論 文

지연된 블록킹 방법을 사용한 동시성 제어 기법의 성능 분석에 관한 연구

The Study for Performance Analysis of Concurrency Control using Deferred Blocking

남 태 희*, 박 재 운**, 위 승 민***
Tae-Hee Nam, Jae-Woon Park, Seung-Min Wee
(동주여자전문대학 무역사무자동화과*/동래여자전문대학 사무자동화과**
/한국해양대학교 해사수송과학과***)

◀ 차 례 ▶

요 약

동시성 제어 기법은 트랜잭션 처리 시스템의 성능에 중요한 영향을 미친다. 전통적인 록킹 기법은 대기 트랜잭션이 로크를 가지고 있으면서 진행 중인 다른 트랜잭션을 블록시키는 블록킹 현상을 일으킨다. 제안된 방법은 트랜잭션 실행의 다음 단계에서 그들의 블록킹 형태를

지연함에 의해서 블록킹 확률을 감소시킨다. 트랜잭션 실행은 전통적인 록킹같은 블록킹 상태와 트랜잭션이 로크를 기다리지만 다른 트랜잭션은 블록화하지 않는 비블록킹 상태로 나누어 볼 수 있다. 그러나 비블록킹 상태 중에 처리된 데이터는 트랜잭션을 취소 시킬 수 있다. 블록킹과 취소 효과를 적절히 균형을 이루어서 제안된 방식은 모든 데이터 함유와 자원 함유 수준에서 전통적 록킹과 OCC방식보다 더 뛰어난 성능을 가지고 있다.

Abstract

The concurrency control can be critical to the performance of transaction processing systems. Conventional locking takes the blocking phenomenon, where waiting transactions continue to hold locks and block other transactions from progressing. The proposed scheme reduces the blocking probability by deferring the blocking behavior of transactions to the later stages of their execution. The transaction execution can be divided into a nonblocking phase where transactions wait for locks but do not block other transactions and a blocking phase as in conventional locking. However data accessed during the nonblocking phase can lead to transaction abort. By properly balancing the blocking and abort effects, the proposed scheme can lead to better performance than either the conventional locking or the optimistic concurrency control(OCC) scheme at all data and resource contention levels.

I. 서 론

데이터 베이스 시스템에서 동시성 제어 방법으로 잘 알려진 것에는 록킹 방법과 OCC(Optimistic Concurrency Control)법이 잘 알려져 있다.[1] OCC법에서 만약 트랜잭션이 완료되기 전의 트랜잭션과 충돌한다면 그것은 취소된다. 그래서 높은 수준의 OCC 데이터 함유는 트랜잭션이 첫번째 실행된 후에 자원의 추가적인 소비를 가져오기 때문에 취소 확률이 더 높다. 그 결과로 OCC하의 처리 율은 자원에 제약을 받는다. 록킹에서는 다른 트랜잭션에 의해 점유된 로크와 충돌한다면 트랜잭션은 기다린다. 높은 데이터 함유 수준에서 이 로크의 대기 시간은 급격히 증가하고 궁극적으로 처리 율을 제한하게 된다. 이러한 데이터 함유 수준을 줄이려는 노력이 시도되고 있다. 이러한 노력의 일환으로 버퍼 지연 효과를 이용한다. 이것은 처음 실행할 때 자료를 버퍼에 저장해서 그 다음에 I/O를 하지 않는 방법이다.

다른 변수는 재실행을 하는 중에 이미 알고 있는 2개의 조합이다. 이 방법의 비교는 버퍼 사용 비율이나 재실행 중에 자료 재사용 비율은 거의 같다. 만약 재사용 비율이 높다면 순수

1) 국내 참고 문헌 1, 2, 3 참조.

OCC는 자료 함유 수준이 높을 경우에 이미 알려져 있는 다른 OCC 보다 성능이 좋다. 이러한 취소와 대기 방법을 조합하는 다양한 방법들이 있다.

OCC는 다른 형식의 동시성 제어 기법보다 시스템이 자원 제한에 걸리지 않도록 자료 함유 수준이 좋을 때 뛰어난 성능을 갖고 있다. 록킹은 자원 제한에 걸리거나 자료 함유 수준이 낮을 때 더 좋은 성능을 갖는 방식이다. 나머지 방식은 그 중간 단계에서 어느 정도 좋은 성능을 나타낸다. 하지만 이것 중 어떤 것도 자료 함유와 자원 제약 조건의 모든 수준에서 록킹과 OCC보다 지속적으로 더 우수한 방법은 없다.

그래서 제안된 방법으로 기존의 동시성 제어 기법보다 더 우수한 방법을 찾아보자. 본 논문에서는 지연된 블록킹 방법을 사용한다. 이 방법에서 트랜잭션의 블록킹 형태는 그 실행을 다음 단계로 지연시킴으로 해서 블록킹 확률을 감소시키는 방법을 사용한다.

트랜잭션을 실행시킬 때 비블록킹 상태에서 트랜잭션은 로크를 기다리지만 다른 트랜잭션을 블록 시키지 않는다. 반면에 블록킹 상태에서 트랜잭션은 일반적 록킹과같이 작동한다. 비블록킹 상태에 액세스된 데이터는 트랜잭션을 취소시킨다. 이점에서 OCC와 비슷하지만 OCC하에서 트랜잭션은 취소되거나 취소 마크가 된다. 이 방법의 트랜잭션은 비블록킹 상태중에 확실히 로크를 기다리기 때문에 검증 하에 있는 처리 자료들을 회피함으로써 취소 확률을 감소시키다. 먼저 실행상의 비블록 단계에서 블록킹 단계까지 서로 다른 변환점을 가지고 이 기법의 성능을 실험해서 최적의 작동점을 선택한다. 모델링은 하드웨어 자원 함유와 자료 함유[2]와 같은 2개의 상호작용 요소를 분석해서 얻는다.

자원 함유 효과는 큐잉 모델을 사용하여 추정하고 자료 함유는 트랜잭션 대기 또는 취소의 결과를 이용한다. 트랜잭션 취소는 트랜잭션 재실행을 위한 추가적인 자원 소모로 귀착되고, 계속되는 높은 자원 함유는 높은 자료 함유를 초래한다. 자료 함유와 자원 함유 사이에 그러한 상호작용은 반복을 통해서 값을 얻는다. 분석은 비블록 단계의 실행에서 블록 단계의 실행까지 전환되는 최적점을 결정한다.

동시성 제어 기법의 성능에 대한 다양한 분석 연구가 있는데 집중 DB(데이터 베이스)에서 록킹의 분석 조사와 분산 DB에서 서로 다른 동시성 제어 기법의 성능에 대한 분석 작업, 분산 DB 환경에서 최적의 복사본 개수에 관한 분석[3]등이 있다.

II. 제안된 기법과 다른 방법

1. 록킹 기법

데이터베이스 시스템에서의 동시성 문제는 많은 사람들에 의해 고찰되었다. 동시성은 다른 사용자들이 동시에 데이터베이스에 접근한다는 것을 의미한다. 그러한 시스템에서는 각각의 사용자들은 다른 사용자들에 대해 보호되어야 한다. 우리는 데이터 베이스에서 사용자가 트

2) 외국 참고 문헌 10, 12 참조.
3) 외국 참고문헌 3 참조.

랜잭션를 읽는 동안 다른 사용자가 그것을 변경하는 것을 피해야 한다. 사용자는 자료를 읽거나 변경하기 전에 트랜잭션를 로크시켜야 한다. 트랜잭션에서 쓰기 로크는 비개방적이다. 즉 다른 사용자가 트랜잭션를 변경하거나 읽으려면 로크가 해지될 때까지 기다려야 한다. 록킹의 또다른 이유로는 무결성(integrity)에 있다. 사용자는 데이터베이스가 올바른 상태에 있는 것을 원하고 그런 상태에서 벗어날 때 무결성을 회복해야 한다. 어쨌든 사용자는 일시적으로 무결성 침해가 있을 수 있는데, 다른 사용자들에게 보여져서는 안될 것이고, 이것은 로크를 걸어서 행해진다.

록킹은 물론 지연을 야기할 수도 있다. 우리의 목적은 이 지연을 감소시키는 것이다. 읽기 접근은 항상 허가되며 여러명의 Reader와 한명의 Writer는 동시에 같은 트랜잭션에서 작업할 수 있다.

2. Optimistic CC 기법

Optimistic 동시성 제어 방법은 처리 단계를 3단계로 나누어 그 단계를 거치면서 직렬성에 위배되는지를 검사하고 트랜잭션 수를 선정함으로서 처리되는 것들끼리의 충돌을 피하고 동시 제어를 제공하는 방법이다. 우리는 이러한 방법에서 중점적으로 검증 기법과 트랜잭션 수의 선정에 중점을 두어야 한다.

이 방법의 3단계는 다음과 같이 처리된다. 먼저 읽기 단계 동안에는 처리 t가 실행되면 여러 가지 데이터 항목을 읽고 새로운 값을 t에 국한된 변수에 저장한다. 모든 쓰기 연산은 실제 데이터베이스를 갱신하지 않고 임시 지역 변수에서 실행된다. 검증 단계는 처리 t가 직렬성을 위반하지 않고 임시 지역 변수의 값을 데이터베스에 복사할 수 있는지 결정하기 위해서 검증을 수행한다. 쓰기 단계는 처리 t가 검증에 성공하면 변경된 데이터 항목의 값들이 실재로 데이터베이스에 반영된다. 검증에 실패하면 읽기 단계로 복귀된다.

3. 제안된 방법

지연된 블록킹을 갖는 제안된 록킹 기법을 검사해 보면 일반적 록킹에서 동시성 제어 매니저는 액세스된 원소의 로크 테이블을 갖고 있다. 그 원소는 공유 모드와 배타적 모드로 나뉜다. 동일한 원소상에 있는 공유로크는 호환적이고 배타적 로크는 동일 원소상에서 호환되지 않는다. 동일하게 이 방법은 동시성 제어 매니저가 각 원소들이 공유 모드나 배타적 모드로 로크 될 수 있는 로크 테이블을 갖는다. 더욱이 약성과 강성의 두 가지 로크 형식을 소개하고 이들의 호환성은 표1과 같다.

강성 로크 요구와 약성 로크 점유 사이의 로크 충돌을 해결하기 위하여 새로운 개념을 도입하자 강성 로크 요구는 만약 요구된 자료 원소가 현재 약성 로크에 의해 점유되어 있다면 비호환적인 약성 로크 요구를 대신하고 비호환적인 약성 로크는 강성 로크 점유자를 취소 마크가 되도록 한다. 약성 로크 요구는 다른 약성 로크와는 항상 호환적이다.

[표 1] 호환성 테이블

요구자 \ 점유자		강성 로크		약성 로크	
		공 유	배 타	공 유	배 타
약성 로크	공 유	호 환	호 환	호 환	비호환
	배 타	호 환	호 환	비호환	비호환
강성 로크	공 유	호 환	비호환,대체	호환	비호환
	배 타	비호환,대체	비호환,대체	비호환	비호환

공유 모드에 있는 약성 로크와 강성 로크는 호환적이다. 반면에 약성 로크 요구는 강성 로크 점유자에 대해 호환적이지 않고 대기 상태에 놓이게 된다. 지연된 갱신 정책은 트랜잭션이 검증된 후에 완료 시간에 데이터베이스 버퍼에 갱신이 반영된 곳에서 사용된다. 지연된 갱신 정책하에서 실행 트랜잭션에 의한 중간 갱신은 다른 트랜잭션에 영향을 미치지 않는다.

그 원소가 해제되었을 때 만약 비호환적인 강성로크가 그 원소에 걸려 있지 않다면 그 원소는 약성 로크를 받게 되고 그 트랜잭션은 계속 진행된다.그러나 OCC방법에서 트랜잭션은 완료 점에 도달하기 전에 다른 완료 트랜잭션에 의해 갱신된 원소를 액세스하기 때문에 나중에 취소될 필요가 있다. 한 트랜잭션이 K개 원소를 액세스하기 때문에 나중에 취소될 필요가 있다. 한 트랜잭션이 K개 원소를 액세스한 후, 만약 취소 마크가 없고 2가지 모드의 강성 로크가 액세스된 개개의 원소에 걸린다면 이러한 원소의 각각에 강성 로크가 걸리고 블록 상태로 전환되게 된다. 그 밖의 것들은 취소 마크가 되고 이미 알고 있는 OCC처럼 즉시 취소되거나 순수 OCC 처럼 계속 실행되거나 또는 재실행 중에 2가지 방법의 조합으로 실행될 수도 있다.

순수 OCC 형식으로 접근시 장점은 처음 실행할 때 요구된 원소를 주기억 장치로 가져와서 재실행할 때는 I/O 시간을 줄일 수 있다는 점이다. 이미 취소 마크가 된 트랜잭션은 완료될 수 없기 때문에 그 트랜잭션의 갱신은 지연된 갱신 정책하에서는 버퍼로 반영되지 않을 것이다. 즉 취소 마크가 된 트랜잭션은 더 이상 로크를 요구하지 않는다. 만약 트랜잭션이 블록킹 상태로 진입한다면 이 트랜잭션에 의해 얻어진 강성 로크와 비호환적인 약성 로크를 얻은 모든 트랜잭션은 취소 마크가 되고 진행된 엔트리는 블록킹 상태로 된다. 그 트랜잭션은 액세스된 새로운 원소들에 강성 로크를 요구할 것이고 그 원소가 비호환 모드에서 로크에 걸려 있다면 그 로크가 해제될 때까지 기다릴 것이다. 만약 블록킹 상태에서 강성 로크를 요구하는 중에 데드록이 발생한다면 데드록 사이클에 있는 대기 트랜잭션 중의 하나가 취소되거나 선택된다. 한 트랜잭션이 성공적으로 블록킹 상태로 들어갈 때 데드록 때문에 취소되고 완료 처리는 록킹과 같은 방법으로 처리된다.

III. 모델과 분석

분석은 자료 함유와 하드웨어 자원 함유의 효과를 분리하고 반복 실행을 통해서 그 결과를 얻는다. 로크 함유 확률과 평균 로크 대기 시간, 약성 로크와 강성 로크를 얻기 위한 로크 대기 시간과 로크 함유 확률은 같다. 왜냐하면 약성 공유 로크 대기 시간과 로크 함유 확률은 같다. 왜냐하면 약성 공유 로크와 강성 공유 로크는 강성 배타적 로크와 경쟁하기 때문이고 약성 배타 로크와 강성 배타 로크는 다른 강성 로크들과 경쟁하기 때문이다.

동일 모드에서 약성 로크와 강성 로크에 대한 로크 함유 확률은 비호환적 모드에서 얻어진 강성 로크 만의 함수이고 그것과 동일하다. 약성 로크와 강성 로크에 대한 비슷한 요소가 로크 대기 시간을 가지고 있다. 단지 약성 로크만 걸린 첫 K단계 중에 취소될 확률은 첫번째와 그 다음 실행에서 서로 다른 버퍼 처리 확률을 가지고 있고 그들의 취소 확률도 또한 다르다. 강성 로크를 받을 때 자료 원소 중에 비호환 모드에 있는 강성 로크와 약성 로크는 복구 될 것이고 비호환 로크 모드에 있는 현재 약성 로크 소지자는 취소 마크가 될 것이다. 그래서 취소 확률은 강성 로크가 걸린 비율의 함수이다.

L은 한 트랜잭션에 의해 액세스된 자료 원소의 개수이다. 트랜잭션은 i ($0 \le i \le L+1$) 상태이거나 $\tilde{i}$ ($0 \le i \le L-1$) 상태에 있다고 하고, 0 상태는 초기 설정 상태이고 L+1은 완료 상태이다. i 상태에서 트랜잭션은 i 개 원소를 처리하고 $\tilde{i}$ 상태에서 트랜잭션은 i+1 상태로 진행하기 위하여 로크를 대기하는 것이다. K는 임계 단계이고 ($0 \le K \le L+1$) 각 i 단계의 끝에서 약성 로크가 새로운 원소를 액세스 하도록 요구되는 경우이다. i($i<K$) 상태에서 만약 요구된 요소상에 비호환적인 강성 로크가 이미 걸려 있다면 그 트랜잭션은 $\tilde{i}$ 상태로 들어온 것처럼 모델링되고 약성 로크가 얻어질 때까지 이 상태에 머물게 된다. 그렇지 않으면 그것은 i+1 상태로 바뀐다.

트랜잭션이 그것의 처리를 K단계에서 끝냈을 때 만약 거기에 취소 마크가 되지 않았다면 그 지점까지 액스된 K개 원소의 각각에는 강성 로크가 걸리고 그들 원소상에 비호환적인 약성 로크를 가진 대부분의 트랜잭션은 취소 마크가 된다. i 단계의 다음에는 $K<i<L$ 인데 액세스된 다음 원소에 강성 로크가 걸리고 만약에 로크에 대한 대기가 요구된다면 $\tilde{i}$ 상태로 변하게 된다. 만약 K 상태 뒤에 트랜잭션이 취소 마크가 된다면 그것은 남아 있고 원소를 액세스 하지만 어떤 로크도 기다리지는 않는다. L 상태 이후에 만약 그 트랜잭션이 취소된다면 그것은 시간 Tbackoff 동안 기다리고 상태 1로 복귀하며 그 전 상태대로 진행된다. 그렇지 않으면 그 트랜잭션은 L+1 상태에서 처리를 완료한다.

IINPL : 초기 설정을 위한 트랜잭션당 I/O 값

PINPL : 명령어 개수

RINPL : 트랜잭션당 응답 시간 PINPL과 IINPL의 평균값

트랜잭션의 첫번째 실행에 있어서 i 단계에서 평균 응답 시간은 Ri로 모델링하고 Pi 명령어의 평균 실행 시간과 I/O 시간 Ii의 평균과 같다. 취소에 기인한 트랜잭션의 재 실행에 있어서 i 상태에서 평균 응답 시간은 Ri'' 이고 I/O시간의 Ii와 명령어 Pi' 의 평균과 상응한다. Ii' 는 충분한 버퍼를 가진 시스템에서 상당히 작다. $\tilde{i}$ 상태에서 소비한 평균 대기 시간은

로크를 기다리면서 보내는 평균 시간과 같고 $\widehat{R}_i$로서 표기한다. L+1 상태에서 트랜잭션은 완료 레코드를 로그 파일에 쓰고 T_{commit}의 평균 시간으로 모델링한다. 완료 처리를 하는 중에 액세스된 자료 원소상에 배타적 처리는 유지된다. 큐잉 모델에서 초당 트랜잭션의 도착율이 λ 비율로 도착하는 포아송 분포를 한다고 가정하자. 트랜잭션의 취소 확률을 추정하기 위해서 높은 자료함유 환경 분석 논문[4]에 있는 비슷한 근사치를 사용한다. 트랜잭션이 첫번째 실행에서 취소될 확률 PA 는 다음과 같다.

$$P_A \simeq 1-\prod_{i=1}^{K}\left\{(1-\frac{i}{L_{SPACE}})^{L\lambda R_i}(1-\frac{i-1}{L_{SPACE}})^{L\lambda \widetilde{R_{i-1}}}\right\} \qquad (1)$$

LSPACE는 데이터 베이스에 있는 데이터 원소의 개수이다. 비슷한 방법으로 재실행 트랜잭션에 대한 취소 확률 P'_A는 다음과 같이 추정할 수 있다.

$$P'_A \simeq 1-\prod_{i=1}^{K}\left\{(1-\frac{i}{L_{SPACE}})^{L\lambda R'_i}(1-\frac{i-1}{L_{SPACE}})^{L\lambda \widetilde{R_{i-1}}}\right\} \qquad (2)$$

평균 대기 시간 $\widehat{R}_i$를 분석하면 로크 요구를 독립 상태로 가정하기 때문에 i 상태에 독립인 $\widehat{R}_i = \widehat{R}$로 볼 수 있다.

그런데 $\widehat{R}=P_C \times W$ P_C : 로크 요구상에서 로크 함유 확률이고

W : 로크를 얻기 위한 평균 대기 시간

Si : 록킹 모드에 있을때 i 상태로 변할 때부터 완료될 때까지 첫 실행 트랜잭션 동안에 평균 응답 시간

Si': 록킹 모드에 있을때 i 상태로 변할 때부터 완료될 때까지 재실행 트랜잭션 동안에 존재하는 평균 응답 시간

$\widehat{S}_i$: 재실행이나 첫 실행 중의 트랜잭션 평균 응답 시간 (i > K)

$$S_i = \left\{\sum_{j=1}^{L-1}(R_j+P_C\times W)\right\} + R_L + T_{Commit} \qquad (3)$$

$$S'_i = \left\{\sum_{j=1}^{L-1}(R'_j+P_C\times W)\right\} + R'_L + T_{Commit} \qquad (4)$$

$$\widehat{S}_i = \left\{\sum_{j=1}^{L-1}(\widehat{R}_j+P_C\times W)\right\} + \widehat{R}_L + T_{Commit} \qquad (5)$$

단 $\widehat{R}_j$ 는 다음과 같다.

$$\widehat{R}_j = (1-P_A)R_j + P_A R'_j, \quad K<j\leq L \qquad (6)$$

즉 $\widehat{R}_j$ 는 j 상태(j<K)에서 소비한 평균 응답 시간이다.

---

4) 외국 참고문헌 10 참조.

만약 트랜잭션이 첫번째 실행에서 취소되지 않는다면 확률이 $1-P_A$인 강성 로크 모드를 얻었을때 j 상태에서 Ri 시간을 소비한다. 그렇지 않으면 확률이 P_A인 재실행을 하고 j(j>K) 상태에서 R'j 시간을 소비한다.

W_A : $\hat{R}$ 상태까지 로크에 대한 함유때문에 트랜잭션을 기다리는 시간

W_B : K+1 상태에서 L-1까지 로크 대한 함유 때문에 트랜잭션을 기다리는 시간

W_C : L 상태와 L+1 상태에서 로크에 대한 함유 때문에 트랜잭션을 기다리는 시간

이제는 대기 시간을 수식으로 나타내보자.

$$W = W_A + W_B + W_C \qquad (7)$$

K 상태에서 평균 대기 시간은 로크를 기다리면서 경쟁하는 트랜잭션들로부터 w를 얻을 수 있다. 즉 모든 i로부터 Ri=r, R'i=r', $\widehat{R_i} = \hat{r}$ 로 정리하며, 대기 시간 w를 정리하면 다음과 같다.

$$W = \frac{(1-P_A)X + P_AY}{(\hat{r}+b)\beta + L(\hat{r}-c) - \frac{(\beta+K)}{f_3} + Kb}$$

$$\alpha = \frac{(m\rho)^m}{m!(\alpha+\beta)(1-\rho)}, \quad \beta = \sum_{j=0}^{m-1} \frac{(m\rho)^j}{j!}$$

$$\gamma = 1 + \frac{\alpha}{m(\alpha+\beta)(1-\rho)}$$

m 은 프로세서의 개수이고 IO TIME은 I/O 작동을 수행하는 평균 시간이다.

$$R_{INPL} = \gamma \times \frac{P_{INPL}}{MIPS} + I_{INPL} \times IOTIME$$

$$R_i = \gamma \times \frac{P_i}{MIPS} + I_i \times IOTIMES$$

$$R'_i = \gamma \times \frac{P'_i}{MIPS} + I'_i \times IOTIMES$$

Ii' 는 Ii × PMISS로 추정한다. PMISS는 트랜잭션 재실행 중에 반복된 I/O 확률이다. 이것은 자료 원소가 재실행하는 중에 주기억 버퍼에 존재하지 않을 확률이다. 전체적인 트랜잭션 응답 시간 R은 다음과 같다.

$$R = R_{INPL} + \sum_{i=1}^{K}(R_i + \widetilde{R}_i) + \sum_{i=K+1}^{L}(\widehat{R}_i + \widetilde{R}_i) + P_A \sum_{i=K+1}^{L} R_i$$
$$+ \frac{P_A}{(1-P'_A)}(T_{Backoff} + \sum_{i=1}^{K}(R'_i + \widetilde{R}_i))$$
$$+ P_A \frac{(1}{1-P'_A} - 1) \sum_{i=K+1}^{L} R'_i + T_{Commit}.$$

위의 식을 자세히 설명하면, 처음은 초기 설정치이고, 두 번째식는 첫 실행의 처음 K 단계에서 시간이고, 세번째는 최종 실행의 마지막 L-K 단계에 대한 시간이며, 네번째는 트랜잭션이 취소된다면 첫번째 실행의 마지막 L-K 단계에서 응답 시간이고, 다섯번째식은 재실행 트랜잭션의 처음 K 단계에서 응답 시간이며, 여섯째식은 재실행 중에 다시 취소될 마지막 L-K 단계에서 응답 시간이고 마지막 단계는 완료 시간이다.

IV. 분석 과정

모든 i에 대해서 평균 I/O 비율 (Ii)은 0.73이고 평균 I/O(IOTIME)는 0.035초, 완료 시간 (Tcomnit)은 0.025초, TBackoff 시간은 Tcomnit 시간과 같다고 가정한다. 재실행 트랜잭션은 처음 실행한 것과 동일한 원소를 엑세스하고 데이터 베이스 버퍼는 재실행하는 모든 원소들이 그 버퍼에 있을 수 있는 정도로 충분히 크다고 가정한다.

Fig 1은 위의 파라미터를 사용하고 단일 프로세서를 가진 제안된 방법과 록킹과 순수 OCC방법의 처리표에 대한 평균 응답 시간의 시뮬레이션 예측 값을 나타내고 있다.

제안된 방법에서 K의 임계 값은 5로 나타났고, 제안된 방법은 다른 2가지 방법보다 약 25%정도 높은 처리율을 가진다. 원인을 분석해 보면 순수 OCC방법의 처리율은 CPU 활용도에 제한을 받고 특정 규칙의 처리율은 로크 함유율에 영향을 받는다.

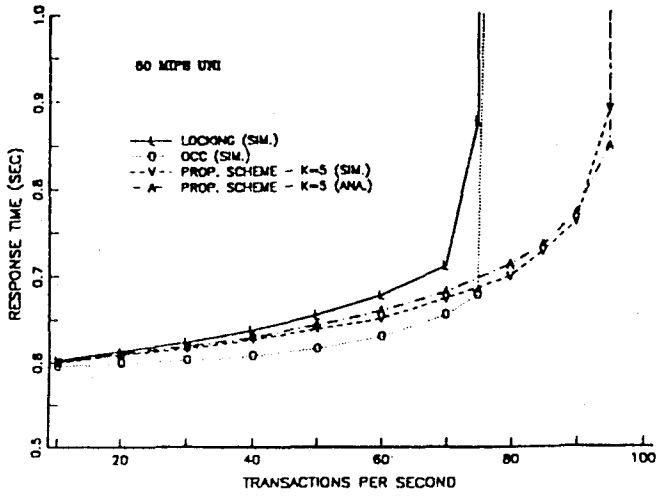

Fig. 1. Response time versus throughput for different approaches.

즉 트랜잭션 비율이 증가함에 따라 로크 함유율을 더 증가시키는 트랜잭션 응답 시간을 증가시키는 결과가 된다. 록킹법의 CPU 활용도는 다른 방법에 비해 75% 정도로 작다. 이 결과만 보더라도 OCC기법이 CPU 자원 제한에 걸릴 때 록킹 기법은 자료 함유의 제한에 걸리게 된다.

제안된 기법은 OCC기법과 비교해서 트랜잭션 취소 확률을 감소시키고 록킹 기법과 비교해서 로크 함유율을 감소시킨다. 이런 역할이 처리율에 대한 두 가지 제한 요소를 감소시키고, 훨씬 더 좋은 처리율을 가져온다.

Fig 1에서 K=5일 때 제안된 기법의 응답 시간은 록킹법과 OCC기법의 중간에 위치한다. 그 이유는 낮은 트랜잭션율에서 로크에 대해 기다리는 것이 트랜잭션을 취소시키는 것보다 응답 시간에 더 많은 영향을 주기 때문이고, I/O를 거의 안하므로 트랜잭션의 재실행 시간이 훨씬 작기 때문이다.

그 결과가 K=5일 때 제안된 기법의 대기 시간 함유 효과는 낮은 트랜잭션율에서 순수 OCC기법보다 훨씬 더 많은 응답 시간을 유발시킨다. 위와 같은 파라미터를 사용했을 때 제안된 방식과 순수 OCC기법의 응답 시간 차이는 초당 60 트랜잭션에서 약 0.2초이다. 순수 OCC기법의 많은 취소율의 충돌은 초당 약 75개 트랜잭션이 응답 시간에 교차하는 높은 트랜잭션 비율을 나타내고 있다.

Fig 2는 서로 다른 K값을 가진 제안된 기법하에서 트랜잭션비율에 대한 첫실행에서 취소될 확률을 나타내고, Fig 3은 그 비율에 대한 로크 함유 확률을 보여준다. 또한 시뮬레이션과 분석적 방법이 거의 비슷하다는 것을 보여주고 있다.

Fig 2와 Fig 3을 비교해 보면, 로크 함유 비율과 트랜잭션 취소 확률에 있어서 K값을 증가시키면 두가지가 서로 반비례함을 알 수 있다. 즉 K값을 증가시키면, 취소 확률은 증가하지만 로크 함유율은 감소함을 알 수 있다. Fig 2에서 알 수 있듯이, OCC기법은 K값을 증가시킴에 따라 훨씬 더 높은 첫 취소 확률을 나타낸다. K값이 0인 제안된 기법과 관련해서 록킹 기법에 대한 취소 확률은 단지 데드록 때문인데 아주 적다. [5]

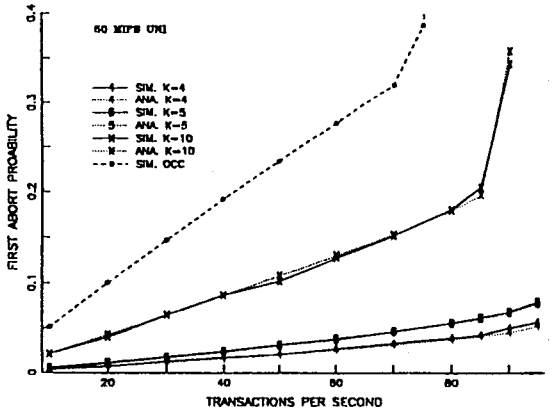

Fig. 2. Impact of varying the critical stage on abort probability.

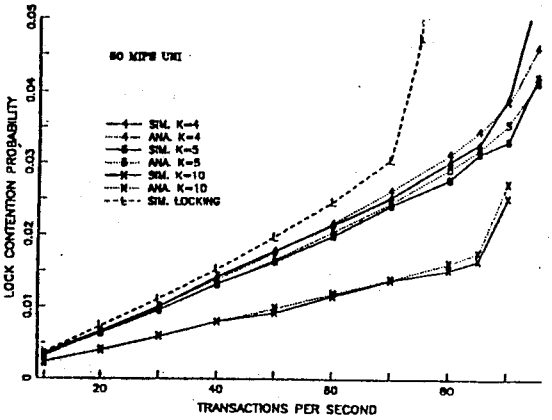

Fig. 3. Impact of varying the critical stage on lock contention probability.

5) 외국 참고문헌 8, 12 참조

Fig 3은 록킹 기법에 대한 로크 함유 확률이고 이것은 K값이 큰 경우에 대한 제안된 기법보다 훨씬 높은 비율을 보인다. 제안된 기법하에서, OCC기법보다 처리율이 약 15% 높은 가장 좋은 K값을 생각해 보자. 최적의 K값을 단일 프로세서인 경우 5에서 8까지 증가할 때이다. 그러나 로크 함유율이나 자원의 소모량 등을 고려할 때 K값은 5가 최적 값이라고 할 수 있겠다.

V. 결 론

제안된 동시성 제어 기법은 트랜잭션 처리 시스템의 성능에 영향을 미친다. 록킹 기법은 대기 트랜잭션이 로크를 가지고 있으면서 진행 중인 다른 트랜잭션을 블록 시키는 블록킹 현상을 초래한다. 높은 데이터 함유 환경하에서, 대기 트랜잭션의 수가 증가함에 따라 로크 테이블 목록보다 더 많은 수가 블록 되고 로크 요구가 거의 이루어지지 않게 된다. 제안된 기법은 트랜잭션의 실행의 나중 단계에서 트랜잭션의 블록킹 형태를 지연시키는 방법으로 록킹과 비교해서 블록킹 확률을 감소시킨다.

OCC 하에서 자료 함유는 하드웨어 자원에 무리한 영향을 주는 트랜잭션 취소를 유발시키고 성능을 감소시킨다.

제안된 기법을 트랜잭션 실행의 초기 단계 중에 발생하는 취소를 허락하고, 트랜잭션의 나중 단계 중에 일어나는 비호환 강성 로크가 해제될 때까지 확실히 기다림에 의해서 OCC기법을 사용하는 것과 비교해서 트랜잭션 취소 확률을 감소시킨다. 제안된 기법은 록킹과 OCC기법의 조합으로 볼 수 있다. 제안된 기법하에서 트랜잭션 실행은 다른 트랜잭션을 블록시키지 않고 로크를 기다리는 비블록킹 상태와 록킹 방법과 같은 블록킹 상태로 나누어 볼 수 있다.

비블록킹 상태 중에 액세스된 데이터는 OCC 기법하에서 트랜잭션 취소와 같다. 제안된 기법하에서 트랜잭션은 비블록킹 상태일 때는 확실히 로크를 기다리기 때문에 취소 확률은 검증하에서 현재 엑세스 되고 있는 자료 원소를 회피함으로써 감소된다. 더욱이 데드록을 제외하고 블록킹 상태 중에 데이터 액세스는 트랜잭션을 취소시킬 수 없다.

K값을 적당하게 선택하기 위해서 트랜잭션이 블록킹 상태로 진입한 후에 임계 상태로 된다는 규칙은 트랜잭션 취소와 로크 대기의 효과 사이에 균형을 유지하도록 해준다. 이러한 접근이 모든 자료함유 수준과 자원 함유 수준에서 록킹이나 OCC기법보다 더 좋은 성능을 나타낸다. 최적의 K값을 알아내기 위해 OCC기법의 록킹 기법보다 뛰어난 높은 자료 함유 환경하에서, K값의 증가는 트랜잭션에 대해 액세스된 평균 원소의 개수에 가까워진다는 것은 바람직하다. 록킹이 OCC기법보다 더 우수한 낮은 자료 함유 환경하에서는 K값이 작은 것이 바람직하다. 대략적인 분석은 제안된 기법의 성능을 추정하도록 계발되고, 시뮬레이션은 그 분석을 검증하도록 구성되어 있다. 분석에서 응답 시간 뿐만 아니라 취소 확률과 로크 함유 확률은 시뮬레이션 결과와 거의 동일하다는 것을 알았다.

앞으로의 연구 과제는 분석 방법을 좀더 개선시키는 문제를 다루었으면 좋겠고, 다중 프로세서의 처리율 특성이나 분석 기법 등에 관해서도 더 많은 연구가 이루어지길 바란다.

국내 참고 문헌

1. 문송천, 데이터베이스 시스템 총론, 형설출판사, 1993
2. 박석, 데이터베이스 시스템, 홍릉과학출판사, 1993
3. 백두권,이경상, 데이터베이스 구조, 상조사, 1994.
4. 이병욱, 데이터베이스 시스템, 생능출판사, 1993.

외국 참고문헌

1. R.Agrawal,M.J.Carey,and M.Livny,"Concurrency control performance modeling: Alternatives and implications", ACM Trans.Database Syst., Vol.12, no.4, pp. 609-654, Dec.1987.
2. M.J.Carey, and M.Livny, "Conflict detection trade-offs for replicated data",ACM Trans.Database Syst., Vol.16, no.4, pp. 703-746, Dec.1991.
3. B.ciciani,D.M.Dias,and P.S.Yu, "Analysis of replication in distributed database systems," IEEE Trans.Knowledge Data Eng., Vol.2, no.2,pp. 247-261,June 1990.
4. B.ciciani,D.M.Dias,B.R.Iyer,and P.S.Yu, "A hybrid distributed centralized system structure for transaction processing," IEEE Trans. Software Eng.,vol.16,no.8,pp.791-806,Aug.1990.
5. A Dan,and P.S.Yu, "Performance analysis of buffer coherency policies in a mulisystem data sharing environment," IEEE Trans.Parallel Distributed syst.,vol.4,no.3,pp.289-305,Mar.1993.
6. P.A.Franaszek,J.T.Robinson,and A.Tomasian, "Concurrency control for high contention environment," ACM Trans.Database Syst., Vol.17, no.2, pp. 304-345, 1992.
7. H.T.Kung and J.T.Robinson, "On optimistic method for concurrency control," ACM Trans.-Database Syst., Vol.6, no.2, pp. 213-226, June 1981.
8. Y.C.Tay,N.Goodman,and R.Suri, "Locking performance in centralized databases," ACM Trans.-Database Syst., Vol.10, no.4, pp. 415-462, Dec. 1985.
9. A.Thomasian and I.K.Ryu, "Performance Analysis of two-phase locking," IEEE Trans. Software Eng., vol.17, no.5, pp.386-401, May 1991.
10. P.S.Yu,and D.M.Dias, "Analysis of hybrid concurrency control schemes for a high data contention environment," IEEE Trans. Software Eng., vol.18,no.2,pp.118-129,1992.
11. P.S.Yu,and D.M.Dias, "Performance analysis of concurrency control using locking with deferred blocking," IEEE Trans. Software Eng., vol.19, no.10,pp.982-996,Oct.1993.
12. P.S.Yu,D.M.Dias,and S.S.Lavenberg, "On the analytical modeling of database concurrency control," J.Ass.Comput.Mach vol.40,no.4, pp.831-872, Sept.1993.

남 태 희* 정회원
- 1989년 경성대학교 경영학과 (경영학사)
- 1992년 경성대학교 산업정보학과 (공학석사)
- 1989~1992년 우성전산 직업훈련원 전산실장
- 1993년~현재 동주여자전문대학 무역사무자동화과 전임강사
- 관심분야 : 데이터베이스, 정보 통신, MIS

박 재 운** 정회원
- 1991년 2월 동아대학교 전자공학과 (공학사)
- 1983년 2월 동아대학교 전자공학과 (공학석사)
- 1993년 2월 동아대학교 전자공학과 (박사수료)
- 현재 동래여자전문대학 사무자동화과 부교수
- 관심분야 : SS 통신, 무선 LAN

위 승 민***
- 1986년 2월 한국해양대학 항해학과 (공학사)
- 1993년 2월 경성대학교 산업정보학과 (공학석사)
- 현재 한국해양대학교 해사수송과학과 박사과정중
- 관심분야 : 데이터베이스, 멀티미디어.