공간 데이터베이스 관리 시스템을 위한 버전 기반의 공간 레코드 관리 기법

김희택[0*], 김명근*, 김호석*, 배해영*
*인하대학교 컴퓨터·정보공학과
{htkim[0], kimmkeun, hskim, hybae}@dblab.inha.ac.kr

Version Based Spatial Record Management Techniques for Spatial Database Management System

Hee-Taek Kim[0*], Myoung-Keun Kim*, Ho-Seok Kim*, Hae-Young Bae*
*Dept. of Computer Science & Engineering, Inha University

요 약

기존의 공간 데이터베이스 관리 시스템에서는 공간 데이터의 검색 연산이 주된 연산이었지만, 최근 공간 데이터베이스 관리 시스템에서는 이동 객체의 실시간 위치 갱신 및 추적 등 공간 데이터의 갱신 연산 또한 빈번하게 발생하고 있으며, 이에 트랜잭션간 동시성 향상의 필요성이 증가하고 있다. 기존의 일반적인 데이터베이스 관리 시스템에서는 트랜잭션의 동시성 문제를 해결하고 성능을 향상시키기 위해서 많은 기법들이 연구되었으며, 그 중에서 다중버전(Multi-Version) 알고리즘은 각 트랜잭션간의 상호간섭을 최소화 시키면서 동시성을 향상시키기 위한 알고리즘이다. 하지만 공간 데이터베이스 관리 시스템에 트랜잭션의 동시성 향상을 위하여 기존의 멀티버전 알고리즘을 적용할 경우, 공간 레코드의 속성 데이터만 변경되어도 공간 레코드 전체에 대한 버전을 저장해야 하기 때문에 저장 공간의 낭비가 발생한다.

본 논문에서는 공간 레코드에 대해 트랜잭션간의 동시성을 향상시키고, 공간 레코드 버전의 저장 공간 낭비를 줄이기 위한 방법으로 속성 데이터 버전과 공간 데이터 버전을 분리하여 생성, 관리하는 레코드 관리 기법을 제안한다. 본 기법은 검색 트랜잭션은 갱신 트랜잭션의 영향을 전혀 받지 않고 트랜잭션을 진행할 수 있으며, 갱신 연산 시 공간 레코드 전체의 버전을 생성하는 대신에 공간 레코드를 속성 데이터 버전과 공간 데이터 버전으로 분리하여 갱신된 데이터 버전만 생성, 관리하는 기법이다.

1. 서론

기존의 공간 데이터베이스 관리 시스템에서는 공간 데이터의 검색 연산이 주로 발생하였다. 그러나 최근 공간 데이터베이스 관리 시스템에서는 이동 객체의 실시간 위치 갱신 및 추적 등 공간 데이터의 빈번한 갱신과 검색 연산이 발생한다. 이에 공간 데이터베이스 관리 시스템에서의 검색 트랜잭션과 갱신 트랜잭션간 동시성 향상의 필요성이 증가하고 있다.

이미 텍스트(text) 데이터베이스 관리 시스템에서는 트랜잭션의 동시성에 관련된 문제를 해결하고 성능을 향상시키기 위해서 많은 기법들이 연구되었다. 그 중에서 다중버전 알고리즘은 검색 트랜잭션과 갱신 트랜잭션의 상호 간섭을 최소화 시키면서 동시성을 향상시키는 알고리즘이다. 이러한 다중버전의 동시성을 높이기 위한 기법으로는 다중버전 타임스탬프(Timestamp) 기법과 다중버전 2PL(Two Phase Locking) 기법이 있다.

다중버전 알고리즘에서 버전은 갱신 트랜잭션 연산에 의해서 생성된 레코드의 복사본을 의미한다. 하나의 레코드에 대해 여러 개의 버전이 존재하기 때문에 검색 트랜잭션과 갱신 트랜잭션은 각각 다른 버전을 사용할 수 있다. 결국 다수의 버전에 대한 접근을 통해 트랜잭션의 동시성을 향상시키는 것이다. 특히 검색 연산이 갱신 연산 때문에 대기하거나, 갱신 연산이 검색 연산 때문에 대기하는 문제점을 제거하였다.

그러나 기존에 연구 되었던 다중버전 알고리즘을 공간 데이터베이스 관리 시스템에 적용할 경우, 하나의 공간 레코드에 대해 여러 개의 버전이 생성되기 때문에 검색 트랜잭션과 갱신 트랜잭션은 각기 다른 버전으로 접근이 가능하다. 따라서 트랜잭션간의 동시성을 향상시킬 수 있다. 하지만 갱신 트랜잭션에 의해 생성된 레코드 전체의 버전을 저장하기 위해서는 저장 공간의 낭비가 발생한다.

본 논문[1]에서는 공간 레코드의 속성 데이터 버전과 공간 데이터 버전을 분리하여 생성, 관리하는 버전 기반의 공간 레코드 관리 기법을 제안하며, 본 기법은 공간 레코드에 대해 트랜잭션간의 동시성을 향상시키고, 공간 레코드 버전의 저장 공간 낭비를 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2장 관련 연구에서는 기존의 다중버전 알고리즘을 이용한 동시성 향상 기법을 살펴본다. 3장에서는 본 논문에서 제안하는 공간 데이터베이스 관리 시스템을 위한 버전 기반의 공간 레코드 관리 기법의 자료 구조를 기술하며, 속성 데이터 버전과 공간 데이터 버전을 구분하여 생성, 관리하는 알고리즘을 기술한다. 마지막으로 4장에서 결론 및 향후 연구를 기술한다.

2. 관련 연구

다중버전 알고리즘은 갱신 트랜잭션이 발생하였을 경우 새로운 버전을 생성하며 검색 트랜잭션은 이미 생성된 여러 개의 버전 중에서 자신에게 적합한 버전을 검색한다. 즉, 다중버전 알고리즘은 검색 트랜잭션과 갱신 트랜잭션이 접근하는 버전을 다르게 하여 트랜잭션의 동시성을 향상 시켰다.

이 장에서는 다중버전의 동시성 제어 기법인 타임스탬프 기법과 2PL 기법에 대하여 기술한다.

1) 본 연구는 정보통신부 지원 ITRC 프로그램의 지원을 받아 수행되었음

2.1. 다중버전 타임스탬프 기법

다중버전 타임스탬프 기법은 트랜잭션간의 동시성을 위해 타임스탬프를 이용한 다중버전 알고리즘이다. 각각의 트랜잭션은 트랜잭션이 시작한 시간을 의미하는 유일한 타임스탬프를 할당받는다.

검색 트랜잭션은 적합한 버전을 검색 후 버전의 검색 타임스탬프와 트랜잭션의 타임스탬프 중 큰 값을 다시 버전의 검색 타임스탬프에 설정한다. 갱신 트랜잭션은 현재 버전의 검색 타임스탬프가 트랜잭션의 타임스탬프보다 클 경우와 현재 검색하는 버전의 갱신 타임스탬프가 트랜잭션의 타임스탬프보다 클 경우에는 철회된다. 그렇지 않을 경우에는 새로운 버전을 생성하여 새로운 버전의 갱신 타임스탬프를 현재 트랜잭션의 타임스탬프로 설정한다.

2.2. 다중버전 2PL 기법

임의의 트랜잭션이 데이터베이스의 레코드(R)에 배타적 잠금(exclusive lock)을 획득하게 되면 다른 트랜잭션은 R에 대한 공용 잠금(shared lock)를 얻는 것이 제한된다. 이러한 잠금의 충돌 문제를 R에 대한 버전을 사용함으로써 피할 수 있다. 트랜잭션 T가 R에 대한 새로운 버전 R'를 생성하게 되면 다른 트랜잭션이 R'를 읽거나 쓰는 것을 잠금을 사용함으로써 막는다. 하지만 새로운 트랜잭션이 R를 읽는 것은 허용함으로써 잠금의 충돌 없이 검색 연산이 가능해진다. 따라서 버전을 사용함으로써 검색 연산이 배타적 잠금이 걸린 레코드 때문에 대기하거나, 갱신 연산이 공유 잠금이 걸린 레코드 때문에 대기하는 문제점을 제거하였다.

3. 버전 기반의 공간 레코드 관리 기법

이 장에서는 본 논문에서 제안하는 공간 데이터베이스 관리 시스템을 위한 버전 기반의 공간 레코드 관리 기법의 자료 구조를 기술하며, 각 연산 별로 속성 데이터 버전과 공간 데이터 버전을 구분하여 생성, 관리하는 알고리즘을 기술한다.

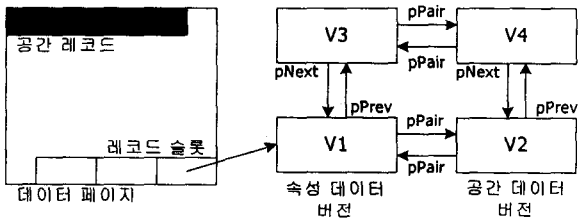
3.1 자료구조

공간 레코드의 속성 데이터 버전과 공간 데이터 버전의 구조는 [그림 3-1]과 같다.

pTxPrev	pPrev	pNext	pPair	pData	Timestamp	TxID

pTxPrev : 같은 트랜잭션에서 생성된 이전 버전을 가리키는 포인터
pPrev : 같은 레코드에서 생성된 이전 버전을 가리키는 포인터
pNext : 같은 레코드에서 생성된 다음 버전을 가리키는 포인터
pPair : 속성 데이터, 공간 데이터를 가리키는 포인터
pData : 데이터를 가리키는 포인터
Timestamp : 버전의 타임스탬프
TxID : 버전을 생성한 트랜잭션 ID

[그림 3-1] 버전 구조



[그림 3-2] 속성 데이터 버전과 공간 데이터 버전의 분리

레코드 페이지의 각 레코드 슬롯은 공간 레코드의 버전이 존재할 경우 속성 데이터 버전을 가리키고 있으며, [그림 3-2]는 본 논문에서 제안하는 공간 레코드의 속성 데이터와 공간 데이터를 분리하여 버전을 생성, 관리하는 전체적인 모습이다.

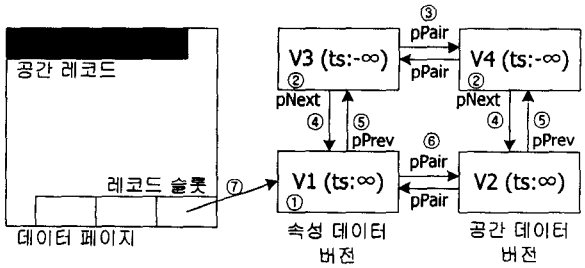
3.2 공간 레코드의 버전 관리 알고리즘

이 장에서는 트랜잭션의 각 연산별로 공간 레코드의 버전 관리 알고리즘에 대해 살펴본다.

3.2.1 삽입 연산

공간 레코드 삽입 연산은 데이터가 없는 버전과 삽입될 데이터를 갖는 버전을 생성하며, 각각의 버전을 연결한다. 데이터가 없는 버전은 레코드가 존재하지 않거나 지워진 것을 의미한다. 전체적인 방법은 [그림 3-3]과 같다.

1) 삽입될 공간 레코드의 속성 데이터 버전 V1과 공간 데이터 버전 V2를 생성한다. 생성된 각 버전의 타임스탬프는 초기값으로 ∞로 설정하며, 이는 이전에 발생한 모든 트랜잭션이 현재 버전을 읽을 수 없도록 한다.
2) 빈 데이터를 갖는 속성 데이터 버전 V3과 공간 데이터 버전 V4를 생성한다. V3과 V4의 타임스탬프는 -∞로 설정한다(현재 진행 중인 검색 트랜잭션의 최소 타임스탬프보다 작은 값).
3) V3과 V4는 pPair로 연결한다.
4) V3과 V4는 삽입 연산으로 생성된 버전을 pNext로 가리킨다.
5) 삽입 연산으로 생성된 버전은 각각 속성 데이터 버전 V3, 공간 데이터 버전 V4를 pPrev로 가리킨다.
6) 삽입 연산으로 생성된 속성 데이터 버전 V3는 V4를 pPair로, 공간 데이터 버전 V4는 속성 데이터 버전 V3를 pPair로 연결한다.
7) 마지막으로 레코드 슬롯에서 레코드 버전을 가리키는 포인터를 가장 최근에 생성된 속성 데이터 버전 V3를 가리키도록 한다.



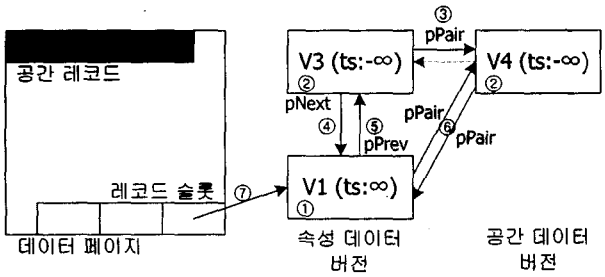
[그림 3-3] 공간 레코드의 삽입 연산

3.2.2 갱신 / 삭제 연산

공간 레코드 갱신 연산은 이전 레코드의 버전과 새로 생성된 레코드의 버전을 각각 연결한다. 전체적인 방법은 [그림 3-4]와 같다.

1) 갱신할 데이터가 속성 데이터이면 속성 데이터 버전 V1을, 공간 데이터이면 V2를, 속성 데이터와 공간 데이터일 경우 V1과 V2를 생성한다. 생성된 각 버전의 타임스탬프는 초기값으로 ∞로 설정한다.
2) 갱신될 공간 레코드가 버전이 없는 안정된(stable) 레코드일 경우 데이터 페이지의 속성 데이터를 저장한 속성 데이터 버전 V3과 데이터 페이지의 공간 데이터를 저장한 공간 데이터 버전 V4를 생성한다. V3과 V4의 타임스탬프는 -∞로 설정한다. 만약 공간 레코드의 버전이 존재할 경우에는 가장 최근에 생성된 속성 데이터 버전을 V3, 공간 데이터 버전은 V4로

설정한 후 단계 4번부터 수행한다.
3) V3과 V4를 pPair로 연결한다.
4) V3과 V4는 갱신 연산으로 생성된 버전을 pNext로 가리킨다.
5) 갱신 연산으로 생성된 버전은 최근에 생성된 속성 데이터 버전, 공간 데이터 버전을 pPrev로 가리킨다.
6) 갱신 연산으로 생성된 속성 데이터 버전은 가장 최근의 공간 데이터 버전을 pPair로, 공간 데이터 버전은 가장 최근의 속성 데이터 버전을 pPair로 연결한다.
7) 마지막으로 레코드 슬롯에서 레코드 버전을 가리키는 포인터를 가장 최근에 생성된 속성 데이터 버전을 가리키도록 한다.



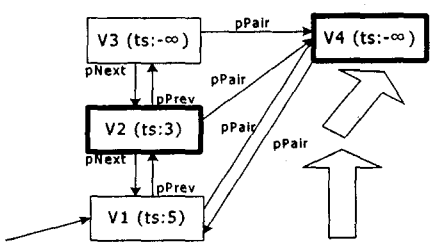
[그림 3-4] 공간 레코드의 속성 데이터만 갱신될 경우

삭제 연산은 공간 레코드를 데이터가 없는 속성 데이터 버전과 데이터가 없는 공간 데이터 버전으로 갱신하는 연산과 같다.

3.2.3 검색 연산

공간 레코드 검색 연산은 검색 연산의 타임스탬프와 버전의 타임스탬프를 비교하여, 적합한 버전을 검색한다. 전체적인 방법은 다음과 같다.

1) 검색할 레코드의 버전이 존재하지 않을 경우에는 데이터 페이지에서 공간 레코드를 검색한다.
2) 검색할 레코드의 버전이 존재할 경우에는 먼저 속성 데이터 버전을 검색한다. 검색하는 속성 데이터 버전을 현재 트랜잭션이 생성하였을 경우 현재 버전을 검색 후 pPair의 공간 데이터 버전을 조사한다. 또한 속성 데이터 버전의 타임스탬프가 검색 트랜잭션의 타임스탬프보다 작을 경우에는 현재 버전을 검색 후 pPair의 공간 데이터 버전을 조사한다. 그러나 현재 버전이 자신이 읽어야 할 타임스탬프와 맞지 않을 경우에는 pPrev가 가리키는 이전 버전을 조사한다. 자신에게 적합한 속성 데이터를 검색한 후 다시 pPair가 가리키는 공간 데이터를 검색한다.
3) 공간 데이터 버전의 검색 방법은 속성 데이터의 버전 검색 같은 방법으로 공간 데이터의 버전이 현재 트랜잭션으로 생성되었을 경우와 공간 레코드 버전의 타임스탬프가 검색 트랜잭션의 타임스탬프보다 작을 경우 검색할 수 있다.



[그림 3-5] ts가 4인 트랜잭션의 검색 연산

[그림 3-5]는 검색 트랜잭션의 타임스탬프가 4일 경우의 공간 레코드 검색 연산의 예이다. 검색 트랜잭션은 속성 데이터 버전에서는 타임스탬프가 3인 버전을 검색하며, 공간 데이터 버전에서는 타임스탬프가 -∞인 버전을 검색한다.

3.2.4 완료 / 철회 연산

완료 연산은 먼저 시스템으로부터 증가된 타임스탬프를 부여받은 후, 갱신 트랜잭션으로 생성된 모든 버전의 타임스탬프를 증가된 타임스탬프로 변경한다. 또한 갱신 트랜잭션 이전에 생성된 모든 버전을 쓰레기 수집기(Garbage Collector)에 추가한다. 본 논문에서는 쓰레기 수집기의 알고리즘에 대해 언급하지 않는다.

갱신 트랜잭션이 철회될 경우, 갱신 트랜잭션으로 생성된 모든 버전들을 갱신 트랜잭션 발생 이전의 버전으로 교체한다. 또한 갱신 트랜잭션으로 생성된 모든 버전을 삭제한다.

4. 결론 및 향후 연구

본 논문에서는 하나의 공간 레코드에 대한 버전을 여러 개 유지하기 때문에 검색 트랜잭션과 갱신 트랜잭션은 각기 다른 버전을 사용할 수 있어 검색 트랜잭션은 갱신 트랜잭션의 영향을 받지 않으며 검색 연산을 수행하였다.

또한 공간 레코드의 속성 데이터 버전과 공간 데이터의 버전을 따로 생성, 관리하기 때문에 크기가 큰 공간 레코드 전체의 버전을 저장하지 않고, 갱신된 데이터만 버전을 저장할 수 있다. 이로 인하여 공간 레코드 버전의 저장 공간 낭비를 줄일 수 있다.

향후 연구로는 공간 레코드의 속성 데이터 버전과 공간 데이터 버전으로 분리된 환경에서의 회복 알고리즘이 연구 되어야 한다.

5. 관련 논문

[1] Baojing Lu, Qinghua Zou, William Perrizo "A Dual Copy Method for Transaction Separation with Multiversion Control for Read-only Transactions", ACM 2001
[2] CHRISTOS H. PAPADIMITRIOU and PARIS C. KANELLAKIS "On Concurrency Control by Multiple Versions", ACM 1984
[3] C. Mohan, Hamid Pirahesh, Raymond Lorie "Efficient and Flexible for Transient Versioning of Record to Avoid Locking by Read-Only Transactions", ACM SIGMOD 1992
[4] D. Agrawal and V. Krishnaswamy "Using Multiversion Data for Non-interfering Execution of Write-only Transaction", ACM 1991
[5] Divyakant Agrawal and Soumitra Sengupata "Modular Synchronization in Multiversion Database : Version Control and Concurrency Control", ACM 1989
[6] MICHAEL J.CAREY and WALEED A. MUHANNA "The Performance of Multiversion Concurrency Control Algorithm", ACM 1986
[7] Mohan.. C, Levine. F, A "An Efficient and High Concurrency Index management Method Using Write-Ahead Logging" ACM SIGMOD 1992
[8] P.Bohannon, D.Lieuwen, R.Rastogi, S.Seshadri, A.Silberschatz, S.Sudarshan "Logical and physical versioning in main memory databases", VLDB 1997
[9] PHILP A. BERNSTEIN and NATHAN GOODMAN "Multiversion Concurrency Control - Theory and Algorithm", ACM 1983
[10] SILBERSCHATZ. A. "A Multi-version concurrency control scheme with no rollbacks", ACM 1982.