1995, pages 33-38.

[5] PHLIP L. LEHMAN, S. BING YAO "Efficient Locking for Concurrency Operations on B +-Trees," Reading In Databases Systems, pages 224-234.

[6] T. Johnson and D. Shasha. "The performance of current B +-trees algorithms," ACM Transactions on Database Systems, 18(1): 51-101, 1993.

[7] 김상욱 "B+트리의 일괄 구성: 알고리즘 및 성능 분석" 한국정보과학회 제 23 권 제 7 호, pages 1113-1121.