주기억 데이터베이스 시스템을 위한 병행수행 제어 프로토콜

심 종 익[†] · 배 해 영[††]

요 약

대부분의 주기억 데이터베이스 시스템에서는 병행수행 제어를 위하여 2단계 로킹 기법(2PL)을 사용하고 있다. 이 방법은 다른 병행수행 제어에 비해 단순하며 많이 사용되고 있는 장점이 있다. 그러나 기존의 병행수행 제어 방법은 데이터가 주기억 장치에 저장되어 있을 경우 적합하지 않을 수 있다. 본 논문에서는 주기억 데이터베이스 시스템 환경에 적합한 새로운 낙관적 병행수행 제어 프로토콜을 제안한다. 제안된 프로토콜은 검증 단계에서 충돌이 발견되면 이를 해결하기 위해 충돌횟수에 대한 정보를 이용한다. 주기억 데이터베이스 환경에서 2PL과 비교하여 트랜잭션의 처리율이 높아지는 결과를 얻었다.

Concurrency Control Protocol for Main Memory Database Systems

Jong-Ik Shim[†] · Hae-Young Bae[††]

ABSTRACT

Most of the main memory database systems use two-phase locking(2PL) for concurrency control. The 2PL method is preferred over other methods for concurrency control because of its simplicity and common usage. However, conventional concurrency control solution will function poorly when the data are memory resident. In this paper, we propose a new optimistic concurrency control protocol for a main memory database system. In our proposed protocol, transaction conflict information is used in validation phase to improve data conflict resolution decisions. Our experiments show that the proposed protocol performs better than 2PL in terms of throughput for main memory database system environments.

1. 서 론

최근 들어 통신 산업, 항공 우주 산업, 공정 제어 및 방위 산업 등 빠른 데이터 처리를 요구하는 분야가 확대되고 있다. 데이터의 고속 처리는 높은 트랜잭션 처리율(transaction rate)과 낮은 지연(latency)이 요구되는데 데이터가 디스크에 상주하는 기존의 데이터베이스 시스템은 디스크를 접근하는 동안 데이터의 양에 상관없이 높고 고정된 비용이 들어 이러한 응용분야에 적합하지 못하다. 반면, 모든 데이터가 주기억 장치에 저장되는 주기억 데이터베이스 시스템은 입출력으로 인한 지연시간이 거의 없어 기존의 데이터베이스 시스템보다 빠른 응답을 제공할 수 있다. 현재 집적 회로의 설계 기술이 빠르게 발전하여 칩의 밀도가 증가하고 반도체 메모리의 가격이 하락하여 주기억 장치의 용량은 점점 증가하는 추세이다. 따라

† 정 회 원:한서대학교 전산통계학과
†† 종신회원:인하대학교 전자계산공학과
논문접수:1996년 8월 27일, 심사완료:1996년 12월 4일

서 주기억 데이터베이스 시스템은 경제적으로 실현 가능하며 앞으로 일반적이 될 것이다[5, 7].

주기억 데이터베이스 관리 시스템에 관한 최근까지의 연구 결과 MM-DBMS, MARS, TPK, System M, Dali 등이 설계되거나 개발되었다[1, 4, 9, 11, 12, 16, 18]. MM-DBMS 시스템은 관계형 데이터 모델을 구현하고 병행수행 제어를 위해 2단계 로킹 기법을 사용하고 있으며 MARS는 로크 단위가 전체 릴레이션인 2단계 로킹 기법을 사용하고 있다. TPK는 멀티 프로세서 주기억 장치 트랜잭션 시스템으로 트랜잭션을 연속적으로 수행하여 병행수행 제어의 필요성을 없애버렸다. System M은 병행수행 제어를 위한 로크 서버가 있는데 2단계 로킹 기법을 사용한다. Dali는 주기억 저장 관리 시스템으로 병행수행 제어를 위해 2단계 로킹 기법을 사용하면서 새로운 로킹 모드를 적용할 수 있는 메커니즘을 제공하고 있다.

이와 같이 대부분의 주기억 데이터베이스 시스템은 병행수행 제어를 위해 기존의 2단계 로킹 기법을 그대로 채택하고 있다. 이는 2단계 로킹 기법이 기존의 데이터베이스 시스템에서 많이 연구되어 왔으며 상용 데이터베이스 시스템에서 널리 사용되고 있다는 장점 때문일 것이다. 뿐만 아니라, 주기억 데이터베이스 시스템에서는 그 특성상 백업 및 회복 기능을 가장 중요한 연구 과제로 삼고 있어 병행수행 제어 문제는 상대적으로 시스템의 성능 향상에 주요한 요인으로 생각되지 않고 있다.

주기억 장치는 디스크와 비교하여 몇 가지 특성을 지닌다. 첫째, 주기억 장치는 디스크에 비하여 접근 속도가 매우 빠르기 때문에 트랜잭션의 수행이 상대적으로 빠르게 완료될 수 있다. 따라서 주기억 장치 데이터베이스 시스템은 디스크 기반 시스템에 비해 트랜잭션간의 충돌이 훨씬 작게 된다. 둘째, 디스크가 일정한 크기의 데이터를 접근하는데 고정된 비용이 드는 반면, 주기억 장치는 접근 비용이 데이터의 크기에 비례한다. 셋째, 주기억 장치에서 하나의 페이지를 로킹하기 위한 시간은 하나의 페이지를 접근하는 시간과 비슷하다. 따라서 로크의 획득과 해지를 위한 동작이 디스크 기반 시스템과는 다르게 큰 오버헤드로 작용하게 된다. 이러한 특성들은 주기억 데이터베이스 시스템의 효율적인 병행 수행 제어를 위해 디스크 기반 시스템과는 다르게 처리되어야 함을 의미한다.

본 논문에서는 주기억 장치 특성에 적합한 병행수행 제어 프로토콜을 제안한다. 제안하는 프로토콜은 낙관적인 방법을 기초로 하였으며 충돌을 해결하기 위해 충돌한 트랜잭션들간의 충돌 횟수를 이용한다. 즉, 검증 단계(validation phase)에서 충돌이 발견되면 충돌을 많이 발생시키는 트랜잭션에 우선 순위를 주는 해결 방법을 사용한다. 이러한 방법은 긴 트랜잭션에 의해 데이터 충돌(data conflict)이 많이 발생하는 원인을 줄여 전체 시스템의 처리율(throughput)을 향상시킬 수 있다. 또한 낙관적인 기법에서 발생할 수 있는 기아(starvation)문제도 해결하게 한다.

본 논문의 구성은 다음과 같다. 2장에서는 주기억 데이터베이스 시스템의 병행수행 제어와 관련된 연구를 기술하고 3장에서 낙관적인 방식에 기초한 새로운 병행수행 제어 프로토콜에 대하여 기술한다. 4장에서는 성능 평가를 위하여 사용된 주기억 저장 관리 시스템의 특성을 기술하고 결과를 평가하였으며 5장에서 결론을 맺는다.

2. 병행수행 제어

2.1 주기억 DBMS에서의 로킹 기법

로킹 기법을 주기억 DBMS를 위한 병행수행 제어 기법으로 선택할 때 고려해야 할 점은 로크를 설정하고 해제하는 오버헤드이다. 디스크 기반 시스템에서는 로크를 설정하고 해제하는 연산이 디스크 접근 시간에 비해 매우 빠르게 수행되기 때문에 상대적으로 오버헤드가 적었다. 하지만 주기억 데이터베이스 시스템에서는 데이터 접근 시간이 디스크 기반 시스템에 비해 매우 빠르기 때문에 로크를 위한 오버헤드가 상대적으로 크게 된다. 이러한 문제는 로크에 대한 정보를 별도의 해쉬 테이블을 만들어 관리하지 않고 데이터에 로크 설정 비트를 두어 어느 정도 빠르게 처리될 수 있다. 하지만 2단계 로킹 기법은 직렬성에 의해 정의된 엄격한 데이터 일치성 요구를 만족시키기 위하여 충돌을 빈번하게 일으키며 충돌한 트랜잭션이 동일한 데이터 항목을 접근하면 접근이 허용되지 않게 하기 위해 블록킹을 사용한다.

주기억 데이터베이스 시스템을 위한 병행수행 제어에 관한 연구는 거의 이루어지지 않고 있으나 디스크를 기반으로 하거나 주기억 장치를 기반으로 한 실

시간 데이터베이스 시스템에서의 연구는 활발히 진행되어 왔다[2, 14, 8, 10, 15]. 일반 데이터베이스 시스템의 성능 평가 기준은 트랜잭션들의 평균 응답 시간이다. 하지만 실시간 데이터베이스 시스템은 종료시간(deadline)내에 완료된 트랜잭션들의 수로 평가하기 때문에 실시간 특성에 맞도록 병행수행 제어 기법의 보완이 필요하다[17]. 실시간 데이터베이스 시스템을 위해 2단계 로킹 기법을 기반으로 한 병행수행 제어 연구에는 [2, 14] 등이 있다.

2.2 주기억 DBMS에서의 낙관적인 기법

공유 데이터를 접근하는 트랜잭션들 사이의 충돌을 해결하기 위해 2단계 로킹 기법이 블록킹을 사용하는 반면 낙관적인 기법에서는 충돌한 트랜잭션을 취소(roll-back)시킨다. 낙관적인 병행수행 제어에서 트랜잭션은 읽기(read), 검증(validation), 그리고 쓰기(write)의 세 가지 단계로 구성된다[13]. 이중 검증단계가 트랜잭션의 상태가 결정되는 중요 단계이다. 검증 시험은 트랜잭션 Ti와 Tj의 연산사이에 충돌을 기초로 한다. 트랜잭션 Ti와 Tj사이에 데이터 충돌이 있는지 검사하는데는 전진 검증(foreward validation)과 후진 검증(backward validation)의 두 가지가 있다[6]. 만약 검증결과 규칙에 위배된 경우에는 충돌을 해결해야 한다.

후진 검증 기법에서의 데이터 충돌은 검증하고 있는 트랜잭션(validating transaction)의 읽기 세트(read set)와 완료된 트랜잭션의 쓰기 세트(write set)를 비교하여 검출한다. 즉, (그림 1)에서 검증할 트랜잭션 Tj의 읽기 세트와 이미 수행이 완료된 T1과 T2의 쓰기 세트를 가지고 검사한다. 충돌 해결 방법은 이미 완료된 트랜잭션이 직렬화 순서상 검증하고 있는 트랜잭션보다 앞서기 때문에 검증하고 있는 트랜잭션을 재시작하는 것이다.

전진 검증 기법에서는 트랜잭션의 검증이 현재 수행중인 트랜잭션들에 대해 이루어진다. 이것은 검증하고 있는 트랜잭션이 직렬화 순서상 읽기 단계에서 동시 수행중인 모든 트랜잭션들보다 앞선다는 가정에 기초한다. 따라서 데이터 충돌의 검출은 검증하고 있는 트랜잭션의 쓰기 세트와 활동중인(active) 트랜잭션의 읽기 세트를 비교하여 수행된다. 즉, (그림 1)에서 검증할 트랜잭션 Tj의 쓰기 세트와 동시에 수행중인 읽기 단계의 트랜잭션 active1과 active2의 읽기 세트를 가지고 검사하게 된다. 이러한 경우 데이터 충돌은 검증하고 있는 트랜잭션이나 혹은 읽기 단계에서 충돌한 트랜잭션들을 재시작함으로써 해결할 수 있다.

낙관적인 기법은 데이터 충돌이 발생되었을 경우 블록킹을 사용하지 않으면서 트랜잭션들을 병행적으로 수행하도록 한다. 또한, 교착상태가 발생하지 않기 때문에 2단계 로킹 기법과 같이 교착상태를 발견하기 위한 오버헤드가 없다.

디스크를 기반으로 하거나 주기억 장치를 기반으로 한 실시간 데이터베이스 시스템의 병행수행 제어 연구[8, 10, 15]는 데이터 충돌이 작을 경우 2단계 로킹 기법보다 낙관적인 병행수행 제어 프로토콜이 우수함을 보여주고 있다. 하지만 이러한 연구들은 모두가 종료시간과 트랜잭션의 중요도 등 실시간 트랜잭션에 관련된 여러 가지 요소들을 포함하는 정책을 갖고 있기 때문에 주기억 데이터베이스 시스템에 그대로 적용할 수는 없다.

(그림 1) 트랜잭션의 세 단계
(Fig. 1) The Three Phase of a Transaction

3. 새로운 낙관적 병행수행 제어

낙관적인 방법에서 후진 검증 기법은 재시작의 고려대상이 검증하고 있는 트랜잭션뿐인 반면 전진 검증 기법은 트랜잭션의 중요도에 따라 검증하고 있는 트랜잭션 혹은 충돌된 트랜잭션들을 선택적으로 재시작할 수 있어 더 유연성이 있다. 또한 전진 검증은 후진 검증에 비해 데이터 충돌을 더 일찍 검출할 수 있어 시간과 자원의 낭비를 줄일 수 있다. 일반적으로 트랜잭션의 읽기 세트는 쓰기 세트보다 훨씬 크다. 따라서 후진 검증은 오래된 쓰기 세트와 매우 큰 읽기 세트를 비교하게 된다. 반면에 전진 검증은 수행중인 트랜잭션의 읽기 세트와 작은 쓰기 세트만을

비교한다. 후진 검증은 오래된 쓰기 세트가 더 이상 필요가 없을 때까지 보관해야 하는 오버헤드를 갖는다. 그러나 전진 검증은 검증 과정 중 새로운 트랜잭션이 시작되는 것을 허락한다. 새로이 제안하는 주기억 데이터베이스를 위한 병행수행 제어 프로토콜은 전진 검증 기법에 기초로 하여 충돌 발생시 이를 해결하기 위해 트랜잭션들간의 충돌 횟수를 이용한다.

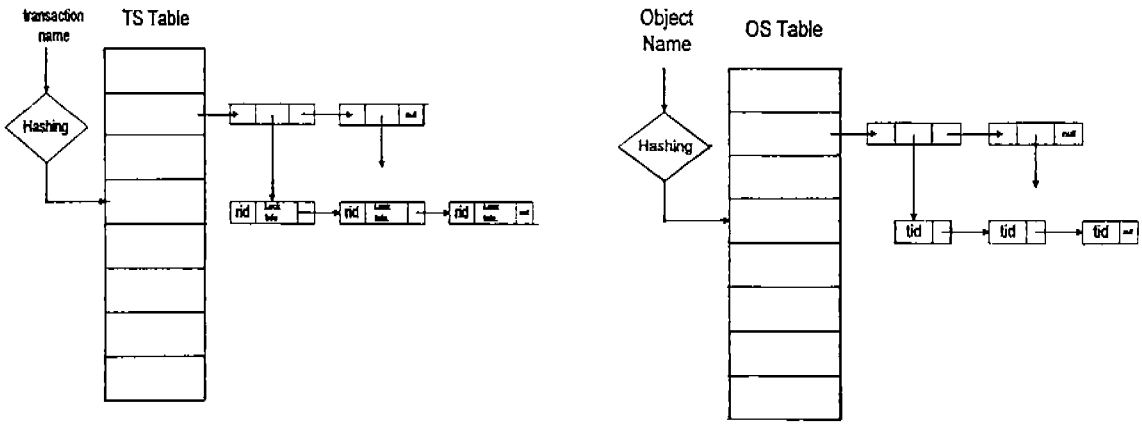
3.1 읽기 단계

제안하는 프로토콜은 낙관적인 기법을 기반으로 하고 있어 활동중인 모든 트랜잭션은 아무런 제약 없이 검증 단계에 도달할 때까지 병렬로 수행된다. 읽기 단계에서는 트랜잭션이 참조하고자 하는 데이터를 자신의 지역 작업영역에 복사하여 연산을 수행한다. 이 단계에서는 트랜잭션의 수행이 끝나 검증 단계에 도착하여 데이터 충돌을 검사하고 충돌이 발생되었을 경우 이것을 해결하기 위해 이용되는 데이터 오브젝트 세트(OS)와 트랜잭션 세트(TS)를 생성한다 (그림 2). 오브젝트 세트 OS(T)는 읽기 단계에 있는 트랜잭션 T가 쓰기 연산을 하는 데이터 오브젝트들의 집합이고 트랜잭션 세트 TS(Q)는 데이터 오브젝트 Q에 읽기 연산을 하는 활동중인 트랜잭션들의 집합이다.

제안하는 프로토콜은 실제 구현하기 위해 로킹 메커니즘[6]을 기초로 한 로크 테이블을 갖는다. 이것은 수행중인 모든 트랜잭션들이 공유할 수 있는 전역 테이블로써 읽기 단계-로크(R-lock)와 검증-단계 로크(V-lock)의 두 가지 모드를 갖게 된다. R-lock은 읽기 단계에 있는 트랜잭션에 의해 세트가 되며, V-lock은 검증 단계에 있는 트랜잭션에 의해 세트되게 된다.

3.2 검증 단계

트랜잭션이 읽기 단계가 끝나고 검증 단계에 도착하면 데이터 충돌이 있는지를 검사하게 된다. 검증하고 있는 트랜잭션 T_v가 쓰기 연산을 하는 오브젝트 세트 $OS(T_v)$내에 항목 Q를 가지고 있고, 이 항목 Q에 읽기 연산을 하는 활동중인 트랜잭션들의 집합 TS(Q)가 존재하면 충돌이 발생한 것이다. 즉, 검증하고 있는 트랜잭션 T_v와 충돌한 트랜잭션들을 T_c($c=1, 2, \ldots n, c \neq v$)라고 할 때, $OS(T_v)$에 쓰기 연산을 위해 사용된 데이터 오브젝트 Q가 존재하고, 그 데이터 오브젝트 Q에 읽기 연산을 하고 있는 트랜잭션의 집합 TS(Q)내에 T_c가 존재한다는 것이다. 만약 $OS(T_v)$내에 항목 Q가 존재하지 않으면 트랜잭션 T_v가 쓰기 연산을 하지 않고 있는 것이기 때문에 충돌이 발생하지 않아 쓰기 단계로 넘어가게 된다. 그러나 충돌을 발견하면 취소시킬 트랜잭션을 선택하여야 한다. 제안하는 프로토콜은 활동중인 다른 트랜잭션들과 충돌의 발생 가능성이 많은 트랜잭션을 우선적으로 완료하도록 한다. 검증하고 있는 트랜잭션 T_v의 다른 트랜



(그림 2) 병행수행 제어 테이블
(Fig. 2) Concurrency Control Table

잭션과의 충돌 횟수가 conflictNo(T_v)이고 충돌한 트랜잭션 T_c의 다른 트랜잭션과의 충돌 횟수가 conflictNo(T_c)라고 하자. 이때 conflictNo(T_v)가 conflictNo(T_c)보다 크면 검증 단계에 있는 T_v를 쓰기 단계로 보내어 완료가 되도록 하고 활동중인 충돌한 트랜잭션 T_c를 재시작 시킨다. 반대로 conflictNo(T_v)가 conflictNo(T_c) 보다 작으면 검증하고 있는 트랜잭션 T_v를 취소한다. 만약, 충돌횟수가 많아 긴 트랜잭션일 가능성이 높은 T_c가 취소된다면 재시작되므로 자원의 낭비가 심해지고 충돌의 발생 가능성이 더욱 높아지게 된다.

각 트랜잭션에 대한 충돌 횟수 conflictNo(T)의 계산은 읽기 단계에서 생성된 자신의 OS(T)와 TS(Q)의 정보를 이용한다. 먼저 트랜잭션 T가 쓰기 연산을 하는지 OS(T)에서 찾는다. 이때 OS(T)에 어떠한 데이터 오브젝트도 존재하지 않으면 쓰기 연산을 하지 않는 것이다. 따라서 다른 트랜잭션과 충돌이 없는 것이다. 이때, 트랜잭션 T가 검증하고 있는 트랜잭션일 경우에는 쓰기 단계로 넘어가 완료하게 한다. 만약 트랜잭션 T에 의해 쓰기 연산이 되는 데이터 오브젝트 Q_i(i = 1, 2, ... m)가 존재하면 Q_i에 읽기 연산을 하는 트랜잭션들이 존재하는지를 TS(Q_i)에서 찾는다. 이때, 모든 Q_i에 대한 TS(Q_i)의 수가 트랜잭션 T에 대한 충돌 횟수가 된다.

이렇게 계산된 충돌 횟수를 이용하는 방법은 활동중인 트랜잭션간의 데이터 충돌이 많이 발생되지 않는 방향으로 지속적인 제어를 하는 결과를 얻는다. 긴 트랜잭션은 상대적으로 다른 트랜잭션들과 데이터 충돌을 많이 발생시키기 때문에 검증 단계에 도착하면 우선적으로 완료시키도록 한다. 만약 긴 트랜잭션이 자주 취소되어 재시작된다면 시스템 전체의 데이터 충돌 수를 계속 높이게 되어 결국 시스템 전체의 트랜잭션 처리량은 떨어지게 된다. 짧은 트랜잭션은 데이터 충돌을 많이 일으키지 않기 때문에 비교적 다른 트랜잭션들과 충돌 없이 완료가 될 수 있다. 반면 충돌 횟수가 작은 트랜잭션은 재시작될 확률이 높으며, 반복적으로 취소되는 현상이 일어날 수 있다. 이것을 해결하기 위해 트랜잭션 T가 취소되어 재시작할 때에 재시작 횟수 RN[T]를 기록하게 된다. 재시작 횟수는 트랜잭션이 검증 단계에 도착하여 충돌 횟수를 비교할 때 반영된다. 따라서 재시작 횟수에 따라 트랜잭션의 완료될 가능성이 점차 높아지게 되어 이러한 기아 현상을 해결하게 된다.

제안된 병행수행 제어 프로토콜은 다음과 같다.

```
validate(Tv);
{
  valid := true;
  foreach Q ∈ OS(Tv) {
    foreach Ta ∈ TS(Q) {
      if conflictNo(Tv) + RN[Tv] < conflictNo(Ta) then {
        restart (Tv);
        adjust(Tv);
        valid := false;
        exit loop;
      }
    }
    if not valid then exit loop;
  }
  if not restarted Tv then {
    foreach Ta ∈ TS(Q) {
      restart (Ta);
      adjust(Ta);
    }
    execute write_phase(Tv);
  }
}
adjust(T);
{
  foreach Q ∈ OS(T) {
    if TS(Q) ∩ T ≠ { } then
      TS(Q) := TS(Q) − (TS(Q) ∩ T);
  }
  reset OS(T);
}
```

3.3 쓰기 단계

제안된 프로토콜에서는 검증 단계와 쓰기 단계를 하나의 임계 구역(critical section)에서 실행하도록 되어 있다. 트랜잭션이 검증 단계를 거쳐 쓰기 단계로 넘어오면 완료(commit)된 것으로 본다. 쓰기 단계에서 이루어지는 작업은 지역 작업영역에 있던 데이터

오브젝트들을 데이터베이스에 반영하는 일이다.

3.4 프로토콜의 정확성

프로토콜의 정확성은 제안된 프로토콜(OCC-CN)에 의해 생성된 모든 트랜잭션들의 실행결과(history)가 주기(cycle)를 갖지 않는 직렬성 그래프(serialization graph)로 표현되어 직렬성이 보장되는 것으로 증명한다[3]. H를 실행결과라 하고 SG(H)를 H에 대한 직렬성 그래프라고 하자. SG(H)는 정점이 종료된 트랜잭션을 나타내고 간선이 충돌한 트랜잭션들간의 연산 순서를 나타내는 방향 그래프이다. 만약 $T_i \rightarrow T_j (i \neq j)$이면 T_i의 연산이 T_j의 연산과 충돌함을 나타낸다. H가 직렬성을 갖고 있음을 증명하기 위해서는 SG(H)가 주기를 구성하지 않음을 증명하면 된다. 부등호 "<"은 연산들이 수행된 시간적인 순서를 의미한다. SG(H)에 간선 $T_i \rightarrow T_j$이 존재하면 다음과 같은 세 가지 종류의 충돌이 발생할 수 있다.

(1) $r_i[x] \rightarrow w_j[x]$

이것은 T_i의 읽기 $r_i[x]$가 T_j의 쓰기 $w_j[x]$에 영향을 미치지 않기 때문에 T_j가 검증 단계에 도착하기 전에 T_i가 완료되는 경우이다. OCC-CN에서 T_i가 검증 단계에 도착하였을 때 T_i가 x에 대한 쓰기 연산을 하지 않으면 OS(T_i)내에 데이터 항목 x가 존재하지 않으며 이러한 경우에는 트랜잭션 T_i가 완료가 되어 timestamp(T_i) < timestamp(T_j)가 된다.

(2) $w_i[x] \rightarrow r_j[x]$

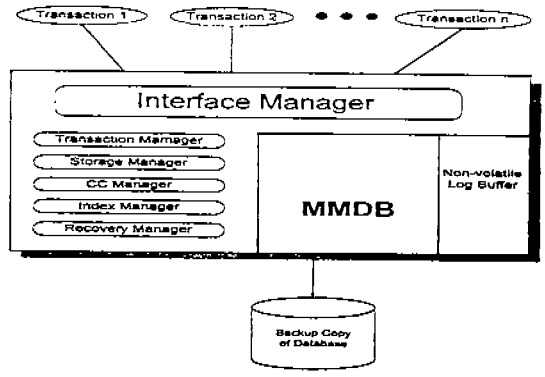
이것은 T_j가 읽기 단계에서 $r_j[x]$의 수행이 끝나기 전에 T_i의 쓰기 단계가 끝나는 경우이다. OCC-CN에서 검증하고 있는 트랜잭션 T_i의 conflictNo(T_i)가 충돌한 트랜잭션 T_j의 conflictNo(T_j)보다 크지 않으면 T_i는 항상 취소되기 때문에 반드시 timestamp(T_i) < timestamp(T_j)가 된다.

(3) $w_i[x] \rightarrow w_j[x]$

이것은 T_j가 검증 단계에 도착하기 전에 T_i가 완료되거나 T_j가 검증 단계에 도착한 후에 T_i가 완료되는 경우이다. OCC-CN에서는 검증 단계와 쓰기 단계가 동일한 임계 구역에서 수행이 되기 때문에 항상 timestamp(T_i) < timestamp(T_j)가 된다.

정리 : 제안된 OCC-CN 프로토콜에 의해 생성된 모든 실행결과는 직렬가능(serializable)하다.

증명 : 만약 SG(H)가 주기 $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n \rightarrow T_1$을 갖는다면 (1), (2), (3)에 의해서 timestamp(T_1) < timestamp(T_1)이 된다. 따라서 이것은 모순이 되어 SG(H)는 주기를 갖지 않는다.



(그림 3) 시스템 구조
(Fig. 3) System Architecture

4. 성능평가

이 장에서는 주기억 데이터베이스를 위한 병행수행 제어 프로토콜의 성능을 측정하기 위해 사용되는 시스템 구조와 실험 환경에 대하여 기술한다. 성능평가를 위하여 2PL과 제안한 병행수행 제어 프로토콜을 비교하였다.

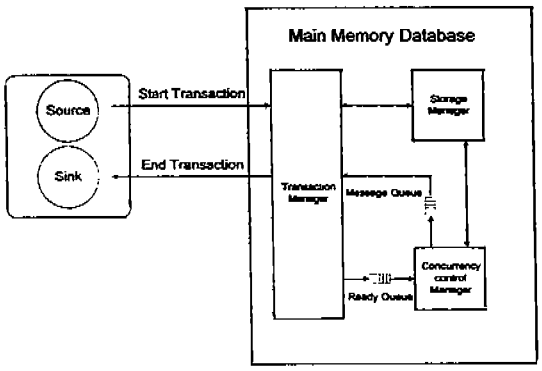
4.1 시스템 구조

(그림 3)은 UNIX OS하에서 구현된 주기억 저장관리 시스템이다. 데이터베이스는 공유 가상 메모리상에 저장되며 몇 개의 기본 구성 요소로 이루어져 있다. 주 메모리상의 자원을 관리하는 저장 관리자(Storage Manager)와 트랜잭션과 저장 관리자 사이의 서비스를 담당해 주는 트랜잭션 관리자(Transaction

Manager), 그리고 제안된 병행수행 제어 프로토콜이 구현되어 있는 병행수행 제어 관리자(Concurrency Control Manager)가 있다. 이외 회복 관리자(Recovery Manager)가 설계되어 있으나 아직 구현되어 있지는 않으며 데이터에 대한 빠른 접근을 위해 해쉬 인덱스를 제공하고 다중 사용자 환경을 제공하기 위해 공유 메모리를 사용한다.

성능 평가를 위하여 주메모리 저장 관리자를 이용한 클라이언트-서버(Client-Server) 모델을 (그림 4)와 같이 구성하였다. 이 모델은 크게 클라이언트 부분의 트랜잭션 생성기와 서버에 해당하는 데이터베이스 서버로 되어있다. 트랜잭션 생성기에 해당하는 소스(Source)는 트랜잭션의 생성을 모델링하는 여러 개의 클라이언트를 시뮬레이션하며 여러 개의 연산으로 이루어진 트랜잭션의 생성을 모델링한다. 싱크(Sink)는 완료된 트랜잭션에 대한 통계를 다루는 부분이다.

서버는 트랜잭션 관리자와 병행수행 제어 관리자 그리고 저장 관리자로 되어 있다. 클라이언트에서 생성되는 각 트랜잭션은 몇 개의 읽기와 쓰기 연산으로 구성되며 이는 주기억 장치에 저장되어 있는 데이터베이스중 임의의 데이터 항목에 대하여 이루어진다.



(그림 4) 시스템 모델
(Fig. 4) System Model

2PL과 OCC-CN에 대한 공정한 평가를 위해 각 트랜잭션에 구성된 읽기 혹은 쓰기 연산 과정은 수행이 완료될 때까지 두 가지 모델에서 동일하게 유지되도록 하였다.

〈표 1〉는 실험에 쓰여진 주요 파라메터들과 그 값을 요약한 것이다. 데이터베이스는 페이지들의 집합으로 구성하였고 최대 1000페이지로 하였다. 트랜잭션 길이(Transaction Length)는 트랜잭션에 의해 수행되는 연산의 수를 나타낸다. 이 값은 최대 길이(Max. Length)와 최소 길이(Min.Length)사이에 균등하게 주어지며 평균길이는 (Max.Length +Min.Length)/2이다. 트랜잭션의 모든 데이터 접근 연산중 쓰기 연산의 백분율인 쓰기 확률(Write Probability)은 0.2로 하였다. 또한 읽기 연산의 확률(Read Probability)은 0.8로 하였다.

〈표 1〉 실험 파라메터
〈Table 1〉 Experiment Parameters

Parameter	Meaning	Setting
DBSize	Database size in pages	1000 pages
TranLength	Transaction Length	5 steps
WriteProb	Write probability	0.2
ReadProb	Read probability	0.8
Min.Lengrh	Minimum Transaction Length	2 steps
Max.Length	Maximum Transaction Length	8 steps
MPL	Muliprogramming Level	10-100

4.2 평가 결과

병행수행 제어 프로토콜의 성능은 먼저 단위 시간당 완료된 트랜잭션의 수로 정의되는 시스템의 처리율(throughput)로 측정한다. 앞에서 정의된 정적 변수를 동일한 조건으로 놓고 다중 프로그래밍 단계(multiprogramming level:MPL)를 증가시키면서 분석하였다. MPL은 10에서부터 100까지 증가시키면서 성능을 측정하였다. 다음으로 측정한 것은 재시작율(restart rate)로서 데이터 충돌을 해결하기 위하여 취소된 트랜잭션의 비율로 표현되며 2PL의 경우에는 보류된 상태(blocked state)에 놓여 있는 트랜잭션의 비율인 블록킹율(blocking ratio)을 가지고 비교하였다.

먼저 두 가지 타입의 병행수행 제어에 대한 구현 오버헤드(implemenation overhead)를 생각해 보면, 두 가지 모두 물리적인 구현 방법이 거의 동일하기 때문에 비슷한 오버헤드를 갖는다. 먼저 데이터 접근에 대한 제어를 위해서 로킹 기법을 사용하고 있으며 두 가지 모두 해싱 연산과 로크 테이블을 통한 관리를 하고 있다. OCC-CN은 각 트랜잭션별로 쓰기와 읽기

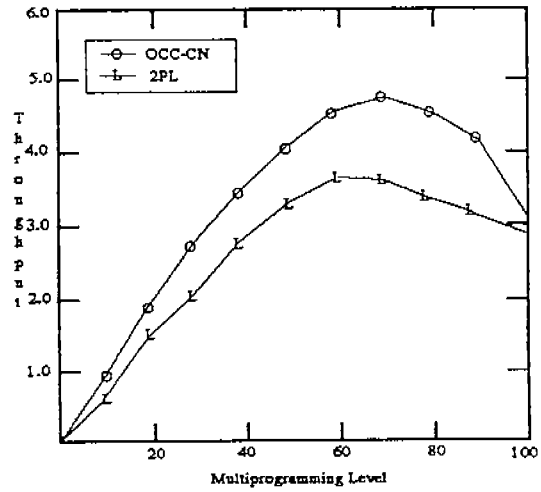
세트에 해당하는 OS와 TS에 정보를 유지하기 위해 오버헤드가 생기는 반면 2PL은 교착상태를 검출하고 해결하기 위해 추가 부담이 있다. OCC-CN은 지역 작업영역에서 연산이 이루어짐으로 메모리 사용 측면에서는 비용이 더 들어간다고 볼 수 있다. 따라서 두 가지 타입의 병행수행 제어에 대한 구현 오버헤드는 성능에 큰 영향을 미친다고 보지 않는다.

(그림 5)는 〈표 1〉의 파라메터의 값에 의해 시험한 결과이다. 이 그림에서 OCC-CN이 2PL보다 단위 시간당 처리량이 많다. 또한 MPL이 증가함에 따라 2PL이 임계점(critical point)에 먼저 도착하며 처리율이 더 이상 증가하지 않고 더 빨리 감소한다. 주기억 데이터베이스 환경으로 인하여 데이터 충돌이 디스크 기반 데이터베이스 환경에 비해 상당히 낮다. 이러한 환경에서 검증단계의 트랜잭션은 읽기 단계의 활동중인 다른 트랜잭션들과 대부분 충돌이 발생하지 않고 완료된다. 반면 2PL은 보류상태에 의한 지연 시간으로 인하여 단위 시간당 처리율이 OCC-CN보다 못하다.

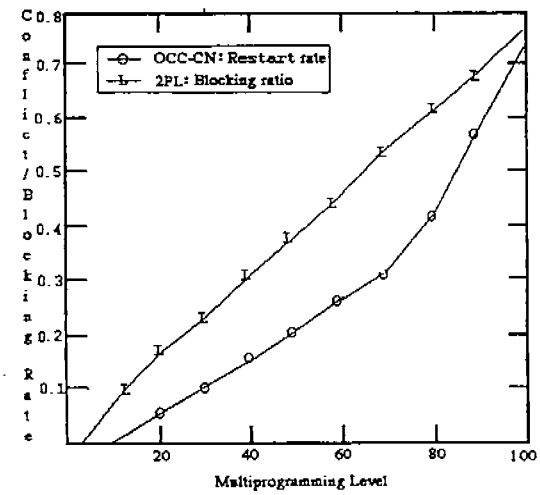
(그림 6)은 OCC-CN의 재시작율과 2PL의 블록킹율을 나타낸 결과 치이다. OCC-CN의 재시작율은 2PL의 블록킹율에 비해 매우 낮은 것을 알 수 있다. 물론 이것은 절대적인 비교치는 아니며, 두 가지 타입의 병행수행 제어 프로토콜의 행동 특성을 고려하여 분석해야 한다. 2PL에서는 보류상태의 지연시간과 교착상태를 검출하고 이를 발견하여 취소시킬 트랜잭션의 선택을 위한 오버헤드가 존재한다. 뿐만 아니라 보류 지연 시간(blocking delay time)을 예측하기가 힘들다. 반면 OCC-CN의 경우 재시작은 보류상태에 비해 일반적으로 오버헤드가 크다고 볼 수 있으나 짧은 트랜잭션들을 재시작시켜 오버헤드를 줄인다.

두 가지 모두 로크기법을 사용하고는 있지만 OCC-CN에서 사용하는 V-lock은 트랜잭션 수행중 맨 마지막 단계에서 발생되며 로킹 기간은 검증 단계와 쓰기 단계 동안이다. 반면 2PL에서는 트랜잭션이 갱신을 위하여 데이터를 접근하려고 할 때마다 쓰기 로크가 발생되며 로킹 기간은 트랜잭션의 살아있는 동안이다. 또한 이 프로토콜에서 사용하는 R-lock은 읽기 단계동안 어떠한 트랜잭션들도 블록 되지 않는 반면 2PL하에서는 읽기와 쓰기 로크 사이의 충돌은 충돌한 트랜잭션들을 블록 시키게 된다.

평가를 위해 트랜잭션은 짧은 트랜잭션(short transaction)과 긴 트랜잭션(long transaction)을 균등하게 발생하였다. 만약 충돌이 발생되어 긴 트랜잭션이 취소된다면 그 트랜잭션의 재시작으로 인해 발생되는 오버헤드가 크며 또한 연산의 수가 작은 짧은 트랜잭션에 비해 지속적인 충돌을 유발시킬 확률이 많아진다. 따라서 제안한 프로토콜은 데이터 충돌을 많이



(그림 5) 시스템 처리율
(Fig. 5) System Throughput



(그림 6) 트랜잭션 충돌율
(Fig. 6) Transaction Conflict Ratio

발생시키는 상대적으로 긴 트랜잭션을 우선적으로 처리함으로써 충돌을 완화시키는 효과가 나타나며 결국 시스템의 처리율을 높이는 것으로 분석되고 좀 더 넓은 연산 범위를 갖는다고 볼 수 있다.

5. 결 론

본 논문은 주기억 데이터베이스 시스템을 위해 낙관적인 기법에 기초한 새로운 병행수행 제어 프로토콜을 제안하였다. 이 방법은 트랜잭션들간의 충돌 횟수를 충돌을 해결하기 위한 정보로 이용한다. 낙관적인 기법은 트랜잭션의 검증단계에서 처리비용이 다소 높지만 비교적 데이터 충돌에 대한 부담이 적은 주기억 데이터베이스 시스템 환경에서는 유리하다. 시스템 전체의 처리율을 높이기 위해 트랜잭션의 검증 단계에서 충돌 횟수를 고려하였으며 편중된 트랜잭션 재시작 성향을 보완하기 위하여 재시작 횟수를 고려한 프로토콜을 제안하였다. 제안된 새로운 프로토콜은 주기억 장치 환경에 알맞은 조건으로 2PL과 성능을 비교 평가하여 시스템의 전체 처리량이 높아지는 결과를 얻었다.

향후 연구과제는 적절한 로크의 단위를 위해 다양한 조건에서 성능을 평가하는 작업이다. 로크의 단위가 커지면 충돌이 발생할 가능성이 높아지지만 로크로 인한 오버헤드가 증가된다. 반대로 로크의 단위를 작게 하면 충돌은 줄어들지만 로킹 연산에 대한 오버헤드가 상대적으로 크게 된다. 따라서 적절한 로크의 단위를 설정하는 것이 시스템의 성능을 높이는 요인이 된다.

참 고 문 헌

[1] A. C. Ammann, M. B. Hanrahan, and R. Krishnamurthy, "Design of a memory resident DBMS," in Proc. IEEE COMPCOM Conf., pp. 54-57, 1985.

[2] Agrwal, D., A. E. Abbadi and R. Jeffers, "Using Delayed Commitment in Locking Protocols for Real-Time Databases," Proc. of ACM-SIGMOD Int'l Conf. on Management of Data, pp. 104-113, 1992.

[3] Bernstein, P. A., V. Hadzilacos, and N. Goodman, "Conncurrency Control and Recovery in Database Systems," Addison-Wesley, 1987.

[4] D. Gawlick and D. Kinkade, "Varieties of concurrency control in IMS/VS Fast Path," Data Eng. Bull., Vol. 8, No. 2, pp. 3-10, June 1985.

[5] D. J. Dewitt et al., "Implementation Techniques for Main Memeory Database Systems," Proc. ACM SIGMOD Conf., pp. 1-8, June. 1984.

[6] Harder, T., "Observations on Optimistic Concurrency Control Schemes," Information Systems, Vol. 9, No. 2, 1984.

[7] Hector Garcia-Molina and K. Salem, "Main Memory Database Systems:An Overview," Transaction on Knowledge and Data Engineering, IEEE, Vol. 4, No. 6, pp. 509-516, 1992.

[8] Huang, J., J. A. Stankovic, K. Ramamritham, and D. Towley, "Experimental Evaluation of Real-time Optimistic Concurrency Control Schemes", Proceedings of the 17th VLDB Conf., 1991.

[9] H. V. Jagadish, Daniel Lieuwen, Rajeev Rastogi, and Avi Silberschatz, "Dali:A High Performance Main Memory Storage Manager," Proc. of the 20th VLDB Conf., pp. 48-59, 1994.

[10] Jayant R. Haritsa, Michael J. Carey, and Miron Livny, "Dynamic Real-Time Optimistic Concurrency Control," 11th IEEE Real-Time Systems Symposium, pp. 94-103, 1990.

[11] K. Li and J. F. Naughton, "Multiprocessor main memory transaction processing," Proc. Int'l. Symp. on Databases in Parallel and Distributed Systems, Austin. TX, pp. 177-189, 1988.

[12] K. Salem and H. Garcia-Molina, "System M:A Transaction Processing Testbed for Memory Resident Data," IEEE Transactions on Knowledge and Data Engineering, Vol. 2, pp. 161-172, Mar. 1990.

[13] Kung, H. T. and J. T. Robinson, "On Optimistic Methods for Concurrency Control," ACM Transactions on Database Systems, Vol. 6, No. 2, pp. 213-226, 1981.

[14] Lee, J., S. H. Son, "Using Dynamic Adjustment

of Serialization Order for Real-time Database Systems," Proc. of 14th IEEE Real-time Systems Symposium, pp. 66-75, 1993.

[15] Le Gruenwald, Sichen Liu, "A Performance Study of Concurrency Control in a Real-Time Main Memroy Database System," SIGMOD Record, Vol. 22, No. 4, pp. 38-44, 1993.

[16] Margaret H., Eich, "MARS:The Desgin of A Main Memory Database Machine," Proc. of the 1987 International workshop on Database Machines, Oct., 1987.

[17] M. Singhal, "Issues and Approaches to Design of Real Time Database Systems," SIGMOD RECORD, ACM, Vol. 17, No. 1, pp. 19-33, 1988.

[18] T. J. Lehman, "Design and Performance Evaluation of a Main Memory Relational Database System," Ph.D. thesis, University of Wisconsin, Madison, 1986.

심 종 익

1985년 인하대학교 전자계산학과 졸업(이학사)
1987년 인하대학교 대학원 전자계산학과 졸업(이학석사)
1994년~현재 인하대학교 대학원 전자계산공학과 박사과정
1986년~1987년 금성통신(현 LG전자) 연구소 연구원
1988년~1993년 LG산전 연구소 선임연구원
1994년~현재 한서대학교 전산통계학과 전임강사
관심분야: 데이터베이스(특히, 실시간 데이터베이스 시스템, 멀티미디어 데이터베이스, 주기억 데이터베이스 시스템)

배 해 영

1974년 인하대학교 응용물리학과(공학사)
1978년 연세대학교 대학원 전자계산학과(공학석사)
1989년 숭실대학교 대학원 전자계산학과(공학박사)
1985년 Univ. of Houston 객원교수
1992년~1994년 인하대학교 전자계산소 소장
1982년~현재 인하대학교 전자계산공학과 교수