트랜잭션의 동시성제어를 위해 갱신전 이미지를 이용한 일시적 버전 제어 기법

김호석[0*] 김명근* 조숙경* 배해영*
*인하대학교 전자계산공학과
bluesnow@dreamx.net[0], KimMKeun@netsgo.com, skyoe@netsgo.com, hybae@inha.ac.kr

Transient Versioning Algorithm Using Before-Image for Concurrency Control of Transaction

Ho-Seok Kim[0*], Myung-Keun Kim*, Sook-Kyoung Cho*, Hae-Young Bae*
Dept. of Computer Science & Engineering, Inha University

요 약

데이터베이스시스템에서 트랜잭션의 동시성제어에 관련된 문제를 해결하고 성능을 향상시키기 위해서 많은 기법들이 소개되어졌다. 그 중에서 멀티버전(Multi-Version) 알고리즘은 각 트랜잭션간의 상호간섭을 최소화시키면서 동시성을 향상시키기 위한 알고리즘 중 하나이다. 하지만 멀티버전 알고리즘은 데이터베이스에 레코드에 대한 버전을 저장하기 위한 저장공간의 낭비와 버전에 대한 최신의 정보를 유지하기 위한 Garbage Collection 연산의 부하가 따른다.

본 논문에서는 트랜잭션간의 동시성을 향상시키기 위한 방법으로 시스템의 메모리 공간에 일시적인 레코드의 버전을 생성하여 버전 풀(Version Pool)을 관리하여 레코드의 안정된 버전(Stable Version)을 관리할 수 있는 기법을 제안한다. 판독트랜잭션은 안정된 버전을 찾기 위해 별도의 연산에 대한 부하없이 버전 풀에서 가장 먼저 생성된 버전을 읽어오기만 하면 된다. 또한 갱신 트랜잭션은 로크(Lock)을 사용하지 않고 레코드의 갱신을 데이터베이스에 곧바로 반영하며, 판독 트랜잭션도 판독연산에 로크를 사용하지 않고 버전 풀에서 관리하는 레코드의 안정된 버전을 선택하여 읽어 가는 기법을 제안한다.

1. 서 론

데이터베이스시스템에서 레코드를 동시에 갱신/판독 (write / read) 연산을 하는 트랜잭션들을 제어하는데 발생하는 문제를 해결하기 위하여 동시성 제어 알고리즘들이 많이 소개되어졌다. 이러한 알고리즘 중에서 멀티버전 알고리즘은 갱신 트랜잭션과 판독 트랜잭션 사이의 간섭을 최소화하여 시스템의 동시성을 향상시키기 위하여 사용되어진다.

멀티버전 데이터베이스시스템에서 버전(version)은 갱신 트랜잭션의 연산에 의해서 생성된 레코드의 복사본을 의미한다. 이렇게 생성되어진 다수의 버전에 대한 접근을 통해 트랜잭션간의 동시성제어의 효율을 높이는 것이다. 이러한 멀티버전 데이터베이스 시스템에서 동시성 제어를 위한 기법으로 멀티버전 타임스탬프(Timestamp) 기법과 멀티버전 2PL(Two Phase Locking) 기법이 있다.

멀티버전 타임스탬프 기법은 판독 트랜잭션의 판독 연산이 절대 취소(abort)되지 않는 장점을 지니고 있다. 하지만 판독 트랜잭션의 판독연산은 갱신 트랜잭션과 동기화를 시켜야 하며 레코드의 버전에 관련된 정보를 갱신해야 하는 부하를 지니고 있다. 로킹(Locking) 기법을 기반으로 하는 멀티버전 2PC 기법은 완료된 트랜잭션에 대한 별도의 정보를 유지하고 있어야 하며 판독 트랜잭션은 이 트랜잭션 리스트에서 안정된 버전을 비교 검색해야 하는 연산의 부하가 따른다. 또한 멀티버전 데이터베이스시스템은 레코드의 멀티버전에 대하여 최신 버전을 유지해야 하는 Garbage-Collection 연산을 수행하여야 한다. 따라서 데이터베이스 내에 다수의 버전을 유지하기 위한 별도의 저장공간과 최신버전에 대한 정보를 갱신하는 연산의 부하를 줄일 수 있는 방법이 필요하다.

본 논문[1]에서는 로크를 사용하지 않고 트랜잭션간의 효율적인 동시성제어를 위한 방법으로 버전을 유지하기 위한 저장공간의 최적화와 데이터베이스의 안정된 버전을 유지하는 방법을 제안한다. 기존의 멀티버전 알고리즘은 데이터베이스에 대한 갱신이 발생하면 데이터에 대한 새 버전을 생성하여 데이터베이스에서 관리하는 것과 달리, 본 논문에서 제안하는 기법은 트랜잭션에 Lock을 사용하지 않으며 레코드에 갱신이 발생하면 새로운 버전을 생성하지 않고 곧바로 데이터베이스에 반영을 시킨다. 대신 갱신 데이터에 대한 갱신 이전의 이미지 버전(Before-Image-Version)을 일시적으로 메모리 상의 버전 노드(Version Node)에 해당 트랜잭션별로 유지하고 여러 개의 그 버전 노드를 모아서 버전 풀(Version Pool)을 형성함으로서 버전을 위한 저장공간의 최적화와 효율성을 최대화시키는 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장 관련연구에서는 기존의 멀티버전 알고리즘을 이용한 동시성 제어로 멀티버전 타임스탬프 기법과 멀티버전 2PL(Two Phase Locking) 기법을 살펴본다. 3장에서는 본 논문에서 제안하는 일시적인 버전(Transient Version) 유지를 위한 버전노드(Version Node)와 버전 풀(Version Pool)에 대한 자료구조 기술하며, 레코드의 갱신 이전 이미지(Before-Image)를 이용한 일시적인 버전의 생성 및 버전의 제어 알고리즘을 기술한다. 마지막으로 4장에서는 결론을 기술한다.

2. 관련연구

멀티버전 데이터베이스시스템에서 갱신트랜잭션은 연산을 통해서 레코드의 복사본인 버전을 생성한다. 판독트랜잭션은 여러 개의 버전 중에서 안정된 버전이라고 판단되어지는 것을 선택하여 읽는다. 갱신트랜잭션이 레코드에 직접 갱신연산에 대한 반영을 하지 않고 새로운 버전을 생성함으로써 판독트랜잭션은 레코드를 판독하는데 유연성을 가질 수 있는 것이다. 멀티버전 알고리즘을 기반으로 하는 트랜잭션의 동시성제어 기법은 기본적으로 타임스탬프(Timestamp)기법과 2PL (Two-Phase Locking) 기법이 있다.

관련연구에서는 멀티버전 알고리즘을 기반으로 한 동시성제어 기법인 타임스탬프 기법과 2PL 기법에 대하여 기술한다.

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

2.1. 멀티버전 타임스탬프(Timestamp)

초기에 등장한 멀티버전을 이용한 동시성 제어기법이 트랜잭션에 타임스탬프를 이용한 알고리즘이다. 각각의 트랜잭션은 유일한 타임스탬프를 할당받는다. 타임스탬프는 트랜잭션이 시작한 시간을 의미하며, 각 트랜잭션은 서로 다른 타임스탬프를 가진다.

각 버전은 갱신트랜잭션의 타임스탬프에 의해서 각각 label을 가지게 된다. 즉, 트랜잭션 Ti가 Write(x)를 실행함으로써 x의 새로운 버전 x'을 만들고 이러한 버전은 타임스탬프로 label을 가진다.

타임스탬프를 이용한 연산의 알고리즘은 다음과 같다.

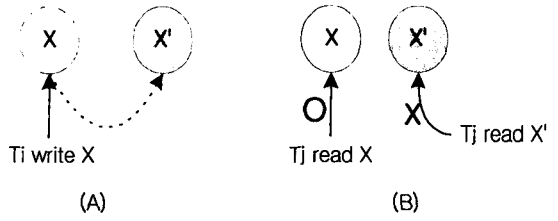
(1) $r_i[x]$는 $r_i[x_k]$로 전환되어진다. (x_k는 i 트랜잭션의 타임스탬프보다 작거나 같으면서 가장 큰 타임스탬프를 가진 x의 버전을 가리킨다.)

(2) $w_i[x]$는 다음 두 가지의 경우가 발생한다.

- 이미 $r_j[xk]$ (TS(k) < TS(i) < TS(j)) 가 진행되어졌다면 $w_i[x]$는 연산이 거부된다.
- 위의 경우가 아니라면, $w_i[x]$는 $w_i[x_i]$로 전환되어진다.

2.2. 멀티버전 2PL(Two Phase Locking) 기법

데이터베이스의 레코드 x 에 갱신로크를 걸게되면 x에 대한 판독로크를 얻는 것이 제한된다. 이러한 로크충돌문제를 레코드 x에 대한 버전을 사용함으로써 피할 수 있다. 트랜잭션 T_i가 x에 대한 새로운 버전 x_i를 생성하게 되면 다른 트랜잭션이 x_i를 읽거나 쓰는 것은 로크를 사용함으로써 막는다. 트랜잭션 T_i에 의해서 X' 버전이 생성되면 트랜잭션 T_j 는 X 레코드를 읽는 것을 허용함으로써 로크의 충돌없이 판독연산이 가능해진다.



(A) 갱신트랜잭션의 버전 생성
(B) 판독트랜잭션의 버전 접근 허용여부

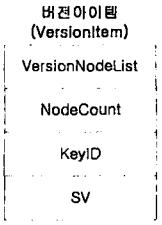
그림[2-1] 멀티버전 2PL 기법

3. 일시적인 버전(Transient Version) 제어 알고리즘

기존에 소개되어진 멀티버전 알고리즘은 갱신 트랜잭션에 의해서 갱신연산을 수행하면 갱신된 데이터에 대한 새로운 복사본을 생성하여 데이터에 대한 멀티버전을 유지하는 방식이었지만, 본 논문에서는 갱신트랜잭션이 데이터에 갱신 연산을 수행하면 갱신연산의 결과를 현재 데이터에 곧바로 반영시킨다. 갱신 연산의 결과를 곧바로 반영시키는 대신에 반영 전의 레코드 값을 버전 풀에 유지 및 관리를 한다. 이 장에서는 갱신 이전의 이미지를 메모리에 일시적인 버전으로 생성하여 관리하기 위한 자료구조와 버전 풀에 대한 관리구조를 소개한다.

3.1 자료구조

본 논문에서는 일시적인 버전을 메모리 상에서 버전 풀에 관리하는 방법을 제안한다. 갱신 트랜잭션에 의해서 생성된 레코드의 갱신 이전 이미지(Before-Image-Version : BIV)는 메모리에서 버전 풀에 저장되는데 버전 풀은 버전아이템으로 구성되어지는 버퍼이다. 버전 객체(레코드)에 대한 정보를 관리하는 버전아이템은 다음 그림[3-1]과 같다.

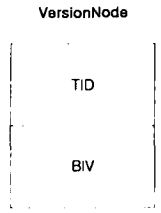


VersionNodeList	: 현재 버전 아이템에서 걸려있는 버전 노드 리스트
NodeCount	: 버전노드리스트에 존재하는 버전 노드의 개수
KeyID	: 레코드 ID, Hash Key
SV	: 안정된 버전(stable version)

그림[3.1] 잠금 아이템 자료구조

VersionNodeList는 현재 버전 아이템의 버전 객체에 대하여 갱신 연산을 수행하고 있는 버전노드들의 리스트를 연결리스트(Linked List)로 구성한 것이다. NodeCount는 현재 버전노드리스트(VersionNodeList)에 유지하고 있는 버전노드의 개수에 대한 정보를 저장한다. KeyID는 버전 객체에 대한 레코드ID 값이다. 버전아이템은 해쉬테이블로 구성되어 관리되어지는데 KeyID는 해쉬테이블의 탐색을 위한 Key값으로 사용되어진다.

버전노드(VersionNode)는 갱신 트랜잭션에 의해서 갱신될 데이터베이스의 레코드 갱신이전 이미지 버전과 해당 트랜잭션ID를 저장한다. 기존의 멀티버전 알고리즘이 갱신 이후의 버전을 생성하여 관리하는 것과는 달리 본 논문에서 제안하는 기법은 갱신 이전의 이미지를 관리함으로써 각 레코드에 대한 안정된 버전을 유지한다.

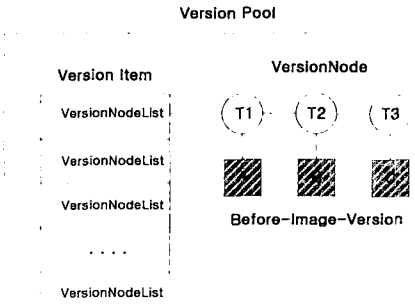


TID : 트랜잭션 아이디
BIV : 레코드의 갱신 이전의 이미지(Before Image Version)

그림[3.2] BIV을 포함한 버전노드

3.2 버전 풀(Version Pool)

레코드에 대한 갱신 이전 이미지 버전에 대한 정보를 유지하기 위해서 버전 아이템과 버전노드를 이용하여 버전 풀을 구성한다.



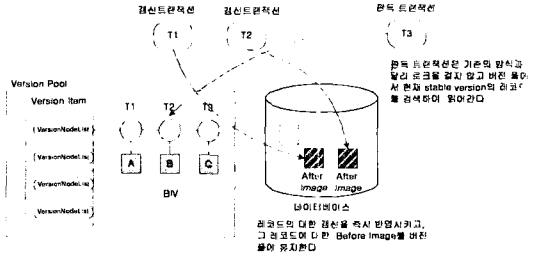
그림[3.3] 버전 풀 구성도

버전 풀은 각 레코드에 대해 생성된 버전에 대한 정보를 유지하기 위해서 버전 아이템을 배열로 구성을 하며 각각의 버전아이템은 버전 노드리스트를 이용하여 버전객체(레코드)에 갱신을 요구한 트랜잭션에 대한 정보를 버전노드에 저장하여 연결리스트로 연결되어진다.

레코드에 갱신연산을 수행하면 트랜잭션은 버전 풀에서 버전 아이템에 트랜잭션ID와 레코드의 갱신이전이미지버전을 저장한 버전노드를 버전노드리스트에 추가한다.

3.3 버전 제어 알고리즘

본 논문에서 제안하는 기법은 트랜잭션에 의한 레코드의 갱신에 대하여 기존의 로킹 프로토콜과 달리 갱신에 대해서 로크를 걸지 않고 트랜잭션 갱신연산의 시간지연 없이 데이터베이스에 곧바로 반영이 되어진다.



그림[3.4] 갱신 트랜잭션에 대한 버전 풀 관리

이 장에서는 각 트랜잭션의 연산에 버전노드와 버전 풀에 대한 변경사항을 살펴본다.

● 갱신트랜잭션에 의한 레코드 갱신

갱신 트랜잭션의 레코드에 대한 갱신연산은 다른 트랜잭션의 연산과는 상관없이 데이터베이스에 곧바로 반영되어지며 레코드의 버전을 관리하기 위해서 해당 레코드의 갱신 이전 이미지 버전을 버전 풀에 저장한다. 버전 풀에 대한 추가연산은 다음과 같다.

(1) 우선 해쉬테이블을 검색하여 버전 풀에서 레코드ID에 해당하는 버전아이템으로 이동을 한다.

(2) 버전 아이템에서 버전노드리스트에 레코드의 BIV를 버전노드를 이용하여 추가시킨다. 버전노드리스트는 연결리스트로 구성되어져 있으며 연결리스트의 마지막에 추가시키도록 한다.

(3) 버전 노드에는 트랜잭션ID와 현재 레코드의 갱신이전 이미지를 저장한다. 만약 현재 추가한 버전 노드가 버전 아이템의 첫 번째 버전 노드라면 버전 아이템의 SV를 BIV로 갱신시킨다.

갱신 트랜잭션에 의해서 버전 아이템에 저장되어지는 버전 노드 중에서 안정된 버전에 대한 값을 판단하는 것은 간단하다. 버전노드리스트에서 가장 첫 번째 연결리스트에 연결되어 있는 버전을 안정된 버전이라고 간주할 수 있다. 그 이유는 연결리스트에 이어지는 노드들은 트랜잭션의 연산 수행 시간 순서대로 연결되어지는 것이다. 그러므로 맨 처음 버전 노드의 BIV 값을 현재 레코드의 가장 안정된 버전이라고 판단할 수 있는 것이다. 그래서 갱신 트랜잭션에 의해서 생성되는 버전 노드들 중에서 가장 먼저 생성되어진 노드의 BIV을 버전 아이템에 저장해 두어서 판독 트랜잭션이 사용할 수 있도록 한다.

● 갱신 트랜잭션 커밋(commit)

갱신 트랜잭션이 정상적으로 연산수행을 끝마치고 커밋을 하면 버전 풀에서 별다른 추가연산 없이 버전아이템의 버전노드리스트에서 해당 버전 노드를 삭제하면 된다. 단, 삭제하고자 하는 버전 노드가 버전노드리스트의 첫 번째 노드라면 그 다음 노드의 BIV가 해당 버전 아이템의 SV로 갱신되어진다.

● 갱신 트랜잭션 취소(abort)

그림[3.3]에서 트랜잭션 T1이 수행도중 취소가 발생하면 T1의 BIV가 안정된 버전(Stable Version)이 되므로 T1의 BIV를 T2의 BIV로 복사하여 레코드의 안정된 버전을 유지한다.

● 판독 트랜잭션의 레코드 읽기

판독 트랜잭션은 데이터베이스의 레코드를 직접 접근하여 원하는 값을 얻어오는 것이 아니라 갱신 트랜잭션에 의해서 생성된 여러 개의 버전 중에서 안정된 버전으로 판단되어지는 것을 읽어오면 된다. 버전 풀에서 버전 아이템은 버전노드리스트 중에서 안정된 버전의 Before-Image를 가지고 있으므로 판독 트랜잭션은 그 값을 읽기만 하면 된다.

4. 결론

본 논문에서는 데이터베이스시스템에서 동시수행되는 트랜잭션에 대하여 로크기법을 사용하지 않고 트랜잭션간의 효율적인 동시성제어를 위한 방법으로 일시적인 버전관리 기법과, 버전을 유지하기 위한 저장공간의 최적화와 데이터베이스의 안정된 버전을 유지하는 방법을 제안하였다. 본 논문에서 제안하는 기법은 갱신 트랜잭션에 로크를 사용하지 않으며 레코드에 갱신이 발생하면 갱신된 이미지에 대한 새로운 버전을 생성하지 않고 곧바로 데이터베이스에 반영을 시킨다. 대신 갱신 데이터에 대한 갱신 이전의 이미지 버전(BIV)을 일시적으로 메모리 상의 버전 노드에 해당 트랜잭션별로 유지하고, 여러 개의 버전 노드를 모아서 버전 풀을 형성함으로써 데이터베이스에 레코드의 버전을 저장하고 낭비되는 저장공간에 대한 문제를 해결할 수 있다.

판독 트랜잭션도 레코드에 로크를 걸지 않으며, 안정된 버전을 찾기 위하여 버전에 대한 타임스탬프 등의 정보를 비교 할 필요 없이 버전 아이템에서 버전노드리스트 중에서 가장 먼저 생성되어진 것을 안정된 버전으로 간주하여 읽어오면 되므로 멀티버전 데이터베이스에서 최신의 버전을 유지하기 위한 별도의 연산이 필요 없다.

5. 관련논문

[1] PHILP A. BERNSTEIN and NATHAN GOODMAN " Multiversion Concurrency Control - Theory and Algorithm ", ACM 1983

[2] MICHAEL J.CAREY and WALEED A. MUHANNA " The Performance of Multiversion Concurrency Control Algorithm ", ACM 1986

[3] CHRISTOS H. PAPADIMITRIOU and PARIS C. KANELLAKIS " On Concurrency Control by Multiple Versions ", ACM 1984

[4] Divyakant Agrawal and Soumitra Sengupata "Modular Synchronization in Multiversion Database : Version Control and Concurrency Control", ACM 1989

[5] Baojing Lu, Qinghua Zou, William Perrizo " A Dual Copy Method for Transaction Separation with Multiversion Control for Read-only Transactions", ACM 2001

[6] C. Mohan, Hamid Pirahesh, Raymond Lorie "Efficient and Flexible for Transient Versioning of Record to Avoid Locking by Read-Only Transactions", ACM SIGMOD 1992

[7] D. Agrawal and V. Krishnaswamy "Using Multiversion Data for Non-interfering Execution of Write-only Transaction", ACM 1991

[8] Arvola Chan, Stephan Fox, Wen-Te. Lin " The Implementation of An Integrated Concurrency Control and Recovery Scheme", ACM 1982

[9] Mohan.. C, Levine. F, A"An Efficient and High Concurrency Index management Method Using Write-Ahead Logging" ACM SIGMOD 1992

[10] SILBERSCHATZ. A. " A Multi-version concurrency control scheme with no rollbacks ", ACM 1982.