주기억장치 데이타베이스 시스템을 위한 실시간 정적 로킹 기법의 설계 및 구현

(Design and Implementation of Real-Time Static Locking Protocol for Main-Memory Database Systems)

김 영 철 † 유 한 양 † 김 진 호 †† 김 준 ††† 서 상 구 ††††

(Young-Chul Kim) (Han-Yang You) (Jin-Ho Kim) (June Kim) (Sang-Ku Seo)

요 약 모든 데이타를 주기억장치에 상주시키는 주기억장치 데이타베이스 시스템은 고성능 실시간 트랜잭션 처리에 적합하다. 주기억장치 데이타베이스 시스템에서 트랜잭션이 데이타베이스에 접근하는 시간이 매우 짧기 때문에 동시성 제어를 위해 이단계 로킹 기법을 사용할 경우, 로크 충돌이 일어날 확률이 적은 반면에, 데이타 객체를 접근할 때마다 수행해야 하는 로킹 연산의 부하는 트랜잭션 수행시간에 비해 상대적으로 큰 비중을 차지하게 된다. 본 논문에서는 로킹 연산의 부하를 최소화하면서 트랜잭션의 우선 순위를 반영한 실시간 정적 로킹 기법을 설계하고, 이를 주기억장치 실시간 데이타베이스 시스템인 Mr.RT에서 구현하였다. 또한 이단계 로킹 기법을 기반으로 하는 기존의 실시간 동시성 제어 기법들(2PL-PI, 2PL-HP)과의 성능 비교를 통하여 실시간 정적 로킹 기법이 보다 좋은 성능을 보임을 확인하였다.

키워드 : 실시간 정적 로킹, 실시간 데이타베이스, 이단계 로킹 기법, 주기억장치 데이타베이스, 동시성 제어

Abstract Main-memory database systems which reside entire databases in main memory are suitable for high-performance real-time transaction processing. If two-phase locking(2PL) as concurrency control protocol is used for the transactions accessing main-memory databases, however, the possibility of lock conflict will be low but lock operations become relatively big overhead in total transaction processing time. In this paper, we designed a real-time static locking(RT-SL) protocol which minimizes lock operation overhead and reflects the priority of transactions and we implemented it on a main-memory real-time database system, Mr.RT. We also evaluate and compare its performance with the existing real-time locking protocols based on 2PL such as 2PL-PI and 2PL-HP. The extensive experiments reveal that our RT-SL outperforms the existing ones in most cases.

Key words : real-time static locking, real-time databases, two-phase locking protocol, main-memory databases, concurrency control

1. 서 론

트랜잭션이 종료되어야 할 마감시간에 대한 제약 조건과 데이타베이스의 일관성을 유지해야 하는 실시간 데이타베이스에 주기억장치 데이타베이스 기술을 많이 활용하고 있다. 주기억장치 데이타베이스란 모든 데이타를 주기억장치에 저장하여 트랜잭션을 고속으로 처리할 수 있을 뿐만 아니라 디스크 입출력에 의해 발생하는 불확실성을 제거하여 고성능의 실시간 트랜잭션 처리에 적합하다고 인식되고 있다[1,2,3].

실시간 데이타베이스 시스템에서는 트랜잭션의 마감시간을 모두 (또는 최대한 많이) 만족시킬 수 있도록 트랜잭션의 수행 순서를 조절하는 스케쥴링 기능이 필요하다. 이러한 실시간 트랜잭션 스케쥴링 기법들은 실시간 시스템에서 제안된 EDF(Earliest Deadline First), LSF (Least Slack First), RM(Rate Monotonic) 알고리즘

† 비 회 원 : 한국전자통신연구원
kimyc@etri.re.kr
hyyou@etri.re.kr
†† 종신회원 : 강원대학교 컴퓨터과학과 교수
jhkim@kangwon.ac.kr
††† 종신회원 : 한국전자통신연구원
jkim@etri.re.kr
†††† 종신회원 : 광운대학교 경영정보학과 교수
skseo@mail.gwu.ac.kr
논문접수 : 2000년 8월 14일
심사완료 : 2002년 9월 16일

등을 기반으로 연구되었다[4]. 하지만 데이타베이스의 일관성을 보장하기 위해 사용하는 동시성 제어 기법에서는 낮은 우선 순위의 트랜잭션이 유지하고 있는 로크를 높은 우선 순위의 트랜잭션이 대기할 수 있다. 이러한 로크 대기로 인해 트랜잭션이 우선 순위대로 수행되지 못하며 실시간 스케쥴을 그대로 반영할 수 없게 된다. 따라서 실시간 스케쥴링의 효과를 반영하면서 동시에 데이타베이스의 일관성을 만족할 수 있는 실시간 동시성 제어 기법에 대한 연구가 진행되었다[4,1,2,3].

실시간 동시성 제어 기법들은 대부분 디스크 기반 데이타베이스 시스템에서 널리 사용되는 이단계 로킹(two-phase locking, 2PL) 기법에 트랜잭션의 우선 순위를 고려하여 높은 우선 순위를 갖는 트랜잭션이 낮은 우선 순위 트랜잭션에 의해 수행이 지연되는 우선 순위 반전(priority inversion)의 문제를 해결하기 위한 여러 가지 기법들이 제안되었다. 우선 순위 반전이란 우선 순위 선점 CPU 스케쥴링을 사용하는 실시간 데이타베이스 시스템에서 낮은 우선 순위 트랜잭션(T_L)이 소유한 로크를 대기하고 있는 높은 우선 순위 트랜잭션(T_H)이 있을 때, 중간 우선 순위를 갖는 트랜잭션(T_M)이 우선 순위에 의해 T_L을 선점할 경우, 더 높은 우선 순위의 T_H가 낮은 우선 순위의 T_M에 의해 수행이 지연되는 것을 말한다. 이러한 우선 순위 반전을 해결하기 위한 실시간 동시성 제어 기법으로 로크를 소유한 낮은 우선 순위 트랜잭션에 로크를 대기하는 높은 우선 순위 트랜잭션의 우선 순위를 상속함으로써 우선 순위 반전을 해결하는 2PL-PI(priority inheritance)와 로크를 소유한 낮은 우선 순위 트랜잭션의 수행이 얼마나 이루어졌는가의 조건에 따라 철회시키거나 우선 순위를 상속하는 2PL-CPI(conditional priority inheritance) 등이 제안되었다[5,1]. 또한 로크 충돌시 로크를 요청하는 높은 우선 순위 트랜잭션이 그 로크를 소유한 낮은 우선 순위 트랜잭션을 철회시킴으로써 우선 순위 반전이 발생하지 않는 2PL-HP(high priority)와 트랜잭션이 완료하기까지 남아있는 여유시간(slack time)의 정도에 따라 로크를 허용하거나 또는 트랜잭션을 철회시키는 2PL-CR(conditional restart) 등이 제안되었다[4]. 하지만 우선 순위 상속으로 여러 트랜잭션들이 자신의 우선 순위가 아닌 보다 높은 우선 순위를 가지고 실행됨으로써 시스템의 스케쥴 가능성에 나쁜 영향을 미치며, 트랜잭션의 철회와 재시작으로 시스템의 성능과 자원을 낭비할 수 있다.

이러한 이단계 로킹 기반 실시간 동시성 제어 기법들은 디스크 기반 데이타베이스 시스템에서 트랜잭션의 동시성을 높이기 위해 설계되었으며 데이타 객체를 접근할 시점에 그때마다 로크를 설정하고 트랜잭션 종료시에 모든 로크를 한꺼번에 해제하게 된다. 그러므로 로크 충돌 검사 및 로크 설정 등 빈번한 로킹 연산과 복잡한 로크 테이블 유지에 상당한 부하를 갖는다. 디스크 기반 데이타베이스 시스템에서 로킹 연산은 디스크 입출력 보다 빠르게 수행되지만, 모든 데이타가 주기억장치에 상주하는 주기억장치 데이타베이스 시스템에서는 상당한 부담이 된다. 또한 주기억장치 데이타베이스 시스템에서는 트랜잭션의 수행이 빠르게 이루어지기 때문에 로크를 유지하는 시간도 짧다. 따라서 로크 충돌이 일어날 가능성은 적은 반면에 로킹 연산의 비용이 실제 데이타를 처리하는 비용 보다 상대적으로 크다는 특징을 갖는다. 그러므로 트랜잭션 수행 중에 동적으로 데이타 객체에 대한 로크를 요청하는 이단계 로킹 기법은 트랜잭션의 로킹 연산 비용이 크기 때문에 주기억장치 데이타베이스 시스템에 큰 부담을 주게 된다. 뿐만 아니라 트랜잭션 수행 중에 로크 대기가 발생할 수 있어서 트랜잭션 수행시간을 예측할 수 없게 만든다.

본 논문에서는 주기억장치 데이타베이스 시스템에서 로킹 연산의 비용을 줄이면서 실시간 트랜잭션들의 동시성 제어에 적합한 실시간 정적 로킹 기법을 설계하고 구현하였다. 실시간 정적 로킹 기법은 트랜잭션의 시작 시점에 모든 로크를 한꺼번에 요청하는 정적 로킹 기법[6]에 바탕을 두고 있다. 이 방법에서는 접근할 모든 데이타 객체에 대한 로크를 획득한 경우에만 트랜잭션의 수행이 시작될 수 있으므로, 일단 수행을 시작한 트랜잭션은 로크 충돌로 대기하지 않고 수행을 완료할 수 있다. 또한 로크를 대기하는 트랜잭션들은 어떠한 데이타 객체에 대해서도 로크를 갖고 있지 않으므로 교착상태가 발생하지 않으며, 트랜잭션의 시작 시점에서 한꺼번에 로킹 연산을 수행하므로 로크 관리가 단순해지는 장점이 있다.

정적 로킹의 이러한 장점 때문에 실시간 동시성 제어 기법으로 일부 시도된 바 있다[7]. 그러나 이 연구는 분산 실시간 데이타베이스에서 실시간 정적 로킹 기법의 성능을 시뮬레이션을 통해 평가하였을 뿐 실제 주기억장치 데이타베이스 시스템에서 구현·평가되지 못하였다. 따라서 이 논문에서는 실제 주기억장치 실시간 데이타베이스 시스템에서 로킹 부하를 최소화하면서 트랜잭션의 우선 순위를 반영하는 실시간 정적 로킹 기법(Real-Time Static Locking Protocol, RT-SL)을 설계하고 이를 구현하였다. 이를 위해 로크를 대기하고 있는 트랜잭션의 우선 순위를 고려하여 높은 우선 순위를 갖

는 트랜잭션이 로크 충돌로 오랫동안 대기하지 않도록 하는 로크 요청 알고리즘과 수행을 완료한 트랜잭션이 로크를 해제하면서 다음에 로크를 허용할 트랜잭션을 결정할 때도 로크를 대기하고 있는 트랜잭션의 우선 순위에 따라 로크를 허용하는 로크 해제 알고리즘을 설계하였다. 그리고 이를 주기억장치 실시간 데이타베이스 시스템으로 개발된 Mr.RT[8]에서 구현하고, 기존의 실시간 동시성 제어 기법들과의 성능비교를 통해 성능이 우수함을 보이고자 한다.

또한, 기존의 낙관적 기법, time-stamp 기법 등을 기반으로 하는 실시간 동시성 제어 기법들이 많이 연구되었다[1,9,10,11,12,13,14,15,16,17]. 하지만 본 논문에서는 로킹을 기반으로 하는 실제 시스템인 Mr.RT 상에서 실시간 동시성 제어 기법들의 성능을 평가하기 위한 것으로 이러한 기법들과의 성능비교는 고려하지 않았다.

이 논문은 다음과 같이 구성된다. 2장에서는 주기억장치 실시간 데이타베이스 시스템에서 실시간 정적 로킹 기법의 로크 요청 알고리즘과 로크 해제 알고리즘을 설계하고, 3장에서는 이를 Mr.RT에서 구현한 내용에 대해 기술한다. 4장에서는 Mr.RT에 대한 간단한 소개와 실시간 정적 로킹 기법의 성능 평가 결과를 요약하고, 끝으로 5장에서 결론을 맺는다.

2. 실시간 정적 로킹 기법의 설계

2.1 로킹 단위

디스크 기반 데이타베이스 시스템에서 데이타 접근을 위한 로킹 연산은 실제 데이타의 검색, 삽입, 삭제, 갱신 등 디스크 입출력을 수반하는 데이타베이스 연산 보다 매우 빠르게 수행된다. 따라서 대부분의 상용 DBMS에서는 트랜잭션들 간의 로크 충돌의 횟수를 줄이고 동시성을 높임으로써 시스템의 성능을 향상시킬 수 있도록 레코드와 같은 작은 로킹 단위를 사용한다.

하지만 데이타가 주기억장치에 상주하는 주기억장치 데이타베이스 시스템에서는 디스크 입출력이 불필요하므로 트랜잭션 처리가 훨씬 빠르다. 따라서 트랜잭션이 로크 충돌로 대기하는 시간이 짧은 반면, 로킹 연산의 비용이 트랜잭션 수행에 차지하는 비중이 상대적으로 크다. 이러한 점에서 주기억장치 데이타베이스 시스템에서는 주로 로킹 단위(locking granularity)로 테이블과 같은 큰 단위를 사용한다[1,2,3]. 또한 일부에서는 전체 데이타베이스를 로킹 단위로 사용한다. 이는 결국 트랜잭션들이 순차적으로 수행되는 것으로 동시성 제어에 필요한 비용을 제거함으로써 시스템의 성능을 높이고자 하는 것이다. 하지만 이러한 경우 한 트랜잭션이 완전히 종료될 때까지 다른 모든 트랜잭션들이 대기해야 하므로 트랜잭션의 마감시간을 보장하는데 매우 취약하다. 따라서 본 논문에서는 테이블을 로킹 단위로 하며, 로킹 모드로는 공유(share) 로크와 배타(exclusive) 로크를 사용한다.

2.2 로킹 데이타

본 논문에서는 테이블을 로킹 단위로 사용함으로써 트랜잭션이 어떠한 테이블을 접근할 것인지를 SQL 문장에서 미리 알 수 있다. 즉, 예를 들어 SQL의 데이타 조작문(Data Manipulation Language)의 경우에서 처럼 트랜잭션이 접근하려는 테이블을 알 수 있다. 따라서 실시간 정적 로킹 기법은 트랜잭션이 수행을 시작하기 전에 읽거나 쓸 테이블에 대한 정보를 갖고 있는다.

주기억장치 데이타베이스 시스템에서는 전체 데이타베이스가 주기억장치에 상주하므로 실제 로킹 대상인 데이타 객체에 로킹 정보를 유지함으로써 해쉬 테이블 조작과 같은 부하와 이로 인한 로킹 연산의 비용을 줄일 수 있다. 따라서 이 논문에서는 시스템 카탈로그의 각 테이블 항목에 다음과 같은 로킹 정보를 유지하고 간단한 조작으로 로크 요청과 로크 해제 동작을 수행할 수 있도록 한다.

- 현재 테이블에 허용된 로크
- 테이블에 대한 로크를 획득한 트랜잭션의 개수
- 테이블에 로크 요청을 하였다가 로크 충돌로 대기하는 로크 요청자 노드 리스트 포인터

현재 테이블에 허용된 로크는 공유(share) 로크, 배타(exclusive) 로크, No_Lock중 한 값을 가지며, 이 테이블에 대해 로크를 요청하는 트랜잭션은 이 값을 가지고 로크 충돌을 체크한다. 여기서 No_Lock이란 테이블에 어떠한 로크도 허용되지 않은 상태를 나타낸다. 그리고 테이블에 대한 로크를 얻은 트랜잭션의 개수를 유지함으로써 트랜잭션이 수행을 완료하거나 철회되어 로크를 해제할 때 로크를 대기하고 있는 트랜잭션에게 로크를 허용할 수 있는 지를 결정할 수 있다. 또한 로크 충돌로 대기하는 각 트랜잭션의 정보를 갖는 로크 요청자들을 우선 순위순으로 유지하는 로크 요청자 노드의 리스트에 대한 포인터를 갖는다.

실시간 정적 로킹 기법에서 트랜잭션은 로크를 요청한 테이블 중에서 어떤 한 테이블에 대해서라도 로크 충돌이 발생하면, 모든 테이블의 로크 요청자 노드 리스트에 로크 요청자 노드를 삽입한다. 이러한 로크 요청자 노드에는 해당 트랜잭션이 테이블에 요청하는 로크와 로크 요청자에 대한 포인터, 그리고 다음 로크 요청자 노드에 대한 포인터를 유지한다.

로크 충돌로 대기하는 트랜잭션마다 하나씩 생성되는 로크 요청자는 트랜잭션에 관련된 다음의 정보를 포함한다.

- 트랜잭션 식별자
- 트랜잭션 우선 순위
- 로크를 요청한 테이블들 중에서 실제 로크 충돌이 발생한 테이블의 개수

시스템에서 수행중인 모든 트랜잭션들은 로킹과 관련된 정보로, 트랜잭션이 접근하는 테이블의 개수, 로크를 획득하거나 또는 대기중인 테이블들에 대한 시스템 카탈로그 항목을 가리키는 포인터 그리고 각 테이블에 접근하는 로킹 모드를 갖는다.

2.3 로크 요청 알고리즘

실시간 정적 로킹 기법은 현재 테이블에 허용되어 있는 로크와 그 테이블에 대해 로크를 요청하는 트랜잭션과 로크를 대기하고 있는 트랜잭션들간의 우선 순위를 비교하여 로크 충돌을 결정한다. 즉, 테이블에 어떠한 로크도 허용되어 있지 않거나 또는 공유 로크가 허용되어 있는 경우에는 로크를 요청한 트랜잭션의 우선 순위가 로크를 대기중인 어떤 한 트랜잭션의 우선 순위 보다 낮을 때, 요청한 로크와 상관없이 로크 충돌로 간주한다. 만약 트랜잭션의 우선 순위와 상관없이 로크 호환성에 따라 낮은 우선 순위 트랜잭션에 로크를 허용한다면 로크를 대기중인 높은 우선 순위 트랜잭션의 수행이 그만큼 더 지연될 수 있다. 따라서 높은 우선 순위를 갖는 트랜잭션이 로크 충돌로 대기하고 있을 때 낮은 우선 순위 트랜잭션의 로크 요청을 충돌로 간주함으로써 높은 우선 순위 트랜잭션의 수행이 오랫동안 대기하지 않고 완료할 수 있다. 테이블에 아무런 로크도 허용되어 있지 않은 상태를 나타내는 No_Lock인 경우에도 로크를 대기하는 트랜잭션이 존재할 수 있는 이유는 트랜잭션이 접근하려는 모든 테이블에 대해 로크를 얻지 못하면 어떠한 테이블에 대해서도 로크가 허용되지 않기 때문이다.

표 1 로크 호환성 테이블

Lock granted \ Lock requested		Share	Exclusive
No_Lock	T_{wait} is not exist	Compatible	Compatible
	$Prio(T_{req})<Prio(T_{wait})$	Conflict	Conflict
	$Prio(T_{req})>Prio(T_{wait})$	Compatible	Compatible
Share	T_{wait} is not exist	Compatible	Conflict
	$Prio(T_{req})<Prio(T_{wait})$	Conflict	Conflict
	$Prio(T_{req})>Prio(T_{wait})$	Compatible	Conflict
Exclusive		*Conflict*	*Conflict*

실시간 정적 로킹 기법에서 로크 충돌 여부는 표 1의 로크 호환성 테이블에 의해 다음의 세 가지 경우와 같이 결정된다. 여기서 T_{req}은 테이블에 대해 로크를 요청하는 트랜잭션, T_{wait}는 테이블에서 로크를 대기하고 트랜잭션들 중에 가장 우선 순위가 높은 트랜잭션, 그리고 Prio(T)는 트랜잭션 T의 우선 순위를 나타낸다.

(1) 테이블에 허용된 로크가 없는 No_Lock인 경우
  (1.1) T_{wait}가 없을 때,
    ① 요청하는 로크와 상관없이 충돌이 발생하지 않는다.
  (1.2) T_{wait}가 있고, T_{wait}의 우선 순위가 T_{req}의 우선 순위 보다 높을 때,
    ① 로크 충돌이 발생한다.
    ② T_{wait}가 다른 테이블에 대해 로크를 대기하고 있는 상태에서 T_{wait} 보다 낮은 우선 순위를 갖는 T_{req}에 먼저 로크를 허용함으로써 T_{wait}의 수행이 지연되는 것을 막는다.
  (1.3) T_{wait}가 있고, T_{req}의 우선 순위가 T_{wait}의 우선 순위 보다 높을 때,
    ① 요청하는 로크와 상관없이 충돌이 발생하지 않는다.

(2) 테이블에 공유 로크가 허용된 경우
  (2.1) T_{wait}가 없을 때,
    ① 배타 로크를 요청하는 경우에만 충돌이 발생한다.
  (2.2) T_{wait}가 있고, T_{wait}의 우선 순위가 T_{req}의 우선 순위 보다 높을 때,
    ① 요청하는 로크와 상관없이 충돌이 발생한다.
    ② T_{wait} 보다 낮은 우선 순위를 가지면서 공유 로크를 요청하는 T_{req}에 먼저 로크를 허용함으로써 T_{wait}의 수행이 지연되는 것을 막는다.
  (2.3) T_{wait}가 있고, T_{req}의 우선 순위가 T_{wait}의 우선 순위 보다 높을 때,
    ① 배타 로크를 요청하는 경우에만 충돌이 발생한다.

(3) 테이블에 배타 로크가 허용된 경우
  (3.1) 요청하는 로크와 상관없이 무조건 충돌이 발생한다.

예제 1: 트랜잭션 T_1, T_2, T_3, T_4가 각각 테이블 (R_2, R_5), (R_1, R_2), (R_2, R_4), (R_1, R_3)에 대해 배타 로크를 요청할 때 로크가 허용되는 순서를 살펴보자. 트랜잭션의 우선 순위는 $T_1 < T_2 < T_3 < T_4$ 순이다. 그림 1은 로크를 대기하는 트랜잭션의 우선 순위와 상관없이 현재 테이블에 허용되어 있는 로크만을 가지고 로크 충돌을 검사할 때의 로크 허용 순서이고, 그림 2는 로크 대

기 트랜잭션의 우선 순위에 따라 로크 충돌을 검사할 때의 로크 허용 순서이다.

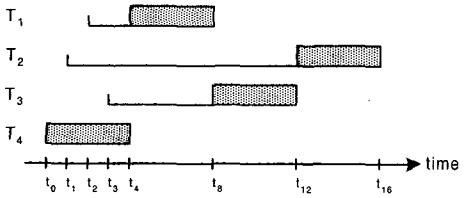


그림 1 로크 대기 트랜잭션의 우선 순위와 상관없는 로크 충돌 검사

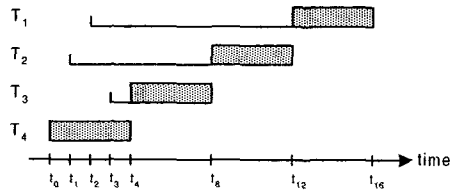


그림 2 로크 대기 트랜잭션의 우선 순위에 따른 로크 충돌 검사

그림 1에서 T_4는 t_0에서 R_1, R_3에 대해 로크를 허용받고 수행을 계속한다. 그 뒤에 t_1에서 R_1, R_2에 대해 로크를 요청하는 T_2는 R_2에 대해서는 로크 충돌이 발생하지 않지만, R_1이 T_4에게 배타 로크로 허용되어 있으므로 로크 대기 상태에 들어간다. 이때 t_2에서 R_2, R_5에 대해 로크를 요청하는 T_1은 아무런 로크 충돌도 발생하지 않으므로 R_2와 R_5에 대해 로크가 허용된다. 그 다음 t_3에서 T_3는 R_2, R_4에 대해 로크를 요청하지만 R_2가 이미 T_1에게 배타 로크가 허용되어 있는 상태이므로 로크 대기 상태에 들어가게 된다. 결국 T_4가 t_4에서 수행을 완료하고 로크를 해제하더라도 R_1에 대해 로크를 대기하고 있던 T_2는 R_2가 T_1에게 배타 로크가 허용된 상태이기 때문에 계속 로크 대기 상태에 있어야 한다. 따라서 로크를 대기하고 있는 트랜잭션의 우선 순위와 상관없이 현재 테이블에 허용되어 있는 로크에 대해서만 로크 충돌을 검사할 경우 낮은 우선 순위를 갖는 트랜잭션 T_1에 의해 그보다 높은 우선 순위를 갖는 트랜잭션 T_2와 T_3의 수행이 지연된다.

로크 대기 트랜잭션의 우선 순위를 고려한 그림 2에서는 T_1이 t_2에서 R_2, R_5에 대해 로크를 요청할 때 R_5에서는 로크 충돌이 발생하지 않고 로크 대기 트랜잭션도 없지만 R_2에는 t_1에서 로크를 대기하고 있는 T_2가 있으므로 T_2 보다 우선 순위가 높을 때에만 수행을 계속할 수 있다. 하지만 T_1은 T_2 보다 우선 순위가 낮기 때문에 로크 충돌이 발생한 것으로 간주하여 로크 대기 상태에 들어간다. 따라서 그다음 t_3에서는 T_3가 R_2, R_4에 대해 로크가 허용되므로 T_2와 T_3는 T_1에 의해 지연되지 않고 수행을 완료할 수 있다.

예제 2: 트랜잭션 T_1, T_2, T_3, T_4가 동일한 테이블을 접근한다고 가정하자. 트랜잭션의 우선 순위는 $T_1 < T_2 < T_3 < T_4$ 순이며, T_1, T_2, T_4는 공유 로크를, 그리고 T_3는 배타 로크를 요청한다.

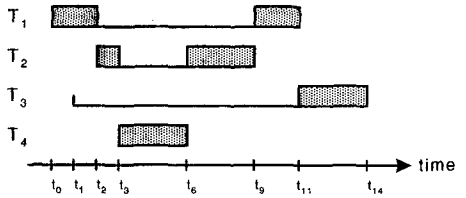


그림 3 로크 대기 트랜잭션의 우선 순위와 상관없는 로크 충돌 검사

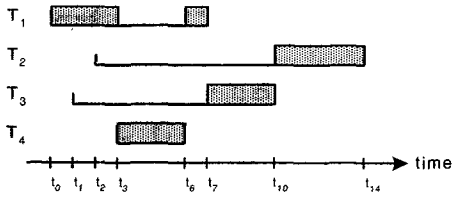


그림 4 로크 대기 트랜잭션의 우선 순위에 따른 로크 충돌 검사

그림 3에서 트랜잭션 T_1이 t_0에서 공유 로크를 허용받은 후 도착한 트랜잭션 T_3는 t_1에서 T_1과 로크 충돌이 발생하므로 로크 대기 상태에 들어가게 된다. 이때 t_2에서 트랜잭션 T_2가 공유 로크를 요청하면 로크 충돌이 발생하지 않으므로 T_2는 우선 순위가 낮은 T_1을 선점하고 수행을 계속한다. 그 결과 우선 순위가 높은 T_3가 우선 순위가 낮은 뒤에 들어온 T_2에 의해 수행이 지연된다.

하지만 그림 4에 있는 이 논문에서 설계한 방법 즉, 대기하는 트랜잭션의 우선 순위에 따라 로크 충돌을 체크하는 방법에서는 t_2에서 공유 로크를 요청하는 트랜잭션 T_2의 우선 순위가 대기하는 트랜잭션 T_3의 우선 순

위 보다 낮으므로 로크 충돌이 발생한 것으로 간주하므로 로크 충돌로 대기하고 있는 높은 우선 순위 트랜잭션 T_3가 공유 로크를 요청하는 낮은 우선 순위 트랜잭션 T_2에 의해 수행이 지연되지 않는다.

2.4 로크 해제 알고리즘

트랜잭션은 수행을 완료하거나 철회할 때 자신이 소유한 모든 테이블에 대한 로크를 해제하고 각 테이블에서 다음에 로크를 허용할 트랜잭션을 결정한다. 실시간 정적 로킹 기법에서는 트랜잭션이 로크를 해제할 때 로크를 대기하고 있는 트랜잭션들의 우선 순위에 따라 다음에 로크를 허용할 트랜잭션을 선정하도록 설계하였다. 이때 만약 다음에 로크가 허용되는 트랜잭션의 로킹 모드가 공유 로크이면, 테이블에서 로크를 대기하는 트랜잭션들 중에서 공유 로크를 요청하면서 다른 모든 배타 로크를 요청하는 트랜잭션들 보다 우선 순위가 높은 트랜잭션들에 대해서만 로크를 허용한다. 따라서 높은 우선 순위를 갖는 트랜잭션에게 먼저 로크를 허용할 수 있으면서 배타 로크를 요청하는 높은 우선 순위 트랜잭션이 공유 로크를 요청하는 낮은 우선 순위 트랜잭션에 의해 오랫동안 로크를 대기하지 않도록 한다.

예제 3: 트랜잭션 T_1, T_2, T_3, T_4가 동일한 테이블을 접근할 때 로크 허용 순서를 살펴보자. 트랜잭션의 우선 순위는 $T_1 < T_2 < T_3 < T_4$ 순이며, T_2, T_4는 배타 로크를, 그리고 T_1, T_3는 공유 로크를 요청한다.

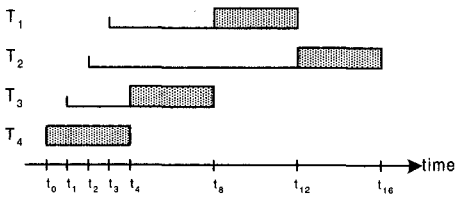


그림 5 로크 대기 트랜잭션의 우선 순위와 상관없는 로크 해제

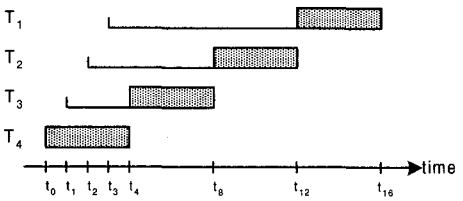


그림 6 로크 대기 트랜잭션의 우선 순위에 따른 로크 해제

그림 5에서 트랜잭션 T_4는 t_0에 배타 로크를 획득하고 트랜잭션 T_3, T_2, T_1 각각 t_1, t_2, t_3에 로크 충돌로 대기 중에 있다. 이때 로크 대기 중인 트랜잭션의 우선 순위를 고려하지 않는다면, 트랜잭션 T_4는 t_4에 수행을 완료하고 테이블에 대한 로크를 해제하면서 공유 로크를 요청하는 T_3와 T_1에게 로크를 허용한다. 이럴 경우 더 높은 우선 순위를 갖는 T_2가 나중에 들어온 낮은 우선 순위의 트랜잭션 T_1이 수행을 완료하고 로크를 해제할 때까지 대기하게 된다. 하지만 실시간 로킹 기법에서는 그림 6에서 보는 바와 같이 대기 중인 트랜잭션 T_2 보다 낮은 우선 순위를 갖는 T_1에게는 로크를 허용하지 않음으로써 T_2의 수행이 T_1에 의해 지연되지 않도록 한다.

3. 실시간 정적 로킹 기법의 구현

3.1 Mr.RT

실시간 정적 로킹 기법은 한국전자통신연구원에서 개발한 주기억장치 실시간 데이타베이스 저장 시스템인 Mr.RT에서 구현하였다. Mr.RT[24]는 주기억장치에 전체 데이타베이스를 저장하며, C++ 언어로 구현된 각각의 관리자에 의해 저장 관리, 트랜잭션 관리, 동시성 제어, 회복 등의 기능을 제공한다. 또한 Mr.RT 서버는 여러 사용자 트랜잭션들의 요청을 동시에 처리할 수 있는 다중 쓰레드 구조를 갖는다. 사용자 트랜잭션은 테이블 생성·삭제, 인덱스 생성·삭제, 트랜잭션 시작·완료·철회, 데이타 접근을 위한 커서 생성·삭제, 데이타 검색·삽입·삭제·갱신 등 액션들의 집합으로 구성된다. 액션은 Mr.RT 서버가 처리하는 작업 단위를 말한다.

그림 7은 Mr.RT 서버의 구조를 나타낸다. 그림에서 보는 바와 같이, 사용자 트랜잭션의 액션은 입력 메시지 큐를 통해 서버에 전달되며, 액션 분배 쓰레드는 그것을 수행 가능한 액션 수행 쓰레드에 할당한다. 그리고 액션 수행 쓰레드는 수행결과를 출력 메시지 큐를 통해 되돌

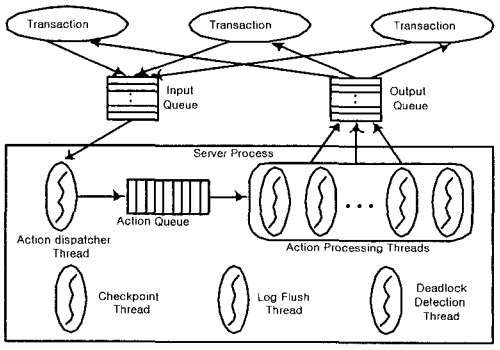


그림 7 Mr.RT 서버 프로세스 구조

```
MODULE Lock_Request(Treq)
BEGIN
    lock mutex;
    conflict := FALSE;
    FOR each relation to be accessed by Treq
        IF Twait exists
            IF (lock granted = No_Lock) and (Prio(Treq) > Prio(Twait))
                increment waiting count of Twait;
            ELSE IF (lock granted = Exclusive) or
                    (lock requested by Treq = Exclusive) or
                    ((lock requested by Treq = Share) and (Prio(Treq) <= Prio(Twait)))
                conflict := TRUE;
                increment waiting count of Treq;
            ENDIF
        ELSE IF (lock granted = Exclusive) or
                ((lock granted = Share) and (lock requested by Treq = Exclusive))
            conflict := TRUE;
            increment waiting count of Treq;
        ENDIF
    ENDFOR

    IF conflict = TRUE
        create lock requester for Treq;
        FOR each relation to be accessed by Treq
            create node of lock requester for Treq;
            insert node of lock requester for Treq into waiting list;
        ENDFOR
        unlock mutex;
        wait lock released;
    ELSE
        FOR each relation to be accessed by Treq
            set lock granted to lock requested by Treq;
            increment holder count;
        ENDFOR
        unlock mutex;
    ENDIF
END
```

그림 8 로크 요청 알고리즘

려준다. 이외에 서버에는 교착상태를 탐지하고 해결하는 교착상태 탐지 쓰레드 그리고 회복 관리를 위해 주기적으로 수행하는 체크포인트 쓰레드와 로그 플러쉬 쓰레드로 구성된다.

Mr.RT에서 처리하는 트랜잭션은 세 가지의 우선 순위(HIGH, MID, LOW)를 가지며, 이러한 트랜잭션들에 대한 스케줄링은 트랜잭션의 액션을 처리하는 액션 수행 쓰레드에 트랜잭션의 우선 순위를 설정하고, 액션 수행 쓰레드들을 설정된 우선 순위에 따라 스케줄링함으로써 이루어진다.

3.2 로크 요청 알고리즘

실시간 정적 로킹 기법의 로크 요청 알고리즘은 트랜잭션이 수행을 시작하는 시점에 접근하는 테이블에 대해 로크 충돌로 대기하거나 또는 로크를 허용 받아 수행을 계속할 것인지를 결정한다.

로크 요청 알고리즘에서는 동일한 테이블을 접근하는 트랜잭션들에 대해 로킹 데이타의 일관성을 유지하기 위하여 쓰레드의 mutex 변수를 이용한다. 즉 mutex 변수에 대한 로크를 먼저 획득한 트랜잭션만이 접근하는 테이블에 대해 로크 요청 알고리즘을 수행하도록 하고, 이 트랜잭션이 로크 충돌로 대기하게 되거나 로크를 허용받아 mutex 변수의 로크를 해제했을 때 다른 트랜잭션이 로크 요청 알고리즘을 수행함으로써 로킹 데이타의 일관성을 보장한다. 다음의 로크 요청 알고리즘에서 T_{req}은 로크를 요청하는 트랜잭션이고, T_{wait}는 로크를 대기하는 트랜잭션들 중에서 가장 높은 우선 순위를 갖는 트랜잭션이다.

3.3 로크 해제 알고리즘

트랜잭션은 수행을 완료하거나 철회할 때 자신이 소유했던 로크를 해제한다. 실시간 정적 로킹 기법에서는 로크 해제 알고리즘에 따라 트랜잭션이 소유했던 모든 테이블에 대한 로크를 해제하고 다음에 로크를 허용할 트랜잭션들을 깨워준다. 로크 해제 알고리즘에서도 로킹 데이타에 대한 일관성을 보장하기 위해 로크 요청 알고리즘에서 사용하는 동일한 쓰레드의 mutex 변수를 이용한다.

로크 해제 알고리즘에서 T_{hold}는 로크를 해제하는 트랜잭션이고, T_{wait}는 로크를 대기하는 트랜잭션들 중에서 우선 순위가 가장 높은 트랜잭션이다. T_{hold}는 먼저 테이블에 대한 로크를 소유한 트랜잭션의 개수를 하나 줄여

```
MODULE Unlock(T_hold)
BEGIN
  FOR each relation to be released by T_hold
    lock mutex;
    decrement holder count;
    IF holder count = 0
      set lock granted to No_Lock;
      IF T_wait exists
        decrement waiting count of T_wait;
        IF waiting count of T_wait = 0
          IF lock requested by T_wait = Exclusive
            FOR each relation to be accessed by T_wait
              delete node of lock requester for T_wait from waiting list;
              set lock granted to lock requested by T_wait;
              increment holder count;
            ENDFOR
            wake up T_wait;
          ELSE
            WHILE lock requested by next T_wait in waiting list = Share
              decrement waiting count of next T_wait;
              IF waiting count of next T_wait = 0
                FOR each relation to be accessed by next T_wait
                  delete node of lock requester for next T_wait from waiting list;
                  set lock granted to lock requested by next T_wait;
                  increment holder count;
                ENDFOR
                wake up next T_wait;
              ENDIF
            ENDWHILE
          ENDIF
        ENDIF
      ENDIF
    ENDIF
    unlock mutex;
  ENDFOR
END
```

그림 9 로크 해제 알고리즘

서 그 값이 0이면, 즉 아무런 트랜잭션도 테이블에 대해 로크를 소유하고 있지 않으면, 테이블에 허용되어 있는 로크를 No_Lock으로 설정한다. 그 다음 로크를 대기하고 있는 트랜잭션 T_{wait}가 있으면, T_{wait}가 로크 충돌로 대기하고 있는 테이블의 개수를 하나 줄여서 그 값이 0이 되면, 이젠 로크 충돌로 대기해야 할 테이블이 없으므로 T_{wait}에게 로크를 허용한다.

4. 성능평가

이 장에서는 실시간 정적 로킹 기법에 대한 성능을 비교·분석한 결과를 기술한다. 성능 비교는 이단계 로킹 기반 실시간 동시성 제어 기법인 2PL-PI와 2PL-HP와 트랜잭션의 순차수행을 대상으로 하였다. 순차수행은 주기억장치 데이타베이스 시스템에서 트랜잭션의 짧은 수행속도로 극단적으로 전체 데이타베이스를 잠금 단위로 함으로써 잠금 연산의 비용을 제거할 수 있다고 제안되었다[18].

성능평가에 사용된 트랜잭션은 데이타베이스에 쓰기 또는 읽기 연산을 수행하며, 트랜잭션의 평균 도착률에 대한 지수분포에 따라 시스템에 도착한다. 시스템에 도착하는 각 트랜잭션은 우선 순위, 접근할 테이블, 읽거나 쓸 레코드 개수 등을 매개변수로 갖는다. 트랜잭션은 마감시간이 지나더라도 수행을 계속하도록 하는 소프트 실시간 트랜잭션을 가정하였다.

표 2 성능평가 매개변수

매개변수	값
데이타베이스 크기	30 relations
트랜잭션이 접근하는 평균 테이블 개수	3 relations
트랜잭션의 평균 예상수행시간	6 ms
트랜잭션의 평균 도착 시간	1~5 sec
트랜잭션의 평균 도착률	2~16 trans./sec.
읽기 전용 트랜잭션의 비율	0 %, 50 %
여유정도	1~8
액션 수행 쓰레드 개수	50

트랜잭션의 마감시간은 어느 정도의 여유정도를 갖는 트랜잭션의 예상수행시간에 트랜잭션이 시스템에 도착한 시간을 더한 값으로, 다음과 같이 계산된다.

마감시간=도착시간+예상수행시간 * 여유정도

트랜잭션의 예상수행시간은 평균 6, 분산 2인 정규 분포를 따르며, 트랜잭션이 각 테이블에 읽거나 쓸 레코드 개수는 예상수행시간에 따라 할당하였다. 트랜잭션이 접근하는 테이블 개수는 평균 3인 지수 분포를 따르며, 실제 접근할 테이블은 균일 분포로 결정하였다. 트랜잭션의 우선 순위는 EDF(earliest deadline first) 방식에 따라 마감시간이 가까울수록 높은 우선 순위가 할당되도록 하였다. 즉 정규분포를 이루는 트랜잭션들의 마감시간 분포를 일정한 비율로 구분하여 마감시간이 긴급한 순으로 Mr.RT의 높은 우선 순위를 할당하였다.

4.1 성능평가 결과

성능평가는 트랜잭션의 평균 도착시간 또는 평균 도착률를 변경하면서 전체 수행되는 트랜잭션의 우선 순위 분포에 따른 트랜잭션의 마감시간 불만족률을 비교하였다. 또한 읽기 트랜잭션의 비율과 트랜잭션 수행의 여유 정도를 변경하면서 마감시간의 불만족률을 비교하였다.

그림 10은 전체 트랜잭션의 우선 순위 분포가 동일할 때, 트랜잭션의 평균 도착시간에 따른 마감시간 불만족률을 나타낸다. 여기서는 트랜잭션 수행시간에 대한 여유 정도를 2로 하였고, 전체 트랜잭션이 삽입연산만을 수행하도록 함으로써 트랜잭션들간의 로크 충돌이 보다 많이 발생하도록 함으로써 비교 대상들 간의 성능 차이를 명확히 보이려 하였다. 전반적으로 RT-SL이 2PL-PI와 2PL-HP 보다도 마감시간을 불만족하는 비율이 적음을 확인할 수 있다. 이는 로킹 연산의 비용이 작고, 트랜잭션이 수행 도중에 로크 충돌로 대기하거나 교착상태로 인해 철회되지 않으면서 수행을 빠르게 완료할 수 있었기 때문이다. 2PL-HP의 경우에는 트랜잭션의 평균 도착시간이 짧을수록 높은 우선 순위를 갖는 트랜잭션에 의해 반복적으로 철회되는 낮은 우선 순위 트랜잭션이 마감시간 내에 수행을 완료하지 못하는 비율이 증가하였다. 그리고 순차수행의 경우에는 트랜잭션의 평균 도착시간이 작을수록 보다 많은 트랜잭션이 수행을 대기하게 됨으로써 마감시간의 불만족률이 증가하였다.

그림 11에서는 전체 수행되는 트랜잭션들 중에서 HIGH 우선 순위를 갖는 트랜잭션의 비율이 50%일 때 마감시간 불만족률을 비교하였다. 여기에서도 트랜잭션 수행시간에 대한 여유 정도를 2로 하였고, 전체 트랜잭션이 삽입연산만을 수행하도록 하였다. 이 경우에도 RT-SL이 2PL-PI와 2PL-HP 보다 근소하게 낮은 마감시간 불만족률을 나타냈다. 그리고 순차수행은 트랜잭션의 우선 순위 분포와 관계없이 트랜잭션의 평균 도착시간이 짧을수록 마감시간 불만족률이 높았다. 한편 2PL-HP는 높은 우선 순위를 갖는 트랜잭션의 비율이 많으므로 수행 중에 반복적인 철회로 인한 마감시간 불만족률이 그림 10에 비하여 감소하였다.

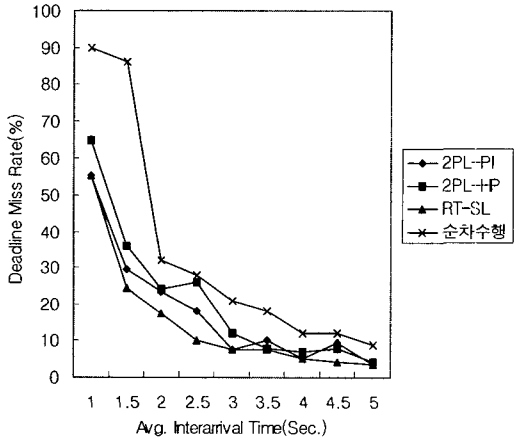


그림 10 트랜잭션의 평균 도착 시간에 따른 마감시간 불만족률(트랜잭션의 우선 순위 분포가 균일할 경우)

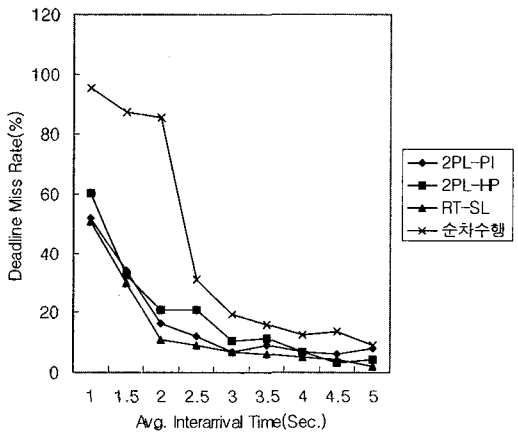


그림 11 트랜잭션의 평균 도착 시간에 따른 마감시간 불만족률(높은 우선 순위(HIGH) 트랜잭션의 분포가 50%일 경우)

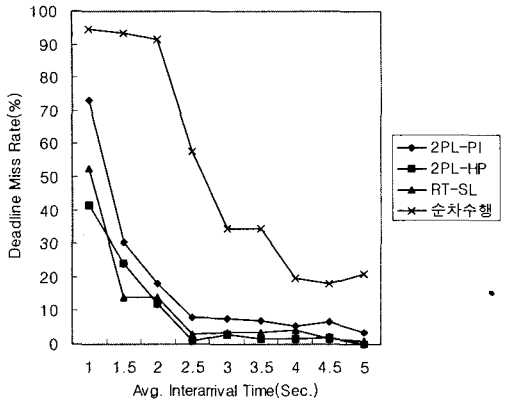


그림 12 트랜잭션의 평균 도착 시간에 따른 마감시간 불만족률(읽기 전용 트랜잭션의 분포가 50%일 경우)

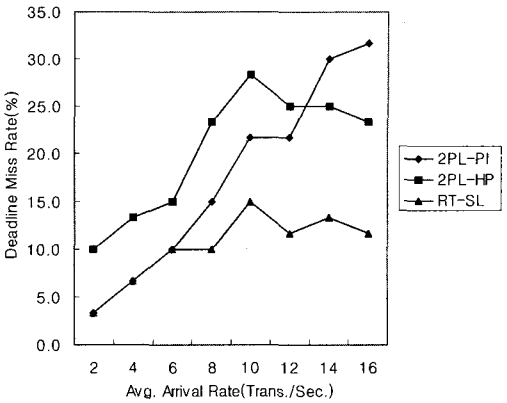


그림 14 트랜잭션의 평균 도착률에 따른 마감시간 불만족률(높은 우선 순위(HIGH) 트랜잭션의 분포가 50%일 경우)

앞의 그림 10과 그림 11은 전체 트랜잭션이 삽입 연산을 수행하도록 함으로써 로크 연산이 트랜잭션의 수행에 보다 많은 영향을 미치도록 하였다. 그러면 읽기 연산만을 수행하는 트랜잭션이 전체 트랜잭션들 중에 50%일 때 마감시간 불만족률은 그림 12와 같다. 이때 트랜잭션 수행시간의 여유정도는 2로 하였고 균일한 우선 순위 분포를 갖도록 하였다. 여기에서는 RT-SL과 2PL-HP가 거의 비슷한 성능을 보였고, 2PL-PI의 경우에는 우선 순위 상속 비용으로 조금 높은 불만족률을 나타냈다. 그리고 순차수행의 경우에는 읽기 전용 트랜잭션의 분포와 관계없이 이전과 비슷한 성능을 보였다.

그림 13과 그림 14는 각각 전체 트랜잭션의 우선 순위 분포가 균일한 경우와 높은 우선 순위(HIGH) 트랜잭션의 비율이 50%일 때, 트랜잭션의 평균 도착률에 따른 마감시간 불만족률을 평가하였다. 이때에도 트랜잭션 수행시간의 여유정도를 2로 하였고 삽입연산만을 수행하도록 하였다. 그리고 동시성 제어 기법들간의 성능을 비교하기 위하여 트랜잭션의 순차수행을 비교 항목에서 제외하였다. 그림 13과 그림 14 모두에서 RT-SL이 2PL-PI와 2PL-HP 보다 마감시간 불만족률이 적음을 확인할 수있다. 이와 같은 결과는 RT-SL이 로킹 연산의 비용이 작고, 트랜잭션이 수행 도중에 로크 충돌로 대기하거나 교착상태로 인해 철회되지 않으면서 수행을 빠르게 완료할 수 있었기 때문이다.

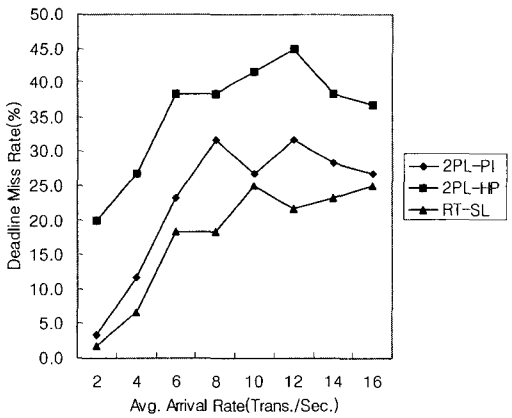


그림 13 트랜잭션의 평균 도착률에 따른 마감시간 불만족률(트랜잭션의 우선 순위 분포가 균일할 경우)

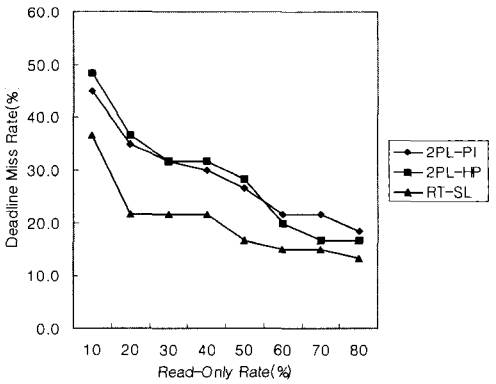


그림 15 읽기 전용 트랜잭션의 비율에 따른 마감시간 불만족률

읽기 전용 트랜잭션은 데이타베이스에 대해 읽기 연산만을 수행하는 트랜잭션으로 전체 트랜잭션에서 읽기 전용 트랜잭션의 비율이 높을수록 로크 충돌이 발생할 가능성이 적고 따라서 그만큼 마감시간을 불만족할 비율이 줄어든다. 그림 15에서는 읽기 전용 트랜잭션의 비율을 변경하면서 마감시간 불만족률을 평가하였다. 여기서 여유정도는 2이고 트랜잭션의 평균 도착률은 12 trans./sec.로 하였다.

그림 15를 살펴보면 RT-SL이 모든 읽기 전용 트랜잭션의 비율에서 2PL-PI와 2PL-HP 보다 마감시간 불만족률이 적었다. 한편, 읽기 전용 트랜잭션의 비율이 증가함에 따라 이들 간의 마감시간 불만족률 차이가 줄어드는 것은 읽기 전용 트랜잭션들이 많을 수록 로크 충돌이 발생할 확률이 감소하여 동시성 제어에 의한 효과가 줄어들기 때문이다.

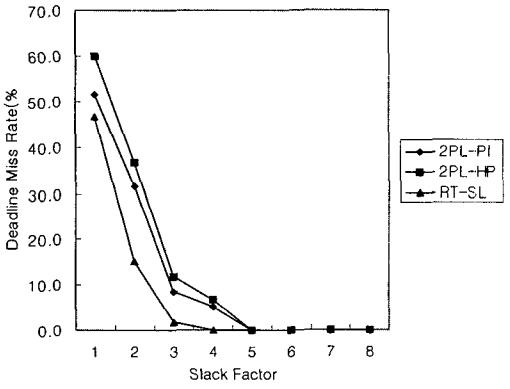


그림 16 트랜잭션 수행시간의 여유정도에 따른 마감시간 불만족률

앞서 언급한 바와 같이 트랜잭션의 마감시간은 어느 정도의 여유정도를 갖는 트랜잭션의 예상수행시간에 트랜잭션이 시스템에 도착한 시간을 더하여 계산한다. 따라서 트랜잭션 수행시간의 여유정도가 클 수록 트랜잭션의 마감시간은 길어지므로 트랜잭션이 마감시간을 넘길 확률이 줄어든다. 그림 16은 이러한 트랜잭션 수행시간의 여유정도를 증가시키면서 그에 따른 마감시간 불만족률을 비교하였다. 이를 위해 트랜잭션은 삽입연산을 하도록 하였고, 트랜잭션의 평균 도착률은 12 trans./sec.로 설정하였다. 그림 16를 보면, 세 가지 기법들의 마감시간 불만족률이 트랜잭션 수행시간의 여유정도가 증가할 수록 현저히 줄어듬을 볼 수 있다. 또한 RT-SL의 마감시간 불만족 비율이 2PL-PI와 2PL-HP 보다 적게 나타났다.

5. 결 론

주기억장치 데이타베이스 시스템은 데이타 접근시에 디스크 입출력이 발생하지 않기 때문에 트랜잭션 수행시간이 매우 짧고 로크 충돌이 발생할 가능성이 적다. 따라서 디스크 기반의 로킹 알고리즘을 사용할 경우 로킹 연산의 비용이 전체 트랜잭션 수행에 커다란 부담이 된다. 디스크 기반 데이타베이스 시스템에서 설계된 이단계 로킹 기법을 기반으로 하는 기존 실시간 동시성 제어 기법들을 주기억장치 데이타베이스 시스템에 적용할 경우, 복잡한 로크 테이블 조작으로 인하여 로킹 연산의 부하가 크며, 트랜잭션이 수행 도중에 로크 충돌로 대기하거나 교착상태로 철회됨으로써 마감시간내에 수행을 완료하지 못하게 된다. 따라서 이 논문에서는 주기억장치 데이타베이스 시스템에 적합한 실시간 동시성 제어 기법으로 실시간 정적 로킹 기법을 설계하고 구현하였다.

실시간 정적 로킹 기법은 트랜잭션이 수행을 시작할 때 접근하는 모든 데이타 객체에 대해 로크를 획득해야 하는 정적 로킹 기법을 기반으로 하므로 트랜잭션이 수행 중간에 로크 충돌로 대기하거나 교착상태로 철회되지 않고 수행을 완료할 수 있다. 또한 로킹 연산의 비용을 줄이기 위해 로킹 단위의 데이타 객체에 로킹 정보를 효율적으로 유지하고 관리할 수 있도록 구현하였다. 실시간 정적 로킹 기법의 로크 요청 알고리즘은 로크를 대기하고 있는 트랜잭션의 우선 순위가 로크를 요청하는 트랜잭션의 우선 순위 보다 높을 때, 로크 충돌로 간주함으로써 높은 우선 순위를 갖는 로크 대기 트랜잭션이 낮은 우선 순위를 갖는 트랜잭션에 의해 수행이 지연되지 않도록 구현하였다. 또한 로크를 해제할 때 로크를 대기하고 있는 트랜잭션들 중에서 우선 순위가 가장 높은 트랜잭션에게 먼저 로크를 허용하도록 하는 로크 해제 알고리즘을 구현하였다.

실시간 정적 로킹 기법은 주기억장치 실시간 데이타베이스 시스템인 Mr.RT에서 구현하였으며, 이단계 로킹 기반 실시간 동시성 제어 기법인 2PL-PI와 2PL-HP도 구현하여 이들 간의 성능을 비교·평가하였다. 성능평가 결과에서 실시간 정적 로킹 기법은 모든 경우에 마감시간을 불만족하는 비율이 적었으며, 다음으로 2PL-PI가 우선 순위 상속의 효과로 낮은 우선 순위(LOW) 트랜잭션의 반복적인 철회가 발생하는 2PL-HP 보다 마감시간 불만족률이 적었다.

향후 연구해야 할 과제로는 주기억장치 실시간 데이타베이스 시스템의 응용환경을 고려한 워크로드하에서 성능을 평가함으로써 실제 응용분야에 적용할 수 있음

을 보이고 또한 실시간 낙관적 동시성 제어 기법을 주기억장치 데이타베이스 시스템에 구현하고 이와의 성능 비교를 수행하는 것이다.

참 고 문 헌

[1] P. S. Yu, K. L. Wu, K. L. Lin, and S. H. Son, "On Real-Time Databases: Concurrency Control and Scheduling," Proc. of the IEEE Vol. 82, No. 1, 1994.

[2] 실시간 데이타베이스 특집, 한국정보과학회지, 제11권, 제1호, 1993.

[3] 황규영, 장지웅, 이영구, 김원영, "주기억 장치 데이타베이스를 위한 저장 시스템," 한국정보과학회지, 제14권, 제2호, 1996.

[4] R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," ACM Trans. on Database Systems, Vol. 17, No. 3, 1992.

[5] Ö. Ulusoy and A. Buchmann, "A Real-Time Concurrency Control Protocol for Main-Memory Database Systems," Information Systems, Vol. 23, No. 2, 1998.

[6] P. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987.

[7] K. Y. Lam, S. L. Hung and S. H. Son, "On Using Real-Time Static Locking Protocols for Distributed Real-Time Databases," Journal of Real-Time Systems, Vol. 13, No. 2, 1997.

[8] 차상균, 박장호, 박병대, 이성직, "M^{2}RTSS: 주메모리 실시간 저장 시스템," 한국정보과학회지, 제14권, 제2호, 1996.

[9] A. Bestavros and S. Braoudakis, "SCC-nS: A Family of Speculative Concurrency Control Algorithms for Real-Time Databases," Proc. Third Int'l Workshop Responsive Computer Systems, 1993.

[10] J. R. Haritsa, M. J. Carey, and M. Livny, "On Being Optimistic about Real-Time Constraints," Proc. ACM Symp. Principles of Database Systems, 1990.

[11] J. R. Haritsa, M. J. Carey, and M. Livny, "Dynamic Real-Time Optimistic Concurrency Control," Proc. IEEE Real-Time Systems Symp., 1990.

[12] J. Huang, J. Stankovic, K. Ramamritham, and D. Towsley, "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes," Proc. 17th Int'l Conf. Very Large Data Bases, 1991.

[13] J. Huang, J. Stankovic, D. Towsley, and K. Ramamritham, "Experimental Evaluation of Realtime Transaction Processing," Proc. IEEE Real Time Systems Symp., 1989.

[14] J. Lee and S. H. Son, "Concurrency Control Algorithms for Real-Time Database Systems," Performance of Cincurrency Control Mechanisms in Centralized Database Systems, 1995.

[15] J. Lee and S. H. Son, "Using Dynamic Adjustment of Serialization Order for Real-Time Database Systems," Proc. IEEE Real-Time Systems Symp., 1993.

[16] Y. Lin and S. H. Son, "Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order," Proc. IEEE Real-Time Systems Symp., 1990.

[17] A. Datta and S. H. Son, "A Study of Concurrency Control in Real-Time, Active Database Systems," IEEE Trans. on Knowledge and Data Engineering, Vol. 14, No. 3, 2002.

[18] H. Garcia-Molina and K. Salem, "Main Memory Database Systems: An Overview," IEEE Trans. on Knowledge and Data Engineering, Vol. 4, No. 6, 1992.

[19] J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993.

[20] J. Huang, J. Stankovic, K. Ramamritham and D. Towsley, "On Using Priority Inheritance In Real-Time Databases," IEEE Real-Time Systems Symp., 1991.

[21] B. Kao and H. Garcia-Molina, "An Overview of Real-Time Database Systems," Advances in Real-Time Systems, pp. 463-486, Prentice Hall, 1995.

[22] L. Sha, R. Rajkumar and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE Trans. on Computers, Vol. 39, No. 9, 1990.

[23] B. Stroustrup, The C++ Programming Language, Addison-Wesley, 1991.

[24] Sun Microsystems, inc., Multithreaded Programming Guide, 1994.

[25] Ö. Ulusoy and A. Buchmann, "Exploiting Main Memory DBMS Features to Improve Real-Time Concurrency Control Protocols," ACM SIGMOD Record, Vol. 25, No. 1, 1996.

김 영 철

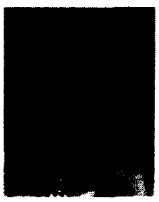
1995년 강원대학교 전자계산학과 졸업(학사). 1999년 강원대학교 전자계산학과 졸업(석사). 2000년 ~ 현재 한국전자통신연구원 연구원. 관심분야는 주기억장치 실시간 데이타베이스, 클러스터 기반 데이타베이스, 저장 시스템

유 한 양

1991년 강원대학교 전자계산학과 졸업(학사). 1995년 강원대학교 전자계산학과 졸업(석사). 1999년 강원대학교 전자계산학과 박사 수료. 2000년 ~ 현재 한국전자통신연구원 선임연구원. 관심분야는 주기억장치 데이타베이스, 실시간 데이타베이스 시스템, 이중화 시스템

김 진 호

1982년 경북대학교 전자공학과 졸업(학사). 1985년 한국과학기술원 전산학과 졸업(석사). 1990년 한국과학기술원 전산학과 졸업(박사). 1990년 ~ 1994년 삼성종합기술원 자문교수. 1995년 ~ 1996년 미국 미시간 대학교 객원 교수. 1990년 ~ 현재 강원대학교 전자계산학과 교수. 관심 분야는 주기억장치 실시간 데이타베이스, 내장형 시스템, 웹 데이타베이스, XML, OLAP, 데이타웨어하우징 등



김 준

1983년 부산대학교 계산통계학과 졸업(학사). 1986년 한국과학기술원 전산학과 졸업(석사). 1986년 ~ 현재 한국전자통신연구원 책임연구원. 관심분야는 멀티미디어 데이타베이스, 클러스터 기반 데이타베이스

서 상 구

1984년 서울대학교 컴퓨터공학과(학사). 1986년 한국과학기술원 전산학과(석사). 1995년 한국과학기술원 전산학과(박사). 1986년 ~ 1989년 현대전자 소프트웨어 사업부. 1995년 ~ 1998년 현대전자 소프트웨어연구소. 1999년 ~ 현재 광운대학교 경영정보학과 조교수. 관심분야는 데이타베이스 최적화, 데이타 웨어하우징, 웹 데이타 관리, 등