실행가능성검사를 이용한 효율적인 실시간 동시성제어알고리즘

(An Efficient Real-Time Concurrency Control Algorithm using the Feasibility Test)

이 석 재 † 신 재 룡 † 송 석 일 †
(Seok Jae Lee) (Jae Ryoung Shin) (Seok Il Song)
유 재 수 †† 조 기 형 †† 이 병 엽 †††
(Jae Soo Yoo) (Ki Hyung Cho) (Byoung Yeop Lee)

요 약 실시간 데이타베이스 시스템에서 높은 우선 순위를 갖는 트랜잭션(High Priority Transaction; HPT)의 선행 처리를 보장하기 위해 2PL-HP(Two Phase Locking with High Priority) 방법이 사용된다. 이 방법은 충돌 발생 시 낮은 우선 순위를 갖는 트랜잭션(Low Priority Transaction; LPT)을 철회 또는 대기시킴으로써 충돌을 해결한다. 그러나 HPT가 마감시간을 지키지 못해서 시스템에서 제거되는 경우에는 LPT의 불필요한 철회 및 대기가 발생된다. 최근에 이러한 문제를 해결하고자 대체버전을 이용한 동시성 제어 알고리즘이 제안되었다. 그러나 이 알고리즘은 항상 대체 버전을 생성해야하며, 또한 복잡한 대체 버전을 관리하기 위한 기법이 추가적으로 요구된다. 본 논문에서는 불필요한 자원의 낭비를 막고 LPT의 불필요한 철회 및 대기를 제거할 수 있는 효율적인 동시성제어 알고리즘을 제안한다. 그리고 성능평가를 통해 제안하는 동시성 제어 알고리즘이 기존의 대체버전을 이용한 동시성제어 알고리즘에 비해 성능이 우수함을 보인다.

키워드 : 실시간데이타베이스, 동시성제어, 실행가능성 검사, 트랜잭션처리

Abstract The 2PL-HP(Two Phase Locking with High Priority) method has been used to guarantee preceding process of a high priority transaction(HPT) in real-time database systems. The method resolves a conflict through aborting or blocking of a low priority transaction(LPT). However, if HPT is eliminated in a system because of its deadline missing, an unnecessary aborting or blocking of LPT is occurred. Recently, to resolve the problem, a concurrency control algorithm using alternative version was proposed. However, the algorithm must always create the alternative version and needs an addtional technique to manage complex alternative versions. In this paper, we propose an efficient concurrency control algorithm that prevents needless wastes of resources and eliminates unnecessary aborting or blocking of LPT. And it is shown through the performance evaluation that the proposed concurrency control algorithm outperforms the existing concurrency control algorithm using alternative version.

Key words : Real-time Database, Concurrency Control, Feasibility Test, Transaction Processing

· 본 연구는 한국과학재단 목적기초연구(특정기초연구 과제번호 R01-1999-0024)와 2001년도 한국학술진흥재단(KRF-2001-041-E00233)의 지원으로 수행되었음.

† 비 회 원 : 충북대학교 정보통신공학과
sjlee@netdb.chungbuk.ac.kr
jrshin@netdb.chungbuk.ac.kr
prince@netdb.chungbuk.ac.kr
†† 종신회원 : 충북대학교 전기전자 및 컴퓨터 공학부 교수
yjs@cbucc.chungbuk.ac.kr
khjoe@cbucc.chungbuk.ac.kr
††† 비 회 원 : 대우정보시스템 CRM사업팀
bylee@disk.co.kr

논문접수 : 2001년 9월 1일
심사완료 : 2002년 5월 9일

1. 서 론

실시간 데이타베이스 시스템(real-time database system; RTDBS)에서 수행되는 트랜잭션들은 시작된 후 어느 시점까지는 반드시 작업이 끝나야 한다는 시간 제약조건을 갖는다[1,2]. 이것을 마감시간(deadline)이라 한다. 대표적인 실시간 데이타베이스 시스템 응용분야로는 항공, 무기체계, 공장 자동화, 로보틱스, 원자력 발전소, 교통 제어 시스템, 네트워크 서비스 또는 주식 시장 응용 등이 있다[3,4,5,6,7]. 실시간 데이타베이스에서 수

행되는 트랜잭션들은 하드(hard), 펌(firm), 소프트(soft) 실시간 트랜잭션으로 분류된다[8,9,10]. 시간제약조건이 반드시 준수되어야 하는 트랜잭션은 하드 실시간 트랜잭션이라 한다. 하드 실시간 트랜잭션은 만약 마감시간을 지키지 못하는 경우, 시스템에서 즉시 제거되어야 한다. 이 때 작업 결과는 치명적인 손실을 초래한다. 펌 실시간 트랜잭션은 마감시간을 지키지 못하는 경우 단지 작업 결과가 무의미해지며 치명적인 손실은 초래하지 않는다. 소프트 실시간 트랜잭션은 마감시간을 초과하더라도 결과의 가치가 다소 떨어질 뿐이며, 치명적인 손실도 초래하지 않는다. 일반적으로 치명적인 손실을 가져올 수 있는 하드 실시간 트랜잭션은 반드시 모든 작업이 마감시간 내에 처리될 수 있어야 하며[11,12,13], 펌 또는 소프트 실시간 트랜잭션들은 마감시간 초과 비율이 최소화되도록 하여야 한다[14,15,16].

최근까지 연구된 동시성 제어 방법들은 크게 2단계 잠금(2PL) 방식을 사용하여 충돌 발생 시 검사를 실시하는 비관적 동시성 제어(Pessimistic Concurrency Control) 방법과 충돌 검사 및 재시작을 완료 직전에 수행하는 낙관적 동시성 제어(Optimistic Concurrency Control; OCC) 방법으로 구분된다. 두 방법에 대한 성능 비교는 여러 논문에서 실시되었으며 실험 환경에 따라 결과에 다소 차이를 보이고 있다[17,18,19,20,21].

일반적으로 실시간 데이타베이스 시스템에서 많이 사용되는 2PL-HP(Two Phase Locking-High Priority) 방법은 높은 우선순위 트랜잭션(High Priority Transaction; HPT)의 선행 처리를 항상 보장하기 때문에 낮은 우선순위 트랜잭션(Low Priority Transaction; LPT)의 철회 및 대기가 불가피하다. 만약 HPT가 마감시간을 지키지 못하는 경우 LPT의 재시작 및 대기는 불필요한 일이 되어 버리므로 자원의 낭비를 초래한다. 이러한 문제를 해결하고자 최근에 제안된 방법이 대체버전을 이용한 동시성 제어 방법[23,26]이다. 이 방법은 항상 대체 버전을 생성시켜서 상황에 따라 유리한 버전을 실행시키도록 한다. 그러나 항상 두 개의 버전을 유지하면서 결국 하나의 버전만이 사용되기 때문에 자원의 낭비를 초래할 수밖에 없다. 또한 대체버전을 이용하여 수행하는 도중 또다른 충돌이 발생되어 제2, 제3의 대체버전이 생길 수 있으므로 이를 관리하기 위해 계층적 구조를 갖는 복잡한 버전 관리방법을 필요로 하고 있다.

본 논문에서는 대체버전을 이용한 동시성 제어 방법에서 사용되던 불필요한 대체버전을 제거하면서 LPT의 불필요한 재시작 및 불필요한 기다림을 방지할 수 있는 방법을 제안한다. 또한 상대적으로 LPT인 비실시간 트랜잭션 및 소프트 실시간 트랜잭션의 처리 효율을 증대시키고 전체적인 마감시간 초과 비율을 최소화하는 방법을 함께 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 실시간 동시성 제어 방법들에 대한 기존의 연구들을 살펴본다. 3장에서는 제안하는 동시성제어 방법에 대해 상세히 기술하고, 4장에서 성능평가 결과를 보인다. 그리고 마지막 5장에서 결론을 맺고 향후 연구방향을 제시한다.

2. 관련 연구

동시성 제어 방법에 대한 기존의 연구들은 충돌 검사와 재시작이 충돌 발생 시점에 수행되는지, 아니면 완료 직전에 수행되는지에 따라 크게 2PL(Two Phase Locking)에 기반한 비관적 동시성 제어 방법과 낙관적 동시성 제어 방법(OCC)으로 나눌 수 있다.

2.1 비관적 동시성 제어 방법

일반 데이타베이스 시스템에서 사용하는 2PL 방법은 우선 순위 역전(Priority Inversion) 및 교착 상태(Deadlock)가 발생한다는 문제점을 갖는다. 즉, 우선 순위를 전혀 고려하지 않고 있기 때문에 실시간 데이타베이스 시스템에는 적합하지 못한 동시성 제어 방법이다. 2PL방법의 문제점을 개선한 2PL-HP 방법은 충돌 해결을 위해 (그림 1)에서와 같이 항상 LPT를 취소 또는 대기시키고 HPT를 수행시킨다. 이 방법은 HPT의 선행처리를 항상 보장하기 때문에 우선 순위에 따른 수행이 필수적인 실시간 응용에 적합한 방법이다.

```
If pr(T_H) < pr(T_R) then {  // pr(X) : 트랜잭션 X의 우선순위
    Abort  T_H;              // T_H : 자원을 사용중인 트랜잭션
    Run    T_R;              // T_R : 자원을 요청하는 트랜잭션
}
else {
    T_R  blocks;
    Run  T_H;
}
```

그림 1 2PL-HP 방법

그러나 2PL-HP 방법에서 최소 여유시간 우선 방식(Least Slack First)을 사용하여 우선 순위를 할당하는 경우 트랜잭션의 연속적인 철회 및 재시작 문제가 발생한다. 최소 여유시간 우선 방식을 사용하면 충돌하여 재시작 된 LPT는 수행시간이 0(zero)이 되므로 그만큼 여유시간이 줄어들게 된다. 따라서 LPT는 높은 우선

순위를 재 할당받게 된다. 이 때 LPT가 자신을 재시작시킨 HPT보다 높은 우선 순위를 재 할당 받게되면 연속적인 철회 및 재시작이 발생하게 된다. 이 문제는 (그림 2)와 같이 충돌 검사 시점에서 LPT의 재시작 후의 우선 순위를 미리 계산하여 충돌 해결 과정에 반영하는 방법[14]에 의해 해결할 수 있다.

```
// T^A : 취소 후 재시작한 트랜잭션
// pr(T_H^A) : 재시작 후에 재 할당받을 우선순위
If pr(T_H) < pr(T_R) AND pr(T_H^A) < pr(T_R) then {
     Abort  T_H;
     Run    T_R;
}
else {
     T_R  blocks;
     Run  T_H;
}
```

그림 2 재시작 트랜잭션의 우선 순위를 고려한 2PL-HP 방법

조건부 재시작(Conditional Restart; CR) 방법은 (그림 3)과 같이 충돌 발생 시 현재 실행중인 LPT의 남아있는 수행시간이 HPT의 여유시간보다 짧은 경우, 현재 수행중인 LPT가 계속 수행되도록 HPT가 대기한다. 이 방법은 앞서 실행중인 LPT의 남은 수행 시간과 대기중인 HPT의 여유 시간을 정확하게 예측할 수 있어야 정확한 결과를 얻을 수 있는 단점을 갖는다. 또한 계속 수행중인 LPT가 제3의 트랜잭션과 충돌되어 철회되는 경우, 대기중인 HPT는 자원을 점유하지 못하고 빼앗기게 된다.

```
If pr(T_H) < pr(T_R) AND pr(T_H^A) < pr(T_R) then {
     If (estimated remaining time of T_H) ≤(Slack time
of T_R) then {
          T_R  blocks;
          T_H  inherits the priority of T_R;
          Run T_H;
     }
     else {
          Abort T_H;
          Run   T_R;
     }
}
else {
     T_R   blocks;
     Run  T_H;
}
```

그림 3 조건부 재시작 방법

우선 순위 상속(Priority Inheritance; PI) 또는 Wait Promote 방법은 트랜잭션 철회 비용이 아주 큰 경우에 유용한 방법으로, 앞서 수행중인 트랜잭션의 계속적인 수행을 보장해 주는 방법이다. 앞서 수행중인 LPT와 HPT간에 충돌이 발생하면 HPT는 대기하도록 하고, 대기중인 HPT의 우선 순위를 수행중인 LPT에게 상속해 준다. LPT가 완료되면 대기하던 HPT가 해당 자원을 넘겨받아 처리하게 된다. 이 방법은 HPT의 대기 시간이 불필요하게 길어질 수 있기 때문에 자원 점유 시간에 대한 예측이 가능한 경우를 제외하고는 적용하기 힘들다.

2.2 낙관적 동시성 제어 방법

트랜잭션의 처리 과정은 크게 읽기 단계(Read Phase), 검사 단계(Validation Phase) 그리고 쓰기 단계(Write Phase)의 3단계로 구성된다. 읽기 단계에서는 트랜잭션의 실행에 필요한 데이타를 로컬 영역으로 복사한 후 처리한다. 검사 단계에서는 트랜잭션의 충돌 검사를 실시한다. 검사 단계에서의 충돌 검사 방법은 이미 완료된 트랜잭션들을 대상으로 충돌 여부를 판별하는 역방향 검사(Backward Validation) 방법과 현재 수행중인 트랜잭션들을 대상으로 충돌 여부를 판별하는 순방향 검사(Forward Validation) 방법이 있다. 전자의 경우에는 충돌 발생 시 항상 충돌 검사를 실시중인 트랜잭션이 철회된다. 후자의 경우에는 충돌 해결 방법에 따라 철회되는 트랜잭션이 다를 수 있다. 이때 검사 단계의 트랜잭션과 읽기 단계의 트랜잭션들 중에 어떤 트랜잭션들을 철회할 것인지를 결정하는 방법은 OPT-BC (Broadcast Commit), OPT-Sacrifice, OPT-Wait, Wait-50, Wait-X 등으로 나누어진다[8,15,17]. 쓰기 단계에서는 완료가 보장된 트랜잭션에 대해 로컬 영역의 데이타를 데이타베이스에 반영한다.

OPT-BC 방법은 검사 단계의 트랜잭션이 충돌되는 다른 트랜잭션들과 비교하여 가장 낮은 우선 순위를 갖지만 않는다면 완료를 보장하고, 나머지 충돌되는 트랜잭션들을 재시작 시킨다. 따라서 읽기 단계에 있는 HPT가 LPT에 의해 재시작 될 수 있으므로 스케줄링 시 HPT가 검사 단계에 빠르게 도착할 수 있도록 고려해주어야 한다.

OPT-Sacrifice 방법에서는 검사 단계 트랜잭션은 충돌하는 상위 트랜잭션(Conflict Higher Priority; CHP)이 있는 경우 재시작 된다. 즉, 검사 단계에 진입한 트랜잭션의 우선 순위가 충돌하는 트랜잭션들 중에서 가장 높은 경우에만 완료가 보장된다.

OPT-Wait(Wait-50, Wait-X) 방법은 검사 단계의

LPT가 불필요하게 재시작되는 것을 줄이기 위한 방법이다. 즉, 검사단계의 LPT는 CHP가 읽기 연산만을 갖는다면 끝나기를 기다린 후 더 이상 CHP가 존재하지 않을 때 완료를 보장받는다. 그러나 대기하는 동안 또 다른 CHP가 추가되는 경우 대기시간이 길어지거나 재 시작될 수 있다.

Wait-50 방법은 CHP가 50% 미만인 경우에 완료가 보장되고, 그렇지 않은 경우 재시작 된다. 이렇게 함으로써 중·상위 우선 순위를 갖는 트랜잭션들의 무조건적인 재시작을 줄일 수 있다.

Wait-X 방법은 Wait-50 방법을 50% 뿐만 아니라 시스템 환경에 따라 30% 또는 70% 등으로 다양하게 변화 가능하도록 확장한 방법이다.

지금까지 살펴본 OCC 방법들은 트랜잭션의 처리가 완료된 이후에 검사 단계에서 충돌 검사 및 해결이 이루어지므로 블로킹(blocking)이 발생하지 않는다. 그러나 충돌을 재시작으로만 해결하기 때문에 블로킹과 재시작을 적절히 이용하는 2PL-HP 방법에 비해 불필요한 재시작이 많다. 따라서 OCC 방법은 자원 경쟁이 적은 경우에 유리하다.

2.3 대체 버전을 이용한 동시성 제어 방법

폼 실시간 데이타베이스에서는, OCC 방법의 단점인 낭비적인 수행보다 2PL-HP 방법의 단점인 낭비적인 재시작과 기다림의 문제가 시스템에 더 큰 영향을 미친다[21,23]. 낭비적인 수행은 OCC 방법에서 LPT가 HPT에 의해 검사단계에서 재시작되는 경우에 발생하는 문제점이다. 낭비적인 재시작은 LPT를 재시작 시킨 HPT가 마감시간을 지키지 못하고 시스템에서 제거되는 경우이다. 낭비적인 기다림은 마감시간을 지키지 못하는 HPT에 의해 LPT가 불필요하게 대기하는 것이다.

대체버전을 이용한 동시성 제어 방법[23,26]에서는 OCC 방법의 보류된 갱신을 채택하여 HPT가 LPT를 즉시 재시작 시키지 않으면서 계속 진행할 수 있도록 하고있다. 즉, HPT가 LPT와 충돌하는 경우 LPT를 잠시 정지시켰다가 HPT가 비정상적으로 종료할 경우, LPT를 다시 재개시키는 것이다. 이 방법을 정지/재개 방법이라 한다. (그림 4)와 같이 앞서 수행 중이던 LPT가 잠금을 요구하는 HPT와 충돌하게되면, LPT는 즉시 재시작(Immediately Restart; IR) 버전과 정지/재개(Deferred Restart; DR) 버전을 동시에 유지한다. LPT는 HPT의 수행 결과에 따라 HPT가 정상적으로 완료되는 경우 IR 버전을 이용하고, 비정상적으로 완료되는 경우 DR 버전을 이용하여 수행을 계속하는 방식이다. 따라서 두 가지 버전을 동시에 유지하면서 상황에 따라 적절한 버전을 이용함으로써 낭비적인 재시작 및 낭비적인 수행을 줄일 수 있다.

이 방법은 혼합 트랜잭션을 지원하기 위해 소프트 실시간 트랜잭션의 경우에는 2PL-HP 방법을 사용하고, 폼 실시간 트랜잭션의 경우에는 블로킹과 즉시 재시작 및 정지/재개 버전을 필요에 따라 사용할 수 있도록 하고 있다. 그러나 충돌이 발생할 경우 LPT는 항상 두 개의 버전을 유지하게 되므로 불필요한 자원의 낭비가 생길 수 있다. 또한, 즉시 재시작 버전의 수행 시 다른 HPT와의 충돌이 발생할 경우 제2, 제3의 즉시 재시작 버전이 생성되므로 계층적 구조의 복잡한 버전관리 방법이 추가적으로 요구된다.

```
If (HPT conflict with LPT) then {
     Run HPT;
     Run LPT's IR version;
     Waiting for Running LPT's DR version;
}

If (HPT commit) then
     Run LPT's IR version;
else
     Run LPT's DR version;
```

그림 4 대체버전을 이용한 동시성 제어 방법

3. 제안하는 실시간 동시성제어 알고리즘

본 장에서는 대체버전을 이용한 동시성제어 방법에서 생성되는 LPT의 두 가지 버전 중에서 불필요한 하나의 버전을 제거하면서 2PL-HP방법의 단점인 LPT의 불필요한 재시작 및 불필요한 기다림을 방지할 수 있는 방법을 제안한다. 또한 상대적으로 LPT인 비실시간 트랜잭션 및 소프트 실시간 트랜잭션의 처리 효율을 증대시키고 전체적인 마감시간 초과 비율을 최소화하는 방법을 제안한다.

3.1 개요

제안하는 실시간 동시성 제어 알고리즘에서는 폼 실시간 트랜잭션들간에 충돌이 발생하면 항상 HPT의 마감시간 보장 여부를 판단한다. 이것을 실행가능성 검사(Feasibility Test)라 한다. (그림 5)와 같이 앞서 수행중인 HPT가 마감시간을 지키지 못한다고 판단된 경우, 자원을 요청하는 LPT의 불필요한 기다림을 방지하기 위해 HPT를 시스템에서 제거시키고 LPT가 해당 자원을 사용할 수 있게 한다. 또한 자원을 요청하는 HPT에 대해서도 실행가능성 검사를 실시하여 HPT가 마감시간을 지키지 못한다고 판단된 경우, HPT를 시스템에서 제거시켜 앞서 수행중인 LPT의 계속적인 수행을 보장하여 LPT의 불필요한 재시작을 방지한다.

```
//TH의 우선 순위가 TR보다 낮은 경우
if pr(TH) < pr(TR) then {
    if (TR is feasible) then {
        Abort TH;
        Run   TR;
    }
    else
        Abort TR;
}
//TH의 우선 순위가 TR보다 높은 경우
else {
    if (TH is feasible) then
        TR blocks;
    else {
        Abort TH;
        Run   TR;
    }
}
```

pr(X): 트랜잭션 X의 우선순위
T_H: 자원을 사용중인 트랜잭션
T_R: 자원을 요청하는 트랜잭션

그림 5 제안하는 방법의 기본 알고리즘

또한 상대적으로 LPT인 비실시간 및 소프트 실시간 트랜잭션이 HPT인 펌 실시간 트랜잭션과 충돌할 경우 무조건적으로 재시작 또는 기다리도록 하는 기존의 방법과는 달리 (그림 6)과 같이 HPT의 실행가능성 검사를 통해 HPT가 마감시간을 지키지 못한다고 판단된 경우, HPT를 시스템에서 제거시키고 LPT인 비실시간 및 소프트 실시간 트랜잭션이 자원을 사용하게 한다. 따라서 LPT의 불필요한 재시작 및 기다림을 방지하여 자원의 낭비를 줄이고 시스템의 처리율 향상을 꾀할 수 있다. 그리고 소프트 실시간 트랜잭션의 경우 마감시간을 초과하더라도 가치가 다소 떨어질 뿐 그 결과는 유효하기 때문에 비실시간 또는 소프트 실시간 트랜잭션들끼리 충돌이 발생하는 경우에는 선행 처리중인 트랜잭션이 비록 LPT라 하더라도 계속 수행시켜 처리율의 향상을 가져올 수 있도록 하였다.

```
// Soft Tr의 수행 중에 Firm Tr이 해당 자원을 요구
if p(TH^soft) < p(TR^Firm) then {
    if (TR^Firm is feasible) then {
        Abort TH^soft;
        Run   TR^Firm;
    }
    else Abort TR^Firm;
}
else
    Error;

// Firm Tr의 수행 중에 Soft Tr이 해당 자원을 요구
if p(TH^Firm) < p(TR^Soft) then
    Error;
else {
    if (TH^Firm is feasible) then
        TR^Soft blocks;
    else {
        Abort TH^Firm;
        Run   TR^Soft;
    }
}
```

T_H^{soft}: 자원을 사용중인 Soft 실시간 트랜잭션
T_R^{Firm}: 자원을 요청하는 Firm 실시간 트랜잭션
T_H^{Firm}: 자원을 사용중인 Firm 실시간 트랜잭션
T_R^{Soft}: 자원을 요청하는 Soft 실시간 트랜잭션

그림 6 혼합 트랜잭션을 위한 알고리즘

3.2 실행가능성 검사(Feasible Test)

과거에 제안되었던 많은 실행 가능성 검사방법들은 트랜잭션이 접근할 데이타 아이템의 목록을 미리 알고 있는 하드 실시간 환경에서 데이타 처리 시간, I/O 시간, 데이타 잠금 획득 및 해제에 필요한 시간, 스케쥴링 시간등을 계산하여 해당 트랜잭션 실행에 필요한 시간을 계산하고 있다.[27,28,29] 그러나 이러한 방법들은 트랜잭션이 접근할 데이타 아이템 목록을 정확하게 미리 알고 있어야 하며, 실행시간 계산에 필요한 연산들의 정확한 소요 시간을 알 수 있어야만 계산이 가능하다. 따라서 실제 환경에 적용하기에는 많은 어려움이 있다.

본 논문에서는 과거에 제안되었던 실행 가능성 검사 방법들과 달리 트랜잭션의 실행 가능성 검사에 사용되는 예상 실행 시간은 과거의 실행 시간 결과로부터 통계적인 방법으로 계산한 평균값을 사용한다. 이 평균값에는 과거의 트랜잭션이 접근한 데이타 아이템의 수, 실행시의 시스템 부하, 충돌 발생 정도등에 의한 지연시간등이 모두 포함 되어 있기 때문에 트랜잭션의 실행시간 계산에 필요한 각각의 연산 시간을 정확하게 구할 수 없는 실제 시스템에 쉽게 적용하여 사용할 수 있는 방법이다.

본 논문에서는 실행 가능성 검사를 위해 사용하는 애플리케이션이 제공하는 기능에 따라 발생 가능한 트랜잭션의 종류는 무한하지 않고, 일정 개수만큼의 트랜잭션으로 분류가 가능하다고 가정한다.

예를 들어, 의료 정보 시스템의 경우 (표 1)과 같이 응급환자 발생 신고 접수, 구급차 출동 및 환자 이송, 환자 상태 기록, 투약 정보 기록, 기존 진료기록 확인, 환자의 보호자 연락처 등의 기본 정보 확인, 입원 수속 처리, 진료비 정산 등의 기능을 제공하는 애플리케이션이 있을 수 있다.

(표 1)에서 응급환자 발생 신고 접수, 구급차 출동 및 환자 이송, 환자 상태 기록 등은 우선순위가 높은 펌 실

표 1 의료 정보 시스템의 트랜잭션 예

분류	기능	트랜잭션
펌 실시간 트랜잭션	■ 응급환자 발생 신고 접수 ■ 구급차 출동 및 환자 이송 ■ 환자 상태 기록	■ 신고 상황 기록, 상태 접수 기록 ■ 구급차 정보 확인, 환자 이송 정보 확인 ■ 체온, 심박동수, 호흡상태, 뇌파 정보 기록
소프트 실시간 트랜잭션	■ 투약정보 기록 ■ 기존 진료기록 확인 ■ 환자 기본 정보 확인 ■ 입원 수속 처리 ■ 진료비 정산	■ 약품 정보, 투약 시기, 용량, 회수 기록 ■ 이전 진료 기록, 특이 반응 확인 ■ 보호자, 연락처, 주소, 직업 확인 ■ 입원실 현황, 입원 기간, 병실 배정 처리 ■ 진료내역, 투약정보 등을 이용해 계산

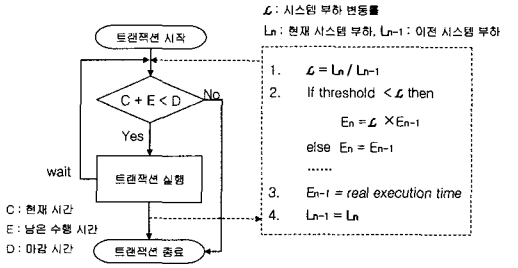
시간 트랜잭션들로 볼 수 있고, 기존 진료 기록 확인, 환자의 기본정보 확인, 입원수속, 진료비 정산 등은 우선순위가 낮은 소프트 실시간 트랜잭션들로 볼 수 있다. 보다 더 자세하게 살펴보면 다음과 같다.

먼저, 환자 상태 기록의 경우 환자의 체온, 심박동수, 뇌파, 호흡상태, X-ray, 초음파 검사 결과 등을 데이타베이스에 주기적으로 기록하는 트랜잭션들이 발생한다. 응급 환자의 상태 변화 정보는 매우 중요한 정보이기 때문에 높은 우선순위를 갖고 처리되어야 하며, 주기적으로 발생되는 트랜잭션이므로 정해진 시간 이내에 처리되지 못하면 곧바로 취소하고 새로운 기록을 남겨야 상태 정보가 의미를 가질 수 있다. 따라서 이러한 상태 정보를 기록하는 트랜잭션들은 펌 실시간 트랜잭션으로 분류할 수 있다. 그리고 진료비 정산의 경우 환자에게 시행된 응급 처치, 각종 검사, 투약 기록 등의 정보를 읽어오고, 주어진 정보에 의해 진료비를 계산하고 처리하는 트랜잭션이 발생한다. 이러한 트랜잭션들은 시급한 처리가 요구되지 않으므로 낮은 우선순위를 갖는 소프트 실시간 또는 비실시간 트랜잭션들로 분류할 수 있다.

응용에서 발생하는 같은 종류의 트랜잭션들은 데이타베이스에 접근하는 패턴과 접근할 데이타 페이지의 수, 연산 시간, 여유시간 등이 비슷한 형태로 나타나게 된다. 따라서 시스템 부하 변동이 없다면 평균 실행시간과 유사하게 실제 실행시간이 나타난다고 가정한다.

시스템 부하가 높아지면 트랜잭션이 실행 중 충돌이 발생할 확률이 높아진다. 반대로 시스템 부하가 낮아지면 트랜잭션이 실행 중 충돌이 발생할 확률은 작아진다. 충돌이 많이 발생하게 되면 트랜잭션은 충돌 해결에 많은 시간을 소비하게 되어 실행 시간이 길어지게 되고, 충돌이 적게 발생하면 충돌 해결에 걸리는 시간이 줄어들기 때문에 실행 시간이 줄어든다. 따라서 시스템 부하가 변동하는 경우 부하 변동에 비례하여 각 트랜잭션의 실행시간도 증가 또는 감소한다고 할 수 있다.

이러한 가정을 바탕으로 (그림 7)과 같은 방법으로




```
boolean feasibility()
{
    if (t + estimated remaining time of T) ≤
            (deadline of T) then
        return true; // feasible
    else
        return false; // infeasible
}
```

그림 7 실행가능성 검사

시스템의 부하 변동 상황 및 각 트랜잭션의 이전 수행시간을 기록하여 해당 트랜잭션의 현재 수행시간을 예측할 수 있다. 이를 위해, 먼저 해당 응용에서 발생하는 트랜잭션들을 타입별로 구분하여 테이블로 관리한다. 이 테이블에는 각 트랜잭션의 평균 수행시간 및 이전 수행 완료 시점의 시스템 부하가 기록된다. 테이블에는 정상적으로 완료된 트랜잭션의 수행시간에 대한 정보만 기록한다.

각 트랜잭션에 대한 예상 수행시간 계산 방법은 크게 4가지로 구분하였고 각각의 계산 방법은 다음과 같다.

(방법 1) 이전 수행시간 $E_{(n-1)}$와 평균 수행시간 $E_{(n-2)}$의 중간 값으로 예상 수행시간 $E_{(n)}$을 결정.

$$E_{(n)} = \frac{E_{(n-1)} + E_{(n-2)}}{2} \qquad \text{(식1)}$$

계산이 가장 단순한 반면에 시스템의 부하 변동을 전혀 고려하지 못하므로 부하 변동이 적은 응용에 적합하다.

(방법 2) 시스템의 부하 변동을 고려하여 일정 주기(시간 또는 트랜잭션의 개수)마다 $E_{(n)}$을 $E_{(n-1)}$로 갱신.

$$E_{(n)} = E_{(n-1)} \quad (식2)$$

매 주기마다 각 트랜잭션의 수행시간이 갱신되므로 해당 응용에 적정한 주기를 찾는 것이 가장 중요하다.

(방법 3) 트랜잭션 완료 시점에서 항상 시스템 부하를 체크하여 테이블에 반영하고, 해당 트랜잭션의 시작 시점에서 시스템 부하를 체크하여 부하 변동 비율을 $E_{(n-1)}$에 곱해준다.

$$E_{(n)} = \frac{l_{(n)}}{l_{(n-1)}} E_{(n-1)} \quad (식3)$$

시스템 부하 변동을 수행시간 계산에 반영하므로 동적인 부하 변동을 갖는 응용에 적합하다. 그러나 매번 시스템 부하를 체크하고 테이블에 반영해야 하는 부담이 있다.

(방법 4) 방법 3과 달리 시스템 부하 변동폭이 경계점(threshold)을 초과하는 경우에만 테이블에 반영함.

$$If\ (Threshold < \frac{l_{(n)}}{l_{(n-1)}})\ then \quad E_{(n)} = \frac{l_{(n)}}{l_{(n-1)}} E_{(n-1)} \quad (식4)$$

시스템의 부하 변동을 반영하면서도 매번 시스템 부하를 계산해야하는 부담을 줄이는 방법이다. 경계점의 폭을 적절하게 결정해 주는 것이 중요하다.

3.3 제안하는 알고리즘의 정확성

(표 2)에서 3개의 트랜잭션 T_1, T_2, T_3은 각각 High priority, Medium priority, Low priority 트랜잭션이다. 또한 데이타에 대한 소유(Hold), 대기(Block) 및 요청(Request) 상태에 따라 각각 H, B, R로 표시한다.

예를 들어 "(T_2,H), (T_1,R)"은 중간 우선 순위를 갖는 트랜잭션 T_2가 현재 데이타를 소유(H)하고 있고, 가장 높은 우선 순위를 갖는 트랜잭션 T_1이 해당 데이타를 요구(R)하는 것을 의미한다. 즉, 표1에서 ⓖ에 해당되는 경우이다. 표1에 제시된 12가지의 상황별 트랜잭션 처리 과정은 다음과 같이 크게 3가지로 분류된다. 첫째, ⓔ, ⓘ,ⓙ와 같이 LPT가 데이타를 소유한 상태에서 HPT가 블록 되는 경우이다. 둘째, ⓖ,ⓚ,ⓛ과 같이 LPT가 데이타를 소유한 상황에서 HPT가 동일 데이타를 요청하는 경우이다. 마지막으로, ⓒ,ⓓ,ⓗ와 같이 HPT가 데이타를 소유한 상황에서 LPT가 동일 데이타를 요청하는 경우이다. (표 2)에 제시한 각각의 상황을 좀 더 자세하게 살펴보면 다음과 같다.

CASE ⓐ : (T_1,H), (T_2,B), (T_3,R)

T_1이 데이타를 소유하고 있고 T_2는 T_1의 완료를 기다리고 있는 상황에서 T_3이 동일한 데이타를 요청하는 경우로 T_2가 대기 상태에 들어갈 때 T_1의 실행가능성 검사를 실시하였으므로 T_1은 "실행 가능" 상태이다. 따라서 T_2가 실행가능성 검사 대상이 되며 그 결과에 따라 T_2가 "실행 가능"이면 계속해서 대기 상태에 있게 되고 "실행 불능"이면 T_2가 제거되고 T_3만 대기 리스트에 추가된다. 이로 인해 T_3의 불필요한 기다림을 방지할 수 있다.

CASE ⓑ : (T_1,H), (T_3,B), (T_2,R)

ⓐ와 비슷한 경우로 단지 대기중인 T_3 보다 더 높은 우선 순위를 갖는 T_2가 동일 데이타를 요청하는 상황이다. 먼저 T_2의 실행가능성 검사가 이루어지고 그 결과가 "실행 가능"인 경우 T_2는 대기 리스트에 추가되고

표 2. 상황별 트랜잭션 처리 과정

	H	B	R	Feasible	Infeasible	비고
ⓐ	T_1	T_2	T_3	T_1완료→T_2완료→T_3완료	T_1완료→(T_2제거→T_3완료)	pr(H)>pr(R)
ⓑ	T_1	T_3	T_2	T_1완료→(T_2완료→T_3완료)	T_1완료→(T_2제거→T_3완료)	pr(H)>pr(R)
ⓒ	T_1	-	T_2	T_1완료→T_2완료	T_1제거→T_2완료	pr(H)>pr(R)
ⓓ	T_1	-	T_3	T_1완료→T_3완료	T_1제거→T_3완료	pr(H)>pr(R)
ⓔ	T_2	T_1	T_3			모순
ⓕ	T_2	T_3	T_1	(T_2재시작→T_1완료)→T_3수행	(T_1제거→T_2완료)→T_3완료	pr(H)<pr(R)
ⓖ	T_2	-	T_1	T_2재시작→T_1완료	T_1제거→T_2완료	pr(H)<pr(R)
ⓗ	T_2	-	T_3	T_2완료→T_3완료	T_2제거→T_3완료	pr(H)>pr(R)
ⓘ	T_3	T_1	T_2			모순
ⓙ	T_3	T_2	T_1			모순
ⓚ	T_3	-	T_1	T_3재시작→T_1완료	T_1제거→T_3완료	pr(H)<pr(R)
ⓛ	T_3	-	T_2	T_3재시작→T_2완료	T_2제거→T_3완료	pr(H)<pr(R)

H: Holder, 자원을 사용중인 트랜잭션
B: Blocking, 대기중인 트랜잭션
R: Requester, 자원을 요청중인 트랜잭션

T_3보다 먼저 서비스를 받게 된다. 그러나 T_2가 "실행 불능"인 경우에는 T_2로 인한 T_3의 불필요한 기다림을 방지하기 위해 T_2는 시스템에서 제거된다.

CASE ⓒ : (T_1,H), (T_2,R)

T_1이 데이타를 소유하고 있는 상황에서 대기하는 트랜잭션이 없는 경우이다. 이때 T_2가 동일 데이타를 요청하면 T_1의 실행가능성 검사가 이루어지고 그 결과가 "실행 가능"인 경우 T_2는 대기 리스트에 추가되고 T_1이 계속해서 수행된다. 만약 T_1이 "실행 불능"인 경우에는 T_1이 시스템에서 제거되고 T_2가 데이타를 소유하게 되어 불필요한 기다림을 방지할 수 있다.

CASE ⓓ : (T_1,H), (T_3,R)

ⓒ와 같은 상황으로 동작 과정 또한 동일하다.

CASE ⓔ : (T_2,H), (T_1,B), (T_3,R)

T_2에 의해 T_1이 대기중인 상황이므로 HPT는 LPT보다 항상 선행한다는 것을 위배하는 경우이다. 따라서 이와 같은 상황은 시스템에서 절대 발생되지 않는다.

CASE ⓕ : (T_2,H), (T_3,B), (T_1,R)

T_2가 데이타를 소유하고 있고 T_3이 대기중인 경우에 가장 우선 순위가 높은 T_1이 동일 데이타를 요청하는 상황이다. 먼저 T_1의 실행가능성 검사가 이루어지고 그 결과가 "실행 가능"인 경우 T_2는 재시작 되고 T_1이 데이타를 소유하게 된다. 이때 T_3은 계속해서 대기 리스트에 남아 있으면서 T_1의 완료를 기다린다. T_1이 완료된 후 T_3이 데이타를 점유하게 되지만 T_2가 재시작 되어 동일 데이타를 재 요청하는 경우에 T_3은 T_2의 실행가능성 검사 결과에 따라 재시작 되거나 또는 계속 수행된다. 반대로 T_1이 "실행 불능"인 경우에는 T_1이 시스템에서 제거되고 T_2가 계속해서 수행된다. 따라서 T_2의 불필요한 재시작을 방지할 수 있다.

CASE ⓖ : (T_2,H), (T_1,R)

ⓕ와 비슷한 경우로 단지 대기중인 트랜잭션이 없는 경우이다. T_1의 실행가능성 검사 결과에 따라 "실행 가능"이면 T_2가 재시작 된다. 반대로 T_1이 "실행 불능"이면 ⓕ의 경우에서와 같이 T_1이 시스템에서 제거되고 T_2가 계속해서 수행된다.

CASE ⓗ : (T_2,H), (T_3,R)

T_3이 동일 데이타를 요청하는 경우로 먼저 T_2의 실행가능성 검사가 이루어진다. 그 결과가 "실행 가능" 이면 T_3은 대기 리스트에 추가된다. 만약 T_2가 "실행 불능" 이면 T_2는 시스템에서 제거되고 T_3이 해당 데이타를 소유하게 되어 불필요한 기다림을 방지할 수 있다.

CASE ⓘ : (T_3,H), (T_1,B), (T_2,R)

ⓔ와 같이 모순되는 경우로 시스템에서 절대 발생되지 않는 상황이다.

CASE ⓙ : (T_3,H), (T_2,B), (T_1,R)

ⓔ,ⓘ와 같이 모순되는 경우로 시스템에서 절대 발생되지 않는 상황이다.

CASE ⓚ : (T_3,H), (T_1,R)

ⓖ와 같이 높은 우선 순위를 갖는 트랜잭션이 동일 데이타를 요청하는 상황으로 T_1의 실행가능성 검사 결과에 따라 "실행 가능"이면 T_3이 재시작 되고 "실행 불능"이면 T_1이 시스템에서 제거되어 T_3의 불필요한 재시작을 방지할 수 있다.

CASE ⓘ : (T_3,H), (T_2,R)

ⓖ,ⓚ와 같은 상황으로 T_3의 불필요한 재시작을 방지하기 위해 T_2의 실행가능성 검사를 실시하고 그 결과에 따라 T_3이 재시작 되거나 또는 T_2가 시스템에서 제거된다.

지금까지 살펴본 바와 같이 제안하는 알고리즘은 우선 순위 역전 현상을 발생시키지 않는다. 또한 데이타베이스의 직렬성을 항상 보장한다. 그러나 2PL 방법을 기반으로 하기 때문에 교착상태 발생은 불가피하다. 예를 들어 (T_2,H)가 데이타 x를 소유한 상황에서 또 다른 동일 우선 순위를 갖는 트랜잭션 (T_2,R)이 데이타 x를 요청하는 경우에는 이미 선행중인 (T_2,H)가 계속 수행된다. 그러나 (T_2,H)의 수행 도중에 (T_2,R)이 소유한 데이타 y를 필요로 한다면 교착상태가 발생한다. 따라서 교착상태에 대한 탐지 알고리즘은 일반적인 방법을 사용하고 있으며, 교착상태에 빠진 트랜잭션들 중에서 시간 제약조건을 고려하여 실행가능성이 더 낮은 트랜잭션을 희생 대상으로 선정한다.

4. 성능 평가

4.1 시뮬레이션 모델

실험은 (그림 8)과 같은 시뮬레이션 모델에서 실시하였다. 시뮬레이션 모델은 실행가능성 검사시기에 따라 3가지로 구분하였다.

(모델 1) 트랜잭션을 시작할 때에만 실행가능성 검사를 수행.

트랜잭션 시작부분에서 수행 가능한 트랜잭션들만을 시스템에 진입시키므로 불필요한 자원 낭비를 줄일 수 있다. 또한 충돌 발생 시마다 실행가능성 검사를 하지 않아도 되고, 구현이 용이하다는 장점을 갖는다. 그러나 충돌이 발생되지 않는 트랜잭션에 대해서도 무조건 검사를 해야 하며, 트랜잭션 수행 도중에 발생되는 충돌에 의한 지연시간을 전혀 고려하지 못하므로 검사 결과에 대한 신뢰성이 시간이 갈수록 떨어진다.

(모델 2) 충돌이 발생된 경우에만 실행가능성 검사를 수행.

충돌 발생 시마다 매번 실행가능성 검사를 수행하므로 결과에 대한 신뢰성이 매우 높다. 그러나 시작 또는 재시작되는 실행 불가능한 트랜잭션들을 필터링하지 못하므로 불필요한 자원을 낭비할 수 있다.

(모델 3) 실행가능성 검사를 트랜잭션 시작 및 충돌 시 수행.

모델 1, 2의 장점을 취한 것으로 시작 또는 재시작되는 트랜잭션들 중에서 실행 불가능한 것들을 미리 시스템에서 제거할 수 있다. 또한 충돌 및 대기에 따른 지연시간을 고려하여 충돌 발생 시 실행가능성 검사를 다시 수행하여 검사 결과의 신뢰성을 높인다.

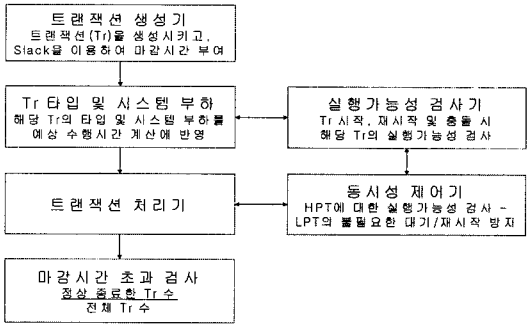


그림 8 시뮬레이션 모델

성능평가는 대체버전을 이용한 동시성 제어 방법[23,26]을 비교 대상으로 하였다. 또한 앞서 제안한 4가지 실행 가능성 검사 방법에 따른 성능평가와 3가지 시뮬레이션 모델간의 성능평가를 실시하였고, 트랜잭션의 마감시간 초과비율을 성능평가 기준으로 삼았다. 트랜잭션의 마감시간 초과 비율에 대한 계산식은 (식 5)와 같다.

$$\text{마감시간초과비율} = \frac{\text{실패한트랜잭션수}}{\text{전체트랜잭션수}} \times 100(\%) \quad \text{(식5)}$$

성능평가 환경은 AMD 1Ghz CPU, 메모리 392MB의 Windows 2000 서버 기반의 PC에서 실시하였고, Visual C++ 6.0을 사용하여 구현하였다.

성능 평가에 사용된 시스템 파라미터는 (표 3)와 같이 대체버전을 이용한 동시성 제어방법의 성능평가 환경과 동일하게 하였다. 시뮬레이션에 사용한 트랜잭션들은 (그림 6)의 혼합 트랜잭션을 위한 알고리즘을 평가하기 위하여 펌 실시간 트랜잭션들과 소프트 실시간 트랜잭션들로 구성하였다. 성능평가를 위해 100가지 타입의 트랜잭션을 정의하고, 총 20,000개의 트랜잭션을 성능평가에 사용하였다. 시스템 부하에 따른 성능 평가를 위해 초당 트랜잭션 도착율은 10~1,000개로 변화시켜 가면서 수행하였다.

표 3 시스템 파라미터

파라미터	의미	설정값
DB_Size	데이타베이스 크기	1000 pages
CPU_Time	데이타page 접근시간	10 ms
Bcopy_Time	메모리블록 접근시간	0.5 ms
NumCPUs	CPU 수	1 cpu
TransSize	트랜잭션당 접근 page 수	16×(0.5~1.5) pages
Slack	트랜잭션의 여유시간	100%~650%
Deadline	트랜잭션의 마감시간	*Est_Time× Slack

*Est_Time : 해당 트랜잭션의 예상수행시간.

수행 도중 처리하는 데이타 페이지 수는 8~24로 균등 분포를 갖는다. 데이타를 읽고 쓰는데 걸리는 입출력 시간은 모든 데이타 페이지가 메모리에 상주하는 것으로 가정하여 메모리 블록 접근시간[24,25]으로 하였다. Slack은 수행시간의 100%~650%로 하였다. 우선순위 할당 방법은 EDF(Earliest Deadline First) 방법을 적용하였다. 시뮬레이션 절차는 다음과 같다.

새로운 트랜잭션은 해당 타입별로 Slack을 고려하여 마감시간을 부여받는다. 트랜잭션들은 시스템에 도착하면 우선순위 큐에서 대기하게 된다. 우선 순위 큐에 있는 트랜잭션들은 우선 순위에 따라 순서대로 실행되도록 큐에서 정렬된다. 새롭게 수행되는 트랜잭션 및 재시작되는 트랜잭션에 대해서는 실행가능성 검사가 수행된다. 그리고 타입별로 평균수행시간 및 시스템 부하를 고려하여 예상 수행시간이 산출된다. 이를 통해 실행 불가능한 트랜잭션들을 미리 시스템에서 제거시킨다. 트랜잭션 수행 도중에 충돌이 발생되면 동시성 제어기는 두 트랜잭션 중에서 높은 우선순위를 갖는 HPT에 대한 실행가능성 검사를 실시한다.

선행하던 HPT의 실행가능성 검사 결과 HPT가 "실행불능"인 경우, HPT를 시스템에서 제거시키고 요청하는 LPT가 자원을 사용하도록 하여 불필요한 대기시간을 제거한다. 이와 반대로, 선행하던 LPT가 사용중인 자원을 HPT가 요청하는 경우, HPT의 실행가능성 검사를 실시한다. 그 결과가 "실행불능"인 경우 HPT는 시스템에서 제거되고, LPT는 계속해서 수행하게 하여 불필요한 재시작 시간을 제거한다. 정상적으로 수행이 완료된 트랜잭션에 대한 정보는 테이블에 기록하여 관리한다.

4.2 실험 결과 및 분석

본 논문에서 제안하는 동시성 제어 알고리즘은 마감시간 초과 비율에 대한 시뮬레이션 결과 대체버전을 이용한 동시성 제어 방법보다 향상된 성능평가 결과를 보였다.

대체버전을 이용한 동시성 제어 방법은 항상 LPT의 정지 버전 및 재시작 버전을 동시에 유지하다가 HPT의 정상적인 완료 여부에 따라 적절한 버전을 수행시킨다. 따라서 마감시간 초과 비율이 기존의 다른 방법보다 우수하다[26]. 그러나 대체버전을 이용한 동시성 제어 방법은 각 트랜잭션들마다 몇 개의 리스트를 메모리 상에 유지하면서 하나의 즉시 재시작 버전과 여러 개의 정지 버전을 관리해야 하므로 시스템 자원의 추가 사용 및 버전 정보의 유지/관리에 따른 비용이 추가로 필요하다. 또한 충돌 발생 회수가 증가할 경우 불필요하게 수행되는 재시작 버전들이 많아지고 이에 따라 심각한 자원 낭비가 초래되므로 일반 트랜잭션들의 정상 수행을 방해하여 전체적인 시스템 성능을 떨어뜨린다.

제안하는 동시성 제어 알고리즘에서는 실행 가능성 검사를 통해 대체버전을 이용한 동시성 제어 방법에서 불필요하게 수행되는 대체버전을 제거하여 자원 낭비와 트랜잭션 유지/관리에 드는 추가적인 비용을 제거하였다. 또한 시스템의 부하를 감소시켜 더 많은 트랜잭션들이 정상적으로 수행되며, 전체적인 시스템 성능 향상을 얻을 수 있다.

다음 (그림 9)는 초당 트랜잭션 도착율에 따른 마감시간 초과 비율을 대체버전을 이용한 동시성 제어방법과 비교한 결과이다. 본 실험에서는 앞서 설명한 시뮬레이션 모델 중 충돌 검사를 트랜잭션 시작 시와 충돌 발생 시 모두 수행하는 모델 3을 사용하였고 실행 가능성 검사 방법은 방법 4를 사용하였다.

제안하는 동시성 제어 알고리즘과 대체버전을 이용한 동시성 제어 방법의 마감시간 초과 비율을 비교해본 결과 (그림 9)와 같이 초당 트랜잭션 개수를 점차 증가시킴에 따라 제안하는 동시성제어 알고리즘이 마감시간 초과 비율 관점에서 전반적으로 평균 약 15.6% 정도의 성능 향상을 보임을 확인할 수 있었다.

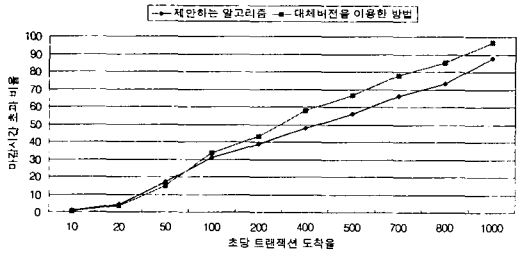


그림 9 초당 트랜잭션 도착율에 따른 마감시간 초과 비율 비교

트랜잭션 도착율이 낮은 경우 대체버전을 이용한 동시성제어 방법이 약간 나은 성능을 보인다. 대체버전을 이용한 동시성제어 방법에서는 충돌이 발생할 경우 항상 대기버전과 재 시작 버전을 생성하여 높은 우선순위 트랜잭션의 실행 결과에 따라 최선의 버전을 선택하여 처리하고 있다. 이와 달리 제안하는 방법에서는 실행가능성 검사를 통해 불필요한 하나의 대체버전을 제거하여 처리율을 높이고 있지만, 예상실행시간 계산의 오차에 의해 최선의 버전이 선택되지 않는 경우가 발생할 수 있다. 따라서 트랜잭션 도착율이 낮은 경우 대체버전을 이용한 방법에 비해 제안하는 방법의 처리율이 약간 낮게 나타나게 된 것이다. 그러나 대체버전을 이용한 동시성제어 방법에서는 트랜잭션 도착율이 증가하는 경우 충돌 발생 횟수가 증가되고 이에 따라 불필요하게 수행되는 재 시작 버전들이 많아지게 된다. 이에 따라 심각한 자원 낭비를 초래하게 되고, 결국 일반 트랜잭션들의 정상 수행을 방해하여 전체적인 시스템 성능이 떨어지게 된다.

반면 트랜잭션 도착율이 높은 경우 제안하는 동시성 제어 알고리즘이 더 나은 성능을 보이는 주된 이유는 대체버전을 이용한 동시성제어 방법에서 불필요하게 수행되던 LPT의 재시작 버전을 실행 가능성 검사를 통하여 효과적으로 제거하여 시스템의 부하를 줄이고 다른 트랜잭션들의 정상적인 처리에 필요한 시간을 더 많이 확보할 수 있었기 때문이다.

다음 (그림 10)은 앞 실험에서 각각의 방법을 적용했을 때 재시작 된 트랜잭션의 수를 비교한 결과로, 충돌에 의해 재시작 된 트랜잭션이 다른 트랜잭션과 또다시 충돌해 발생하는 재시작 트랜잭션의 수도 함께 포함한 결과이다. 트랜잭션 재시작 비율에 대한 계산식은 (식 6)과 같다. 대체버전을 이용한 동시성제어 방법의 경우 트랜잭션의 충돌이 발생할 때마다 LPT의 재시작 버전을 새롭게 생성하므로 충돌이 많이 발생하는 경우 제안하는 알고리즘에 비해 재시작 된 트랜잭션의 수가 많음을 볼 수 있다. 제안하는 동시성제어 알고리즘에서는 불필요한 버전 생성을 실행 가능성 검사를 통해 효과적으로 억제하고 있으며, 불필요한 대기 및 재시작을 방지함으로써 시스템 자원의 낭비를 막고, 처리율 향상 및 마감시간 초과비율이 최소화가 되도록 하고 있다. 결과적으로 제안하는 알고리즘이 대체버전을 이용한 방법에

트랜잭션 재시작 비율면에서 평균 약 34% 성능향상을 보였다.

$$\text{트랜잭션재시작비율} = \frac{\text{재시작트랜잭션수}}{\text{신규트랜잭션수}} \times 100(\%) \quad (식6)$$

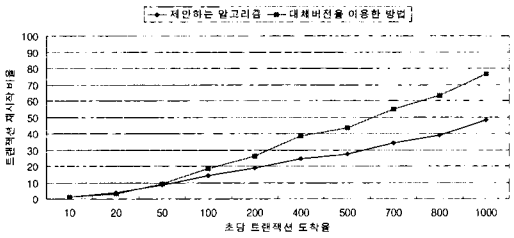


그림 10 초당 트랜잭션 도착율에 따른 트랜잭션 재시작 비율 비교

다음 (그림 11)은 실행 가능성 검사에 사용하는 각각의 방법에 따른 마감 시간 초과 비율을 비교한 결과이다. 시뮬레이션 모델은 앞서와 마찬가지로 모델 3을 이용하였다. 방법 2의 경우에는 한 트랜잭션이 5번 실행될 때마다 예상 수행시간을 갱신하게 하였다. 방법 4의 경우에는 경계점을 이전 시스템 부하에 비해 ±50%이상 변경될 경우로 정하였다.

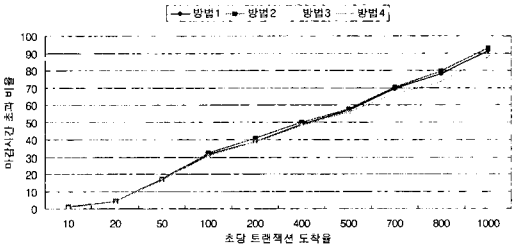


그림 11 실행 가능성 검사 방법에 따른 마감시간 초과 비율 비교

실행 가능성 검사 방법들 간의 성능평가 결과에서 각 방법이 큰 차이를 보이지 않고 있으나, 초당 트랜잭션 도착율이 높아질수록 방법 3, 4가 방법 1, 2에 비해 약간 더 나은 성능을 보인다. 방법 1은 처음 시작부터 트랜잭션들이 매번 실행될 때마다 실행시간을 누적하여 산술 평균을 구하는 방법으로 가장 최근의 실행시간이 예상수행시간 계산에 가장 큰 영향을 미치게 된다. 예를 들어 (그림 12)에 제시한 계산법 A)는 총 5번의 실행시간(1.0, 2.0, 3.0, 4.0, 5.0)에 대한 평균으로 3.0을 산출한다. 여기에서 1.0은 최초(과거) 실행시간이고, 5.0은 가장 최근의 실행시간이다. 계산법 B)는 과거 기록보다 최근 실행시간에 더 큰 비중을 두고 계산하는 방법으로 4.0625를 산출한다. 과거 기록보다는 최근 상황을 바탕으로 미래를 예측하는 것이 더 좋은 결과를 얻을 수 있기 때문에 제안하는 방법 1은 B)와 같은 계산 방법을 택하고 있다. 이로 인해 최근의 시스템 부하 변동 상황을 어느 정도 반영시킬 수 있다.

계산법 A)

$$\frac{1+2+3+4+5}{5} = 3.0$$

계산법 B)

$$\frac{1+2}{2} = 1.5,\ \frac{1.5+3}{2} = 2.25,\ \frac{2.25+4}{2} = 3.125,$$

$$\frac{3.125+5}{2} = 4.0625$$

그림 12 산술 평균과 누적 평균 계산 결과 비교

방법 2는 트랜잭션들이 매번 수행할 때마다 실행시간을 누적하여 평균을 구하지 않고, 일정 시간 또는 일정 회수를 주기로 이전 실행 시간을 예상 실행시간에 바로 반영하는 방법이다. 따라서 방법 2와 같은 경우 적절한 주기를 선택하는 것이 중요하다. 본 논문의 시뮬레이션에서는 일정 회수를 기준으로 주기적으로 반영하도록 하였으며, (그림 11)은 주기를 여러 값으로 변경하여 시뮬레이션 해 본 결과 중 주기를 매 5회로 하였을 때의 결과이다. 주기 선택이 적절하지 못한 경우에는 (그림 11)에 제시한 결과보다 훨씬 더 성능이 떨어지는 결과를 얻을 수 있었다. 따라서 실제 응용에 이용할 경우, 시시각각 변화하는 시스템 상황을 적절하게 반영하여 주기를 변화시킬 수 있는 방법이 추가적으로 고려되어야 하는 단점을 갖는다.

방법 3의 경우 매번 시스템의 부하 변화량을 예상수행시간 계산에 반영하는 것으로 보다 정확한 계산이 가능한 방법이다. 시스템 부하변동이 심한 경우일수록 위의 방법 1, 2에 비해 훨씬 더 좋은 결과를 보인다. (그림 11)은 트랜잭션 도착율이 지수함수 분포로 증가하는 경우로, 방법 1, 2와 그다지 큰 차이를 보이지 않고 있다. 이것은 방법 1도 최근 상황을 어느 정도 반영하고 있고, 방법 2의 주기 선택이 적절했기 때문이다.

방법 4는 방법 3에서 이전 실행 시간을 시스템 부하가 일정한 경계값을 초과하는 경우에만 예상 실행 시간에 반영하도록 하는 방법이다. 이 방법은 경계값을 크게 하거나 또는 작게하는 경우 결과가 다르게 나타나며, 0으로 하였을 경우는 방법 3과 동일한 결과를 얻을 수

있다. (그림 11)에서 볼 수 있듯이 부하변동이 없는 경우 또는 아주 경미한 경우에도 매번 시스템 부하 변동률을 반영하는 방법 3과 이와 달리 부하 변동률이 일정 폭 이상일 경우에만 반영하는 방법 4의 차이가 거의 나타나지 않는다. 즉, 방법 3에서와 같이 부하변동이 없는 경우 또는 아주 경미한 경우에 부하 변동률을 반영하는 것은 무의미함을 알 수 있다. 게다가 트랜잭션 도착률이 아주 큰 경우에는 부하 변동률 계산 횟수가 매우 많아지면서 오히려 시스템 부하를 증가시키기 때문에 방법 4에 비해 성능이 더 떨어지는 결과를 얻을 수 있다.

다음 (그림 13)는 시뮬레이션 모델에 따른 마감시간 초과 비율을 비교한 결과이다. 실행 가능성 검사 방법은 방법 4를 사용하였다.

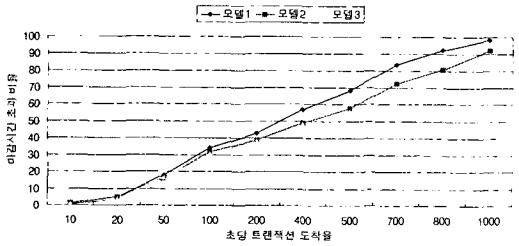


그림 13 시뮬레이션 모델별 마감시간 초과 비율 비교

(그림 13)의 결과에서 시뮬레이션 모델 1은 초당 트랜잭션 도착율이 높아질수록 성능이 현저히 떨어지는 것을 볼 수 있다. 그 이유는 모델 1의 경우 트랜잭션의 시작 시에만 실행 가능성 검사를 실시하기 때문에 트랜잭션의 수행 도중 충돌이 많이 발생하는 경우 불필요한 재시작 및 기다림이 많이 발생하게 되어 마감시간 초과 비율이 급격하게 늘어난다. 모델 2는 충돌 시에만 트랜잭션에 대한 실행 가능성 검사를 실시하지만, 트랜잭션 도착율이 늘어나 충돌이 많이 발생하더라도 충돌이 발생한 트랜잭션에 대한 적절한 처리를 해준다. 따라서 불필요한 재시작 및 기다림을 줄일 수 있어 모델 3에 비해 결과의 차이가 크게 나지 않는다. 그리고 예상대로 모델 1,2의 장점을 취합한 모델 3이 가장 우수한 성능을 보임을 알 수 있다.

5. 결 론

제안하는 동시성제어 알고리즘은 방법은 실행 가능성 검사를 통해 LPT의 불필요한 재시작 및 기다림을 방지할 수 있도록 하고 있다. 그 결과 상대적으로 LPT인 비실시간 트랜잭션 및 소프트 실시간 트랜잭션의 처리 효율을 증대시키고 전체적인 마감시간 초과 비율을 최소화한다.

제안하는 동시성제어 알고리즘에서 사용되는 트랜잭션의 실행 가능성 검사를 위해 트랜잭션의 예상 수행 시간 계산 방법을 제시하였다. 시스템의 부하 변동이 적은 경우와 많은 경우에 적절하게 사용될 수 있는 방법들을 다양하게 제시함으로써 실행가능성 검사 결과의 신뢰성을 높이고 성능을 향상시킬 수 있도록 하였다.

제안된 알고리즘의 우수성을 입증하기 위해 대체 버전을 이용한 동시성제어 방법을 비교 대상으로 하여 성능평가를 실시하였다. 그리고 제안한 실행 가능성 검사 방법들간의 성능평가와 시뮬레이션 모델들에 대한 성능평가를 실시하였다. 성능평가를 통해 초당 트랜잭션 도착율이 높아질수록 제안하는 동시성제어 알고리즘이 우수한 성능을 보임을 알 수 있었다.

향후에는 좀 더 다양한 동적 시스템 부하 변동을 고려한 환경에서 성능평가를 실시하고, 비실시간 및 소프트 실시간 트랜잭션의 처리율 향상 정도를 측정해 보고자 한다.

참 고 문 헌

[1] Ben Kao and Hector Garcia-Molina, "An Overview of Real-Time Database Systems," In Sang H. Son, editor, Advances in Real-Time Systems, chapter 19. Prentice Hall, 1995

[2] Krithi Ramamritham, "Real-Time Databases," International Journal of Distributed and Parallel Databases, 1(2), 1993

[3] G. Ozsoyoglu and R. Snodgrass, "Temporal and Real-Time Databases: A Survey," IEEE Transactions on Knowledge and Data Engineering, 7(4):pp.513-532, August 1995

[4] N. Redding "Network Services Databases," in Proc. of IEEE Global Telecommunications Conference, pp.1336-1340, 1986

[5] R. M. Sivasankaran, B. Purimetla, J. A. Stankovic and K. Ramamritham, "Network Services Databases - A distributed active real-time database applications," in Proc. of IEEE workshop on Real-Time Applications, pp.184-187, 1993

[6] S. Hvasshovd, O. Torbjornsem, "The ClustRa Telecon Databases High Availability, High Throughput, and real-time response," in Proc. of 21th VLDB, pp.469-477, 1995

[7] John Voelcker, "How computers helped stampede the stock market," IEEE Spectrum, Vol.24, pp.30-33, 1987

[8] Jayant R. Haritsa, Michael J. Carey, and Miron Livny, "Dynamic real-time optimistic concurrency control," in Proc. of Real-Time Systems Symposium. pp.94-103, 1990
[9] Jayant R. Haritsa, Michael J. Carey and Miron Livny, "Earliest Deadline Scheduling for real-time database systems," in Proc. of Real-Time Systems Symposium, pp.232-242, 1991
[10] D. Hong, M. Kim, and S. Chakravarthy, "Incorporating load factor into the scheduling of soft real-time transactions for main memory database," in Proc. of RTCSA'96, pp.60-66, 1996
[11] Liu C L, Layland J W, "Scheduling Algorithms for Multi-programming in Hard real-time Environment," 1976
[12] Stankovic J, Spuri M, Natale M D, "Implications of Classical Scheduling Results for Real-Time Systems," 1995
[13] Sha L, Rajkumar R, Son S H, Chang C H, "A Real-Time Locking Protocol," 1991
[14] Abbott R, Garcia-Molina H, "Scheduling Real-time Transactions: A Performance Evaluation," ACM Transactions on Database Systems, 17(3):pp.513-560, 1992
[15] Jayant R. Haritsa, Michael J. Carey and Miron Livny, "Data Access Scheduling in Firm Real-time Database Systems," 1992
[16] Lee J, "Concurrency Control Algorithms for Real-time Database Systems," 1994
[17] Jayant R. Haritsa, Michael J. Carey and Miron Livny, "On Being Optimistic About Real-Time Constraints," In Proceedings of the ACM Symposium on Principles of Database Systems, April 1990
[18] J. Huang, A. Stankovic, "Experimental Evaluation of Real-Time Concurrency Control Schemes," Proceedings of the 17th VLDB Conference, pp. 35-46, 1991
[19] Rakesh Agrawal, Michael J. Carey and Miron Livny, "Concurrency control performance modeling: alternatives and implications," ACM TODS, Vol.12, No.4, pp.609-654, 1987
[20] Jiandong Huang, John A. Stankovic, Krithi Ramamritham and Don Towsley, "Experimental evaluation of real-time optimistic concurrency control schemes," in Proc. of 17th VLDB, pp.35-46, 1991
[21] J. Lee and Sang H. Son, "Using dynamic adjustment of serialization order for real-time database systems," in Proc. of Real-Time Systems Symposium. pp.66-75, 1993
[22] Robbert Abbott and Hector Garcia Molina, "Scheduling real-time transactions with disk resident data," in Proc. of 15th VLDB, pp.385-396, 1989
[23] D. Hong, Sharma Chakravarthy, Theodore Johnson, "Locking Based Concurrency Control for Integrated Real-Time Database Systems," RTDB'96, pp.138-143, 1996
[24] Anthony Chiu, Ben Kao, Kam-yiu Lam, "Comparing two-phase locking and optimistic concurrency control protocols in multiprocessor real-time databases," Parallel and Distributed Real-Time Systems, pp141-148, 1997
[25] W. Richard Stevens, "Advanced Programming in the UNIX Environment," pp427-550, Addison-Wesley, 1992
[26] 홍동권, "대체 버전을 이용한 폼 실시간 데이타베이스 동시성 제어 방법", 한국정보처리학회논문지 제5권 제6호, pp.1377-1389, 1998
[27] Albert Mo Kim Cheng, "Scheduling Transactions in Real-Time Database Systems," Compcon Spring '93, Digest of Papers, pp222-231, 1993
[28] Patrick E. O'Neil, Krithi Ramamritham, Calton Pu, "A Two-Phase Approach to Predictably Scheduling Real-Time Transactions," 1994
[29] Yoshifumi Manabe, Shigemi Aoyagi, "A Feasibility Decision Algorithm for Rate Monotonic and Deadline Monotonic Scheduling," 1st Real-Time Technology and Applications Symposium, pp212-218, 1995

이 석 재
2000년 충북대학교 정보통신공학과(공학사). 2002년 충북대학교 정보통신공학과(공학석사). 현재 충북대학교 정보통신공학과(박사과정 재학). 관심분야는 데이타베이스 시스템, 메모리 상주형 데이타베이스 시스템, 저장 시스템, 실시간 시스템 등

신 재 룡
1996년 충북대학교 정보통신공학과(공학사). 1998년 충북대학교 정보통신공학과(공학석사). 2001년 충북대학교 정보통신공학과(박사과정 수료). 관심분야는 데이타베이스 시스템, 실시간 시스템, 멀티미디어 데이타베이스 등

송 석 일
정보과학회 논문지 : 데이타베이스
제 29 권 제 1 호 참조

유 재 수
1989년 전북대학교 공과대학 컴퓨터공학과(공학사). 1991년 한국과학기술원 전산학과(공학석사). 1995년 한국과학기술원 전산학과(공학박사). 1995년 ~ 1996년 목포대학교 전산통계학과 전임강사. 1996년 ~ 현재 충북대학교 전기전자및컴퓨터공학부 부교수 및 컴퓨터정보통신연구소. 관심분야는 데이타베이스 시스템, 정보검색, 멀티미디어 데이타베이스, 분산 객체 컴퓨팅 등

조 기 형
1966년 인하대학교 전기공학과(공학사). 1984년 청주대학교 산업공학과(공학석사). 1992년 경희대학교 전자공학과(공학박사). 1988년 ~ 현재 충북대학교 전기전자및컴퓨터공학부 교수 및 컴퓨터정보통신연구소. 관심분야는 데이타베이스, 소프트웨어 시스템 설계 및 구현, 컴퓨터 네트워크 설계 등

이 병 엽
1991년 한국과학기술원 전산학과(공학사). 1993년 한국과학기술원 전산학과(공학석사). 1997년 한국과학기술원 경영정보공학(공학박사). 1997년 ~ 현재 대우정보시스템 (주) CRM사업팀 차장. 관심분야는 데이타마이닝, XML, 인공지능, 멀티미디어 데이타베이스, 분산 객체 컴퓨팅 분야 등