데이타 우선순위에 기초한 트랜잭션 스케줄링 알고리즘의 제안 및 실시간 DBMS에서의 성능 비교연구

윤 석 환† · 이 재 영† · 박 치 항†

요 약

본 논문에서는 기존의 알고리즘들이 트랜잭션에 부여하였던 우선순위를 특정 데이타 항목에 부여하여 접근하는 트랜잭션중 가장 높은 우선순위의 트랜잭션을 먼저 수행토록 함으로써 교착상태(deadlock)를 쉽게 예방할 수 있는 점에 착안하여 데이타 항목에 우선순위를 부여하는 데이타 우선순위에 기초한 잠금 프로토콜(DPLP: Data-Priority Based Locking Protocol)을 제안하고, 이 알고리즘의 성능을 실시간 데이타베이스 시스템에서 기존 잠금 프로토콜의 성능과 비교분석하였다. 비교 기준으로는 트랜잭션 도착 시간차(IAT: inter arrival time)에 따른 트랜잭션의 성공 비율(success ratio), 평균 지연 (average-latencss), 충돌 비율(conflict-ratio), 재시작 비율(restart-ratio)을 채택하였다. 성능 비교의 결과는 본 논문에서 제안하는 데이타 우선순위에 기초한 잠금 프로토콜이 기존의 잠금 프로토콜들보다 우수한 성능을 보임을 확인하였다.

A Proposal on Data Priority Based Transaction Scheduling Algorithm and the Comparative Performance Evaluation in Real Time Database Management System

Seok-Hwan Yoon† · Jaeyoung A Lee† · Chee-Hang Park†

ABSTRACT

It is possible to prevent deadlock if the priority which is conventionally given to transactions is endowed to data items and the transaciton with the highest priority among transactions accessing the data item is allowed to proceed. Based on this observation, we proposed a Data-Priority Based Locking Protocol(DPLP) and evaluated its performance against known locking protocols in Real Time Database Management System(RTDBMS). Transaction inter-arrival time(IAT) was varied to determine success ratio, average-lateness, conflict ratio and restart ratio. In these simulations we verified that the proposed DPLP performs better than the other protocols.

1. 서 론

1.1 개 요

'90년대에 들어 종래의 메인프레임 위주의 컴퓨팅 환경이 분산 컴퓨팅 환경으로 바뀌어 가면서 온-라인 트랜잭션 처리(OLTP: On-Line Transaction Processing) 방식에 대한 관심이 고조되고 있다. 이러한 방식은,

† 정 회 원: 한국전자통신연구소 컴퓨터연구단 멀티미디어 연구부

논문접수: 1996년 3월 15일, 심사완료: 1996년 5월 31일

이기종간의 인터페이스까지 포함하는 분산처리 능력이 강화되어 원격지간의 컴퓨터 통신 및 공동작업 등을 가능케 하고 있다. 이는 클라이언트인 단말의 기능이 강화되고 네트워크가 빠르고 안정적으로 발전하여 클라이언트와 서버간의 통신 및 클라이언트간의 통신이 보다 용이해졌기 때문이다.

분산 환경에서의 컴퓨터 통신은 기본 단위인 트랜잭션을 통해서 이루어지며, 특정 클라이언트가 요구하는 내용을 서버 및 다른 클라이언트가 처리하여 다시 그 클라이언트에게 보내는 구조로 되어 있다. 트랜잭션은 특정 클라이언트에서 서버 및 다른 클라이언트에 접근하는 화일시스템(file systems) 명령어와 데이타베이스 명령어 등을 처리하는 일련의 기본 조작들의 순서적 나열이며, 데이타베이스의 현재의 일치 상태(consistent state)를 새로운 일치 상태로 변환하는 연산 행위이다. 트랜잭션이 성공적으로 끝나면, 모든 수정 사항은 반영(committed)되거나 취소된다(aborted).

서버는 동시에 다수의 클라이언트로부터의 트랜잭션을 받아서 처리할 수 있도록 설계되어 구현되어야 한다. 이러한 다중 트랜잭션들은 서버에 의해 조절되는 일정한 순서로 처리되어져야 한다. 이를 트랜잭션 스케줄링이라 하며, 이를 위해서는 서버에 트랜잭션 스케줄러가 반드시 존재하여야 한다. 다중 트랜잭션들은 수행상 상호 충돌(conflict)을 일으키거나, 교착상태(deadlock)를 발생시킬 수 있다. 두 개의 서로 다른 트랜잭션이 동일한 데이타 항목에 대해 쓰기 작업을 하려할 때, 이 두 개의 트랜잭션은 충돌되었다고 하며, 서로 다른 트랜잭션 T와 U가 서로 다른 데이타 항목에 대한 잠금을 가지고 있어 서로의 처리를 기다릴 때 이를 교착상태라고 한다. 트랜잭션 스케줄러는 트랜잭션 스케줄링 알고리즘(TSA:Transaction Scheduling Algorithm)에 의거하여 트랜잭션들을 순차화(serialize)하여 충돌을 해결하고 교착상태를 방지하는 역할을 수행한다.

트랜잭션을 순차화하기 위하여 스케줄러가 사용하는 알고리즘은 잠금 기법(locking)[1, 2], 시간소인 순서화 기법(timestamp-ordering)[3], 낙관적 기법(optimistic method)[4] 등의 기본적 동시성 제어 전략을 적용하여 구현된다.

본 논문에서는, 기존의 알고리즘들이 트랜잭션에 부여하였던 우선순위를 특정 데이타 항목에 부여하여 접근하는 트랜잭션중 가장 높은 우선순위의 트랜잭션을 먼저 수행토록 함으로써 교착상태를 예방할 수 있는 점에 착안하여, 데이타 우선순위에 기초한 잠금 프로토콜(DPLP:Data-Priority Based Locking Protocol)을 제안하고, 실시간 데이타베이스 시스템에서의 시뮬레이션을 통하여 이 알고리즘의 성능을 기존 잠금 프로토콜의 성능과 비교 분석하였다.

1.2 지금까지의 관련 연구 내용

실시간 데이타베이스 시스템(RTDBMS:Real-Time Database Management System)에서의 트랜잭션 스케줄링 문제에 대해서는 그동안 많은 연구가 행해졌다. R.Abbott와 H.Garcia-Molina는 2단계 잠금 동시성 제어 메카니즘을 이용한 데이타 일관성 강화에 초점을 맞추어 일련의 스케줄링 정책을 서술하고 시뮬레이션을 통해 성능을 평가하였다[5]. 평가된 2가지의 트랜잭션 스케줄링 알고리즘은

1) 우선순위에 기초한 충돌 해결 정책(PBLP:Priority-Based conflict resolution Policy)
   잠금된 데이타 항목 중 어느 하나가 높은 우선순위의 트랜잭션에 의해 요구될 때 낮은 우선순위의 트랜잭션을 포기하는 방법
2) 우선순위 상속 프로토콜(PILP:Priority Inheritance Locking Protocol)
   블럭킹(blocking)하고 있는 모든 높은 우선순위의 트랜잭션중 가장 높은 우선순위를 낮은 우선순위의 트랜잭션에 상속시켜 먼저 수행하는 방법

이었다. L.Sha와 그의 동료들은, 교착상태를 제거하여 높은 우선순위 트랜잭션의 블럭킹 타임(blocking time)을 오로지 하나의 트랜잭션 수행시간에 맞추는 기법인 우선순위 실링 프로토콜(PCLP:priority ceiling locking protocol)을 소개하였다[6, 7]. 프로토콜 PCLP의 성능은 시뮬레이션을 통하여 평가되었다[8]. J.Huang과 그의 동료들은 RTDBMS에서의 처리장치 스케줄링(processor scheduling), 동시성 제어(concurrency control), 교착상태 해결, 트랜잭션 가동(wakeup), 트랜잭션 재시작(restart) 등을 처리하는 여러 실시간 처리 기법을 개발하고 평가하였다[9]. 동시성 제어를 위해서 우선순위에 기반한 잠금 프로토콜(PBLP:priority-based locking protocol)를 적용하였는 데, 우선순위

결정시의 파라미터들을 다르게 조합하여 여러 버전을 제시하였다. 이러한 파라미터는 마감시간(deadline), 중요도(criticalness), 잔존 실행 시간(remaining execution time)인데, 이 중 중요도(criticalness)는 실시간 처리의 중요성을 나타내는 것으로서 처리할 모든 트랜잭션을 똑같이 중요하다고 가정하고, 트랜잭션 실시간 우선순위를 결정하는 유일한 요소로서 트랜잭션 마감시간을 고려한 것이었다.

2. 기존의 트랜잭션 스케줄링 알고리즘

트랜잭션 각각은 자신이 읽기할 데이타 항목에 대해서는 공유된 잠금(shared lock)을, 자신이 쓰기할 데이타 항목에 대해서는 배타적 잠금(exclusive lock)을 가져야 한다. 2개의 잠금 요청은 그들이 같은 데이타 항목에 대하여 연산을 하고, 적어도 하나가 배타적 잠금일 때, 충돌이 된다. 잠금 요청이 충돌하게 되면, 잠금 요청 트랜잭션은 스케줄러에 의해 블럭킹된다. 만일 하나 이상의 트랜잭션이 특정한 잠금된 데이타 항목에 의해 블럭킹되어 있다면, 그 데이타 항목에 대한 잠금이 풀릴 때에, 가장 높은 우선순위의 트랜잭션이 먼저 처리되도록 허용된다. 만약에 하나의 데이타 항목이 하나 이상의 트랜잭션에 의해 읽기 모드에 잠금 되어있다면, 그 항목에 대한 새로운 트랜잭션의 읽기 잠금 요청은, 자신의 우선순위가 그 항목에 쓰기 잠금을 얻기 위해서 블럭킹 되어 있는 모든 트랜잭션의 우선순위보다 높을 때에만 받아들여진다. 블럭킹 교착상태(blocking deadlock) 탐지는 트랜잭션이 블럭킹될 때마다 수행된다. 교착상태가 탐지되면, 교착상태 사이클(deadlock cycle)상의 가장 낮은 우선 순위의 트랜잭션이 포기된다. 트랜잭션에 의해 얻어진 잠금들은 그 트랜잭션의 완료와 더불어 풀려진다. 쓰기 데이타 항목에 대하여 지연된 수정작업은 잠금이 풀리기 이전에 수행된다.

잠금 충돌을 방지하는 역할을 담당하는 스케줄러는 〈표 1〉과 같은 함수를 갖는다. 이 함수에서, *Transaction T*에는 고유 번호(ID), 우선순위, 트랜잭션 마감시간 들이 포함되고, *DataItem D*에는 데이타 우선순위가 포함된다. *TransactionPriority*이외에 *DataPriority*를 추가한 이유는 뒤에 다루어지는 우선순위 실링 프로토콜(priority-ceiling protocol)과 같이 데이타 항목에 관련이 있는 프로토콜을 포함하기 위해서 이다.

〈표 1〉잠금 프로토콜에서의 일반적 스케줄러 함수
〈Table 1〉 Generic scheduler function of locking protocols

TR lock-holding transaction
lock_request_handling(DataItem D, Transaction T)

2.1 항시 블럭 프로토콜(ABLP : Always block protocol)

항시 블럭 프로토콜은 엄격한 2단계 잠금 구조에 근거를 두고 있다[10]. 잠금 요구가 충돌을 일으키면, 잠금을 요구하는 트랜잭션들은 충돌하는 잠금을 유지하고 있는 스케줄러에 의해 블럭킹된다. 잠금 요청을 스케줄링할 때에 트랜잭션들의 실시간 특성은 고려하지 않는다. 즉, 잠금 요청들은 선입 선출(FIFO: First-in First-out) 순서에 의해 처리된다. 데이타 접근 스케줄링시 우선순위를 고려하는 잠금 프로토콜의 성능 이득을 측정할 때에, 항시 블럭 프로토콜이 이용된다. 프로토콜 ABLP를 포함한 모든 프로토콜에 대해서, 트랜잭션의 처리장치(processor) 요구는 실시간 우선순위에 기반하여 스케줄된다. CPU를 요구하는 높은 우선순위의 트랜잭션은 보다 낮은 우선순위의 트랜잭션 수행을 선점할 수 있다. 〈표 2〉는 프로토콜 ABLP에서의 스케줄러 함수를 보여 준다.

2.2 우선순위에 기반한 잠금 프로토콜(PBLP : Priority-Based Locking Protocol)

이 프로토콜에서는 높은 우선순위의 트랜잭션이 언제나 충돌시의 승자가 된다[5]. 필요할 때는 언제든지 낮은 우선순위의 트랜잭션을 포기함으로서 우선순위 바뀜 현상을 예방한다. 잠금 충돌을 해결하기 위해서, 잠금 요청 트랜잭션의 우선순위가 잠금을 가진 트랜잭션의 우선순위보다 높다면, 잠금을 가졌던 트랜잭션은 포기되고, 잠금 요청 트랜잭션이 잠금을 부여 받는다. 반대로 잠금 요청 트랜잭션의 우선순위가 보다 낮다면, 잠금 요청 트랜잭션은 높은 우선순위의 트랜잭션에 의해 블럭킹된다.

만약 데이타 항목이 트랜잭션 그룹에 의해 읽기 잠금 되어 있다면, 그 항목에 쓰기 잠금을 요청하는 트랜잭션 T는, 잠금을 공유하는 다른 트랜잭션이 T의 우선순위보다 높을 경우에 블럭킹 된다. 만약, T의 우

〈표 2〉 프로토콜ABLP에서의 스케쥴러 함수
〈Table 2〉 Scheduler function of protocol ABLP

```
TR          lock-holding transaction

lock_request_handling(D, T) {
   if ( D is locked) {
  block T
   }
   otherwise {
  Lock on D is granted to T
   }
}
```

〈표 3〉 프로토콜 PBLP에서의 스케쥴러 함수
〈Table 3〉 Scheduler function of protocol PBLP

```
TR          lock-holding transaction

lock_request_handling(D, T) {
   if ( D is not locked) {
  Lock on D is granted to T
   }
   else if ( Priority(T) > Priority(TR)){
  Lock on D is granted to T
   }
   otherwise {
  T is blocked
   }
}
```

선순위가 모든 잠금을 공유하는 트랜잭션의 우선순위보다 높다면, 잠금 공유 그룹에 존재하는 모든 트랜잭션은 포기된다.

높은 우선순위의 트랜잭션은 낮은 우선순위의 트랜잭션을 결코 기다리지 않는다는 조건은, 트랜잭션의 실시간 우선순위가 라이프타임동안 변하지 않으며 같은 우선순위를 갖는 2개의 트랜잭션은 존재하지 않는다고 가정할 때, 교착상태를 예방한다. 그러므로, 이 프로토콜은 스케줄러가 교착상태 감지 오버 헤드(deadlock detection overhead)를 없애 준다. 〈표 3〉에서 제시한 스케줄러 함수에서 잠금 요청간 충돌을 해결하는데 있어 가장 중요한 변수가 *TransactionPriority* 이다.

2.3 우선순위 상속 프로토콜(PILP : Priority Inheritance Protocol))

프로토콜 ABLP의 단점은 "우선순위 바뀜" 문제이다. 우선순위 바뀜 문제는 높은 우선순위의 트랜잭션이 낮은 우선순위의 트랜잭션에 의해 블럭킹되는 경우를 가리키는 것으로 프로토콜 PILP는 이 문제를 해결하기 위해서 제안된 방법이다[6]. 이 프로토콜은, 한 트랜잭션이 높은 우선순위의 트랜잭션을 블럭킹할 때, 자신이 블럭킹한 모든 트랜잭션들의 우선순위 중 가장 높은 우선순위를 상속 받게 하는 것이다. 상속에 의해 잠금을 가진 트랜잭션은 가장 높은 우선순위를 가지게 되어 잠금을 유지할 수 있다. 프로토콜 PILP는 높은 우선순위 트랜잭션의 블럭킹 시간을 단축시키는 목적을 지니고 있다.

우선순위를 상속받은 트랜잭션이 교착상태 때문에 포기될 때에, 그 트랜잭션은 원래의 우선순위로 환원된다. 잠금을 공유하고 있는 트랜잭션 그룹이 데이타 잠금을 보유하고 있으며, 데이타 항목에 대한 충돌 때문에 높은 우선순위의 트랜잭션이 블럭킹되어 있다면, 공유된 잠금 그룹에 존재하면서 블럭킹된 트랜잭션 보다 낮은 우선순위의 트랜잭션은 블럭킹 되어 있는 트랜잭션의 우선순위를 상속 받는다(〈표 4〉).

2.4 우선순위 실링 프로토콜(PCLP : Priority-Ceiling Protocol)

L.Sha와 그의 동료들이 제안한 우선순위 실링 프로토콜은 우선순위 상속 프로토콜(PILP)을 확장한 것이다[6, 7]. 이는 프로토콜 PILP에서의 교착상태 문제를 제거하였으며, 높은 우선순위 트랜잭션의 블럭킹 지연을 줄이고자 하는 것이다. 데이타 항목의 우선순위 실링은 그 항목에 잠금을 가질 수도 있는 가장 높은 우선순위의 트랜잭션의 우선순위이다. 데이타 항목에 잠금을 얻으려면, 데이타 항목의 우선순위 보다 높은 우선순위를 가져야 한다. 그렇지 않으면 블럭킹된다. 잠금을 가진 트랜잭션이 수행되고 영구 기억장치에 그 결과가 수록되면 데이타 항목의 우선순위 실링값은 재조정된다.

스케줄러가 시스템 내의 데이타 항목 각각에 대하여 트랜잭션 리스트를 관리하여야 한다. 이 리스트에는 데이타 항목에 접근하는 트랜잭션과 접근 예정인 트랜잭션의 고유번호와 우선순위가 포함된다.

이 리스트는 트랜잭션 우선순위에 기반해서 정렬되어 있다. 스케줄러는 잠금된 데이타 항목의 우선순위 실링 값과 잠금을 갖고 있는 트랜잭션의 고유번호를 추적해야 한다. 이 두 변수를 각각 MAX_PCLP, TR로 표시하면, 〈표 6〉은 집합 프로세스(cohort process)의 잠금 요청을 스케줄러가 어떻게 처리하는지를 보여 준다. MAX_PCLP와 TR의 현재 값을 정하

〈표4〉 프로토콜PILP에서의 스케줄러 함수
〈Table 4〉 Scheduler function of protocol PILP

```
TR          lock-holding transaction

lock_request_handling(D, T) {
   if (TR = NULL) {
  Lock on D is granted to T
   }
   else {
  T is blocked
  If ( Priority(T) > Priority(TR)){
     Priority(TR)=Priority(T)
  }}
}
```

〈표 5〉 프로토콜 PCLP에서의 트랜잭션 리스트
〈Table 5〉 Transaction list of protocol PCLP

```
데이타 항목 D 에 접근한 트랜잭션들의 리스트

  Transaction1(ID1, Priority1)
  Transaction2(ID2, Priority2)
  Transaction3(ID3, Priority3)
  Transaction4(ID4, Priority4)

Priority1 > Priority2 > ...
우선 순위의 ceiling 은 Priority1 이다.
```

기 위해서, 시스템 내에 있는 모든 잠금된 데이타 항목의 우선순위 실링과 잠금을 보유하고 있는 트랜잭션의 분류된 리스트를 스케줄러는 유지하고 있다.

RTDBMS에서 프로토콜 PCLP가 비현실적인 이유는 여러가지가 있다. 이중의 하나가 우선순위 블럭킹 원칙의 비관적 속성이다. 비록 동시성 트랜잭션 사이에서는 데이타 충돌이 발생하지 않는다 하더라도, 트랜잭션을 요청하는 접근의 대부분은 교착상태와 연쇄되는 블럭킹을 피할 수 있도록 블럭킹 되어진다.

〈표 6〉 프로토콜 PCLP에서의 잠금 요청 처리 과정
〈Table 6〉 Lock request handling process of protocol PCLP

```
lock_request_handling(D, T) {
   /* Transaction T requests a lock on data item D */
   if (priority(T) > MAX_PCLP) {
  Lock on D is granted to T;
   }
   else {
  T is blocked by TR;
  if ( priority(T) > priority(TR))
     priority(TR) = priority(T);
  }
   }
```

데이타 항목 각각에 대한 접근을 스케줄링 하는 것은 다른 데이타 항목에 접근하는 것과 독립적으로는 수행될 수 없다. 왜냐하면, 시스템내 모든 잠금 된 데이타 항목으로부터의 우선순위 정보를 스케줄러는 받아야 하기 때문이다.

이 프로토콜의 또 다른 단점은 CPU 스케줄링 정책인데, 이는 실행중인 트랜잭션 T가 선점과는 다른 이유(예로서 IO 요구)로 CPU를 놓을 때, CPU 대기행렬에 있는 다른 트랜잭션이 CPU를 점하지 못하도록 한다는 점이다. 트랜잭션 T가 다시 실행되거나 높은 우선순위의 트랜잭션이 CPU 대기행렬에 도달하기 전까지는 CPU가 쉬고 있는 것이다. CPU가 어떤 트랜잭션에 배정되어 있지 않으면, CPU 시간은 단순히 낭비되고 있는 것이다.

3. 데이타 우선순위에 기반한 잠금 프로토콜 (DPLP: Data-Priority-Based Locking Protocol)

본 논문에서는 데이타 항목에 우선순위를 부여하여 교착상태가 발생하지 않도록 트랜잭션 스케줄링을 할 수 있는 개선된 잠금 프로토콜을 제안한다. 각 데이타 항목은 해당 데이타 항목에 접근할 수 있는 모든 트랜잭션들 중 가장 높은 우선순위와 같은 우선순위를 갖는다. 새로운 트랜잭션이 시스템에 도달하면, 각 데이타 항목의 우선순위는, 자신이 해당 데이타 항목에 접근할 수 있는 트랜잭션의 우선순위보다

낮을 때에, 수정된다. 이 프로토콜은 접근되는 데이타 항목의 리스트가 도달하는 트랜잭션에 의해 스케줄러에게 통보된다고 가정한다. 트랜잭션이 종료될 때, 그 트랜잭션의 우선순위를 가졌던 각 데이타 항목은 그 항목에 접근하려 하는 나머지 트랜잭션들 중 가장 높은 우선순위를 가진 트랜잭션의 우선순위로 수정되어, 이것이 고유의 우선순위로 바뀐다. 또한, 각 트랜잭션은 유일한 우선순위를 갖는다고 가정한다.

〈표 7〉은 데이타 우선순위 관리의 내용을 보여 준다. 하나의 트랜잭션 리스트는 각각의 데이타 항목에 대하여 유지된다. 이에는 특정 데이타 항목에 접근을 요청하는 트랜잭션의 고유번호와 우선순위가 포함되어 있다. 이 리스트는 트랜잭션 우선순위에 기반하여 정렬되어 있으며, 가장 높은 우선순위의 트랜잭션이 데이타 항목의 우선순위를 결정한다. 이 리스트는 초기화시와 관련 트랜잭션이 실행 또는 포기될 때에 스케줄러에 의해 수정된다. 트랜잭션 리스트가 비어 있는 데이타 항목에는 어떤 트랜잭션에 배정될 수 있는 우선순위 값보다 낮은 값이 배정된다.

〈표 8〉에 제시한 절차는 트랜잭션의 잠금 요청이 프로토콜 DPLP에 의해서 처리되는 과정을 보여 준다. 트랜잭션 T가 데이타 항목 D에 잠금을 요청한다고 하자. 그 잠금을 얻기 위해서는 트랜잭션 T의 우선순위가 D의 우선순위와 같아야 한다. 즉, D의 현재의 우선순위에 책임을 질 수 있는 트랜잭션이 되어야 한다는 것이다. T의 우선순위가 D의 우선순위보다 작으면, T는 블럭킹된다.

〈표 7〉 프로토콜 DPLP : 초기화 및 트랜잭션 종료 시 스케줄러에 의한 데이타 우선 순위 처리 과정

〈Table 7〉 Protocol DPLP : Data priority handling by scheduler when a transaction is initialized or terminated

```
data_priority_handling(T) {
/* Transaction T is being initialized or terminated */
if T is a new transaction being initialized
for each data item D in T's access list
    if(priority(T) > priority(D))
            priority(D) = priority(T);
otherwise /* T is being committed or aborted */
    for each data item D accessed by T
            if(priority(T) = priority(D))
                    priority(D) = priority(highest priority active
                                            transaction which will
                                            access D);
}
```

데이타 항목 D는, 접근 리스트에 D를 포함하면서도 D의 우선순위보다 높은 우선순위를 갖는 새로운 트랜잭션 T′가 시스템에 도달해서 D의 우선순위를 수정할 때에, 트랜잭션 T′에 의해서 잠금되어질 수 있다. 만약 T가 D에 접근할 필요가 있을 시점에 D가 T′에 의해 잠금되어 있다면, T의 우선순위가 T′의 우선순위보다 높은 경우에, 낮은 우선순위의 트랜잭션 T′는 포기되어지고 T가 D에 잠금을 얻는다. 각 트랜잭션이 유일한 우선순위를 갖는다는 가정은, 높은 우선순위의 트랜잭션은 결코 낮은 우선순위의 트랜잭션에 의해 블럭킹되지 않기 때문에, 프로토콜 DPLP가 교착상태 방지된다(deadlock-free).

〈표 8〉 프로토콜 DPLP : 잠금 요청 처리 과정

〈Table 8〉 Protocol DPLP: Lock request handling process

```
lock_request_handling(D,T)
/* Transaction T requests a lock on data item D */
if(priority(T) = priority(D))
    if (D was locked by a transaction T')
            T' is aborted;
            Lock on D is granted to T;
}
otherwise
    T is blocked by the transaction that determines the current
    priority D;
}
```

프로토콜 DPLP는 읽기/쓰기 잠금 의미론으로 확장될 수 있다. 이러한 확장에 대하여, 각 데이타 항목은 두 개의 우선순위 값, 하나는 읽기 접근에 대한 값, 하나는 쓰기 접근에 대한 값에 연관되어 있다. 어떤 데이타 항목 D에 읽기 잠금을 얻기 위해서는, 트랜잭션 T의 우선순위 값이 D의 쓰기 우선순위 값보다 크거나 같아야 한다. 데이타 항목 D에 대한 트랜잭션 T의 쓰기 잠금 요청은, T의 우선순위가 D의 쓰기 잠금 우선순위와는 같으며 D의 읽기 우선순위보다는 크거나 같은 경우에 받아들여진다. 트랜잭션 T가 D에 대한 쓰기 잠금을 얻을 때, 만약 D가 트랜잭션 그룹에 의해 이미 쓰기 잠금되어 있다면, 읽기 잠금 그룹에 있는 모든 트랜잭션들은 포기된다. 읽기/쓰기 요청을 처리하는 절차가 〈표 9〉에 상세히 표현되어 있다.

프로토콜 PCLP에 비해서, 프로토콜 DPLP는 RT-DBMS에서 구현하기가 보다 실제적이다. 앞서 설명

〈표 9〉 프로토콜 DPLP에서 읽기/쓰기 요청을 처리하는 절차
〈Table 9〉 read/write lock request handling process of protocol DPLP

```
Handling READ LOCK requests
        read_lock_request_handling(D,T) {
         /* Transaction T requests a read lock on data item D */
         if(priority(T) >= write-priority(D)) {
           if(D was write-locked by a transaction T')
             T' is aborted;
           Read lock on D is granted to T;
         }
         otherwise
           T is blocked by the transaction that has assigned the write priority of D;
        }

        write_lock_request_handling(D,T) {
        /* Transaction T requests a write lock on data item D */
         if(priority(T) = write-priority(D) and priority(T) >= read-priority(D)) {
           if(D was read or write locked by any transaction T')
             T' is aborted;
           Write lock on D is granted to T;
         }
         otherwise
           T is blocked by the transaction that has assigned
           the maximum of read and write priorities of D;
        }
```

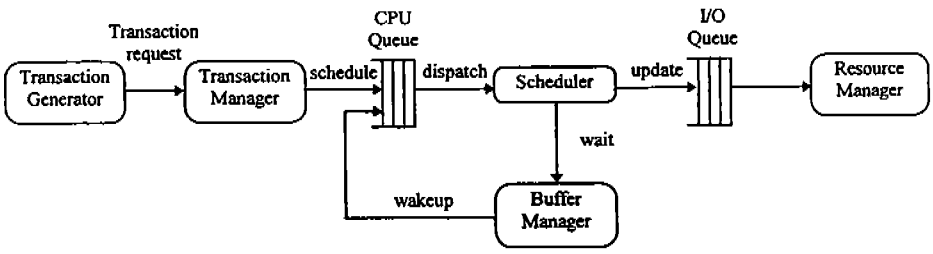
한 바와 같이 프로토콜 PCLP의 주요한 결점은 우리가 제시하는 새로운 프로토콜에 의해 제거된다. 하나의 데이타 항목에 대한 트랜잭션의 접근 요청을 순서화 할 때에는 오로지 그 항목의 우선순위만이 고려된다. 즉, 스케줄링 의사결정은 다른 데이타 항목의 우선순위와 독립적이다. 더군다나, 트랜잭션은 데이타 충돌의 경우에만 데이타 접근 요청에 의해 블럭킹된다.

4. 시뮬레이션 모델 및 환경

시뮬레이터는 대기행렬 모델에 근거를 두고 있으며, 트랜잭션들에 대한 2개의 공유된 물리적 자원 즉, CPU와 디스크를 포함하고 있다. 모델은 5개의 구성요소-트랜잭션 발생기, 트랜잭션 관리기, 스케줄러, 버퍼 관리기, 자원관리기로 구성되었다.

트랜잭션 발생기는 트랜잭션의 도달(진입)을 시뮬레이션하고, 시스템 파라미터 값을 이용하여 각 트랜잭션의 형태, 마감시간, 데이타 접근 리스트를 정한다. 트랜잭션 관리기는 트랜잭션 고유번호들을 발생시키는 역할과 트랜잭션들에게 실시간 우선순위를 부여하는 역할을 수행하고 있다. 시스템에 진입된 각 트랜잭션은 마감시간 형태의 실시간 제약 조건에 연관되어 있다. 프로토콜 PCLP와 프로토콜 DPLP를 제외한 모든 트랜잭션 스케줄링 알고리즘들에 대해서는, 스케줄링 의사결정에 이용하기 위한 유일한 정보로서 마감시간이 이용되는 데, 이는 시스템에 진입하는 트랜잭션에 의해 제공된다. 각 트랜잭션은 고유의 마감시간에 의해서 유일한 우선순위를 부여 받는다. 별다르게 지정되지 않는 한, 가장 급한 마감시간을 가진 트랜잭션에 우선순위를 주는 배정 방식 즉, 보다 급한 마감시간을 가진 트랜잭션이 그렇지 않은 트랜잭션보다 높은 우선순위를 갖는 방식이 채택된다. 같은 마감시간을 갖는 2개의 트랜잭션에 대해서는 시스템에 좀 더 일찍 진입한 트랜잭션이 보다 높은 우선순위를 배정 받는다. 트랜잭션 마감시간은 신축적이다. 즉, 각 트랜잭션은 자기의 마감시간을 넘긴다고 하더라도, 완성될 때까지 수행된다. 각 트랜잭션은 특정한 데이타 항목에 대하여 하나 이상의 데이타베이스 연산(읽기/쓰기)을 수행한다.

트랜잭션들의 동시적 데이타 접근 요청은 스케줄러에 의해 통제된다. 스케줄러는 수행되는 알고리즘에 근거해 데이타 접근을 순서화 한다. 트랜잭션의 접근 요청은 트랜잭션의 실시간 우선순위에 기반하여, 승낙, 블럭킹, 혹은 포기 된다. 접근 요청이 승낙될 경우, 트랜잭션은 메인 메모리에서 해당 데이타 항목을 읽으려 한다. 데이타가 메인 메모리에 없다면,



(그림 1) 시뮬레이션 모델 및 환경
(Fig. 1) Simulation model and environment

트랜잭션은 데이타가 디스크로부터 메인 메모리로 이전될 때까지 기다린다. 디스크와 메인 메모리 사이의 데이타 이전은 버퍼 관리기에 의하여 이루어진다. 메모리 버퍼의 관리를 위해서는 선입 선출 페이지 대체 전략(FIFO page replacement strategy)이 이용된다. 접근을 추적하여 그 데이타 항목이 처리되어진다. 쓰기 연산은 수정된 데이타를 디스크에 쓰기 위한 입출력(IO : Input/Output)활동을 제외하고는 읽기 연산과 유사하게 처리된다. 트랜잭션의 수정 사항을 수행 이후의 시점까지 유지하기 위해 충분한 버퍼 공간이 가정된다. 어떤 트랜잭션이 수행 중의 어느 순간에 포기된다면 그에 의해서 수정된 모든 내용은 무시된다. 포기된 트랜잭션은 같은 충돌이 반복적으로 발생하는 것을 예방하기 위하여 어떤 재시작 지연 이후에 재시작 된다. 재시작된 트랜잭션은 같은 데이타 항목을 전과 같은 방법으로 접근한다. 한 트랜잭션이 모든 자료 접근과 처리 요구 사항을 완성하였을 때 종료될 수 있다. 자료 접근과 처리 요구 사항이 질의어(query/read-only)라면 트랜잭션은 종료된다. 만약 트랜잭션이 수행 중 하나 이상의 데이타 항목을 수정하였다면, 수정 사항을 데이타베이스에 쓰기 위해서 입출력 대기행렬(IO queue)에 들어간다.

자원 관리기는 데이타 항목의 읽기/쓰기를 위해서는 입출력 서비스를, 데이타 항목을 처리하고 동시성 제어 연산(충돌 체크, 잠금 등)을 수행하기 위해서는 CPU 서비스를 제공하는 역할을 한다. CPU 행렬과 입출력 행렬은 둘 다 트랜잭션의 실시간 우선순위에 기반하여 구성되어 있다. 선점 예약 우선순위 스케줄링(Preemptive-resume priority scheduling)은 CPU에 의해 이용된다. 즉, 보다 높은 우선순위의 프로세스는 보다 낮은 우선순위의 프로세스를 선점하며, 보다 낮은 우선순위의 프로세스는 CPU를 기다리는 보다 높은 우선순위의 프로세스가 없을 때에 다시 시작될 수 있다. 선점 현상 이외에도, CPU는 잠금 충돌의 결과로서 혹은 IO를 위한 트랜잭션에 의해서 릴리즈(release)될 수 있다. 다음의 파라미터는 시스템 형상과 작업 부하를 지정하기 위해서 이용된다. 데이타베이스의 크기(db_size)는 데이타베이스에 저장되어 있는 데이타 항목의 갯수에 관련되며, 메모리의 크기(mem_size)는 주기억 장치에 유지될 수 있는 데이타 항목의 갯수이다. IAT(inter arrival time)는 트랜잭션들의 평균 도착 시간 간격이다. 도착률은 포아송 분포(Poisson Distribution)를 가정한다. 트랜잭션의 형태(read-only 또는 update)는 수정 형태 확률을 지정하는 파라미터 tr_type_prob을 이용해서 랜덤하게 결정된다. 트랜잭션에 의해서 접근되는 데이타 항목의 갯수는 파라미터 access_mean에 의해서 결정된다. 데이타 항목의 갯수의 분포는 지수 분포를 이룬다. 수정 트랜잭션의 각 데이타 접근에 대하여, 데이타 항목이 수정될 확률은 파라미터 data_update_prob에 의해서 정해진다. 각 새로운 트랜잭션에 대하여 그 트랜잭션에 유일한 실시간 우선순위를 배정하는 초기의 CPU 비용이 발생한다.

이러한 처리 비용은 파라미터 pri_assign_cost에 의해서 시뮬레이션 된다. 트랜잭션들에 대한 CPU와 IO 시간은 CPU-IO overlap을 달성하기 위한 목적으로 분리되어 고려될 수 있다. 하나의 데이타 항목을 처리하는 CPU 시간은 파라미터 cpu_time에 의해서 지정되며, 디스크에 있는 데이타 항목에 접근하는 시간은 파라미터 io_time에 의해서 정해진다. 이러한 파라미터들은 일정한 서비스 시간 요구를 나타낸다. 시스템에서 트랜잭션 오버 헤드의 가능성을 예방하기 위해 시스템 내의 능동적 트랜잭션의 총 갯수가 파라미터 max_act_tr에 의해서 제한된다. 이러한 제한이 달성되었을 때 도달된 트랜잭션은 일시적으로 이전된다. 프로토콜들을 공정하게 평가하기 위해서는, 여러 동시성 제어 연산들을 수행하는 오버 헤드가 고려되어야 한다. 우리가 제안하는 모델은 다음의 연산 처리 비용을 고려하고 있다.

- 충돌 체크 : 한 트랜잭션에 대한 여러 자료 접근 요구 사이의 충돌 체크
- 잠금 : 데이타 항목에 잠금을 획득하는 것
- 풀림 : 데이타 항목에 대한 잠금을 푸는 것
- 교착상태 감지 : wait-for graph에서의 교착상태감지
- 교착상태 해결 : 교착상태를 해결함
- 리스트 조작 : 동시성 제어 목적으로 이용되는 그래프/리스트의 여러 형태의 연산을 삽입하고 삭제함. 이는 교착상태 감지를 위해 유지되는 WFG (wait-for graph), 프로토콜 PCLP에서 잠금된 데이타 항목의 우선순위 실링 리스트, 프로토콜 PCLP와 DPLP에서 각 데이타 항목을 위해 유지되는 트랜잭션 리스트를 포함하고 있다.

- 검증 테스트(낙관적 프로토콜 OP에서): 검증하는 트랜잭션의 라이프타임동안 실행된 트랜잭션들 각각에 대하여 트랜잭션의 검증 테스트를 수행함(즉, 검증하는 트랜잭션의 읽기 집합과 수행된 트랜잭션들의 각각의 쓰기 집합 사이의 공통적 요소를 체크하는 것)

이러한 동시성 제어 연산들의 각각은 많은 기본적인 연산들을 수행함으로서 실행된다. 기본적 연산의 집합에는 테이블 참조, 테이블 요소 값 설정, 어떤 두 값의 비교, 리스트 요소의 포인터 설정 등이 포함된다. 모든 동시성 제어 정보는 주기억 장치에 저장되어 있다고 가정한다. 잠금 프로토콜을 위한 충돌 체크 연산은 요구된 데이타 항목이 이미 체크되었는지의 여부를 결정하기 위해서 테이블 참조를 수행한다. 프로토콜의 충돌 감지 정책에 의존하면서, 테이블 참조 연산은 잠금 요구 트랜잭션과 데이타 항목의 우선순위를 얻기 위해서 수행될 수 있다. 그리고, 테이블 참조는 우선순위 비교 연산에 의해서 수행되어진다. 시간소인 순서화 프로토콜의 경우에, 충돌 체크 연산은 요구된 데이타 항목의 시간 소인 값을 얻기 위해서는 테이블 참조 연산을 수행하고, 요구하는 트랜잭션의 시간 소인을 각 데이타 항목의 시간 소인과 비교하기 위해서는 비교 연산을 수행한다. 잠금과 풀림 연산은 데이타 항목의 잠금 플래그(flag)를 설정/재설정 하기 위해서 테이블 요소 값 설정 연산을 수행한다. 교착상태 감지는, WFG로부터의 정보를 얻기 위해서는 많은 테이블 참조 연산을, 교착상태 싸이클을 체크할 때 트랜잭션 고유 번호들을 비교하기 위해서는 많은 비교 연산을 요구한다. 교착상태으로부터 회복되기 위해서는, 싸이클 내에서의 최소 우선순위 트랜잭션이 포기된다. 한 요소를 분류된 리스트에 삽입하기 위해서는, 삽입점을 결정하기 위한 약간의 비교 연산과 그 요소를 그 자리에 삽입하기 위한 포인터 설정 연산이 필요하다. 삽입을 위한 리스트에서 적당한 장소를 찾기 위해서는 이진 탐색이 이용된다. 분류된 리스트로부터 한 요소를 삭제하기 위해서는 탐색을 위한 비교 연산과 삭제를 위한 포인터 설정 연산이 필요하다. 삽입/삭제 연산은 또한 비교시에 이용되는 리스트 요소의 고유 번호 혹은 우선순위를 얻기 위해서 테이블 참조를 수행한다. 어떤 트랜잭션 리스트/그래프에서 수행된 비교 연산의 갯수는 리스트/그래프의 싸이즈 즉, 시스템 내 그 트랜잭션의 부하의 크기에 달려 있다. 낙관적 프로토콜에서는, 각 트랜잭션의 검증 단계 동안에, 두개의 데이타 집합에 공통 요소가 존재함을 확인하기 위해서, 이 두 데이타 집합을 교차시켜 검증 테스트를 수행한다. 교차 알고리즘은 많은 비교 연산을 필요로 한다. 한 트랜잭션을 위해 수행된 검증 테스트의 갯수는 검증하는 트랜잭션의 라이프타임동안 수행되어진 트랜잭션 수와 같다. 검증 테스트가 수행되어질 트랜잭션을 정하기 위해서 약간의 테이블 참조와 비교 연산이 수행될 필요가 있다. 기본 연산의 각각을 수행하는 비용이 대략 서로 비교할 만 하다고 가정하면, 기본 연산 처리 비용은 하나의 파라미터 basic_op_cost를 사용함으로서 시뮬레이션 된다.

파라미터 slack_rate는 새로운 트랜잭션에 마감시간을 배정할 때 사용되는 파라미터이다. 한 트랜잭션의 여유 시간은 파라미터 slack_rate 시간 즉, 트랜잭션의 추정된 처리 시간의 평균을 가진 지수분포로부터 무작위하게 선택되어진다. 트랜잭션 처리기가 마감시간을 배정할 때, 트랜잭션 처리 시간의 추정치를 사용하는 반면에, 우리는 시스템 자체는 처리 시간 정보에 대한 지식이 부족하다고 가정한다. 트랜잭션 T의 마감시간은 다음의 공식에 의해 정해진다.

$$\text{deadline}(T) = \text{start\_time} + \text{processing\_time\_estimate}(T) + \text{slack\_time}(T) \quad (4.1)$$

식 (4.1)에서 변수 slack_time(T)은 slack_rate와 processing_time_estimate(T)의 곱의 지수형(exponential)이며, processing_time_estimate(T)은 CPU_requirement(T)와 IO_requirement(T)의 합이다.

items(T)가 트랜잭션 T에 의해 접근되는 데이타 항목의 실제 숫자를 나타낸다고 하면,

$$\text{CPU\_requirement}(T) = \text{items}(T) * \text{CPT\_time} \quad (4.2)$$

query와 수정 트랜잭션에 대해서는 각각 다음과 같이 주어진다.

$$\text{IO\_requirement}(T) = \text{items}(T) * (1 - \text{mem\_size}/\text{db\_size}) * \text{io\_time} \quad (4.3)$$

$$IO\_requirement(T) = items(T)*(1\text{-}mem\_size/db\_size)*io\_time + w\_items(T)*io\_time \quad (4.4)$$

w_items(T)는 T에 의해 수정되는 데이타 항목의 실제 숫자를 의미한다.

5. 성능 평가 결과

모든 트랜잭션 스케줄링 알고리즘에 공통적인 시스템 환경 및 트랜잭션 파라미터 값이 〈표 10〉에 제시되어 있다. 별도로 서술하지 않는 한 이들은 시뮬레이션시 사용되는 값들이다. 이들은 특정 데이타베이스를 시뮬레이션 하려고 의도한 것이 아니고, 알고리즘 사이의 실시간 성능상의 차이점을 충분히 관찰하기 위하여 시스템 부하 및 데이타 충돌 수위를 만족시키기 위하여 선택된 파라미터 값들이다. 스케줄링 알고리즘들은 트랜잭션들 사이의 데이타 접근 충돌을 처리할 때에 상이한 점이 있기 때문에, 알고리즘 성능 특성의 비교시 가장 좋은 방법은 높은 자료 충돌 조건하에서 시뮬레이션을 실시하는 것이다. db_size의 값이 작은 것은 트랜잭션들 사이의 원하는 높은 자료 충돌 수준을 제공하는 자료 충돌 환경을 창출하려는 의도 때문이다. 이렇게 작은 데이타베이스는 보다 더 큰 데이타베이스의 가장 자주 접근되는 부분으로서 고려될 수 있다. cpu_time과 io_time의 값들은 시스템내에 가장 이상적인 CPU와 IO 이용도를 얻기 위해 선택되어졌다. 부하가 없는 시스템에서 기대되는 IO 이용도(U_{io})는 다음과 같이 계산된다.

$$U_{io} = \frac{1}{iat} * access\_mean * \left(1 - \frac{mem\_size}{db\_size}\right) * io\_time + \frac{1}{iat} * access\_mean * tr\_type\_prob * data\_update\_prob * time \quad (5.1)$$

식 (5.1)의 첫번 째 항은 디스크로부터 데이타를 읽는 것에 대응되며, 두번째 항은 어떤 트랜잭션의 수행시 연기되었던 수정 사항을 데이타베이스에 쓰는 것에 대응된다.

부하가 없는 시스템에서의 CPU 이용도 (U_{cpu})는 다음 공식에 의해 추정된다.

$$U_{cpu} = \frac{1}{iat} * access\_mean * cpu\_time \quad (5.2)$$

시뮬레이션시 사용된 IAT 값의 범위는 시스템 내에서 높은 수준의 CPU 이용도와 IO 부하 수준에 상응한다. 파라미터 basic_op_cost는 여러 동시성 제어 연산들에 의해서 사용되는 어떤 기본적인 연산이라도 수용할 수 있을 정도로 큰 값이 배정된다.

시뮬레이션은 각 프로토콜에 특정한 데이타구조(리스트, 그래프)를 유지한다. 동시성 제어 목적을 위해 이용한 모든 데이타 구조는 주기억장치에 저장되어 있다고 가정되기 때문에, 이들에 대한 접근을 시뮬레이션할 경우 입출력 지연은 포함되지 않는다. 그러나, 리스트/그래프에 대한 여러 연산의 처리 비용은 앞의 장에서 상세히 설명된 대로 고려되어진다. 시뮬레이션 프로그램은 데이타 충돌, 잠금에 대한 대기행렬, CPU와 IO의 대기행렬 길이, CPU에서의 처리 지연, 디스크 IO에 대한 처리 지연, 트랜잭션의 포기/재시작, 모델에서 고려되는 모든 동시성 제어 오버 헤드를 명백하게 시뮬레이션한다.

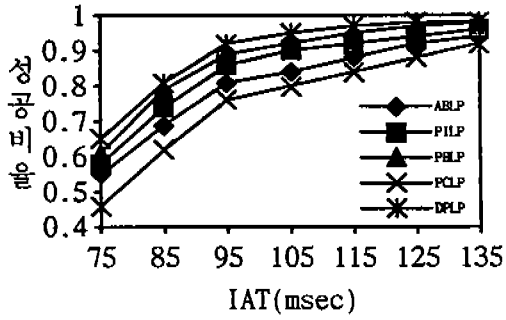
〈표 10〉 성능 평가에 사용된 변수
〈Table 10〉 Configuration parameters used for performance evaluation

Configuration Parameters		
db_size	데이타베이스 크기	200
mem_size	메모리 크기	50 %
cpu_time	CPU 소요 시간	12msec(constant)
io_time	IO 소요 시간	12msec(constant)
max_tr_nr	최대트랜잭션수	20
pri_assign_cost	초기배정비용	1msec(constant)
basic_op_cost	기본연산비용	0.1msec(constant)
Transaction Parameters		
iat	트랜잭션도착시간차	100msec(exponential)
tr_type_pro	트랜잭션형태비	0.5
access_mean	접근평균	6(exponential)
data_update_prob	데이타갱신비	0.5
slack_rate	여유시간비율	5(exponential)

실시간 데이타베이스 시스템에 근거한 시뮬레이션 모델은 트랜잭션 스케줄링 알고리즘들의 실시간 성능을 평가할 때 이용하였다. 데이타베이스에 있는 각 데이타 항목은 각각 시뮬레이션 된다. 프로그램은 또

한 주기억장치에 상주하고 있는 데이타 항목들의 리스트를 추적한다. 수행된 실험의 각 RUN은 500 트랜잭션이 수행될 때까지 계속되었다. 시뮬레이션 결과를 검증하기 위해서 독립적 반복 방법을 이용하였는데, 이는 각 형상을 서로 다른 난수 초기화 변수(seeds)로 25회씩 런(run)한 후, 복제 평균의 평균치를 최종 추정치로 활용하는 것이다. 독립적 관찰의 가정하에 성립된 결과에 대하여 95%의 신뢰구간을 구하였다. 다음의 절에서는 오로지 통계적으로 유의한 성능 결과를 논한다. 각 통계적 데이타 포인트의 신뢰구간의 넓이는 점추정치의 3%이하이다. 그려진 그래프에는 성능 결과들의 평균값만을 플로팅(PLOTTING) 하였다.

성능 평가 기준으로는 성공비율(success ratio), 평균 지연(average latencss), 충돌 비율(conflict ratio), 재시작 비율(restart ratio)을 이용하였다. 성공 비율은 마감시간 제약조건을 만족시킨 트랜잭션 수와 총 트랜잭션 수와의 비율이며, 평균 지연은 마감시간을 지키지 못한 트랜잭션들의 평균 지연 시간이며, 충돌 비율은 처리된 트랜잭션의 수와 관찰된 충돌 갯수와의 비율이며, 재시작 비율은 처리된 트랜잭션 갯수와 관찰된 재시작 트랜잭션 갯수와의 비율이다.

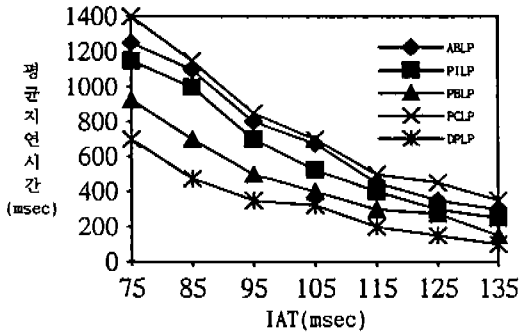


(그림 2) 잠금 프로토콜들의 *IAT*의 변화에 따른 성공비율 결과값
(Fig. 2) success ratio to *IAT* of locking protocols

시스템의 트랜잭션 부하 변화에 따른 트랜잭션 스케줄링 알고리즘들의 실시간 성능 특성을 조사하였다. 파라미터 IAT는 75msec-135msec까지 10msec의 간격으로 변화시켰다. 이 값들은 0.96-0.53의 IO 이용도에 대응한다(식 5.1). 시뮬레이션시 선택된 파라미터 값들에 대해서는 CPU 이용도와 IO 이용도가 거의 같은 값이었다.

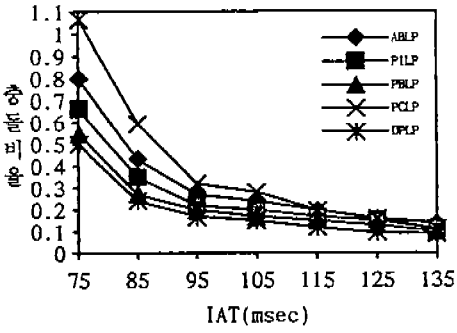
시뮬레이션에서 얻은 결과는 프로토콜 PCLP에 의한 성능이 상대적으로 낮다는 점이며(그림 2,3 참조). 데이타 접근을 스케줄링 할 때 실시간 우선순위를 고려하지 않는 프로토콜 ABLP마저 프로토콜 PCLP보다는 좋은 성능을 보여 준다. 프로토콜 설명시 요약한 대로 PCLP 구현 시의 단점이 이러한 불만족스러운 결과를 초래하였다. 우선순위 실링 블럭킹 규칙의 제한적 속성은, 블럭킹 되지 않은 트랜잭션에 대해서는 데이타 잠금을 허용하지 않는다는 점이다. 프로토콜 PCLP는 충돌 비율의 수준이 높은 데, 이는 데이타 충돌이라기 보다는 우선순위 실링 충돌(각 트랜잭션이 우선순위 실링 규칙에 블럭킹되는 횟수의 평균치) 때문이다. 트랜잭션 블록의 수가 너무 크면 동시성이 낮게 되고 자원 이용도가 낮아진다. 특히 트랜잭션 부하가 높을 경우 많은 트랜잭션들은 마감시간을 치키지 못한다.

본 논문에서의 결과를 분석해 보면, 프로토콜 PILP가 프로토콜 ABLP보다는 상당한 진전이 있다는 사실을 알게 된다. 이러한 진전은 우선순위 상속 방법을 적용하여 높은 우선순위 트랜잭션들의 블럭킹타임을 줄인 때문이다. 그러나, 프로토콜 PILP의 성능은 프로토콜 PBLP의 성능 수준에는 미치지 못한다. 프로토콜 PBLP는 보다 높은 우선순위 트랜잭션들을 결코 블럭킹하지 않으며, 필요할 때는 낮은 우선순위 트랜잭션들을 포기한다. 프로토콜 PBLP는 또한 교착상태의 가능성과 비용을 제거한다.



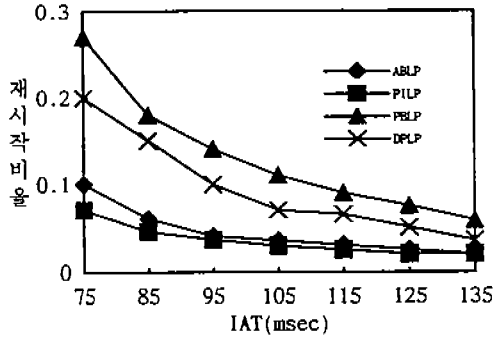
(그림 3) 잠금 프로토콜들의 *IAT* 변화에 따른 *average-lateness* (msec) 결과값
(Fig. 3) *average-lateness* (msec) to *IAT* locking prtocols

본 논문에서 제안하는 새로운 프로토콜 DPLP의 성능은, 특히 트랜잭션 부하가 높을 경우, 프로토콜 PBLP의 성능보다 우수하다. 프로토콜 PBLP에서 시뮬레이션된 얼마간의 트랜잭션 포기와 그에 따르는 자원 낭비는 접근 스케줄링시 자료 요구 사항에 대한 사전 지식을 채택하는 프로토콜 DPLP에 의해서 예방될 수 있다. 프로토콜 DPLP에서는 어떤 두개의 충돌하는 트랜잭션 사이에서 낮은 우선순위의 트랜잭션은 높은 우선순위 트랜잭션이 시스템에 진입하기 이전에, 그 데이타 항목의 우선순위가 높은 우선순위 트랜잭션의 진입에 의해서 수정되기 이전에, 그 데이타 항목을 접근하는 경우에 대해서만으로 포기의 가능성을 제한한다. 이러한 특별한 경우에 대해서, 낮은 우선순위 트랜잭션의 수행 이전에 높은 우선순위 트랜잭션이 그 데이타 항목을 접근할 때 그리고 접근한다면, 낮은 우선순위 트랜잭션은 포기된다.



(그림 4) 잠금 프로토콜들의 *IAT* 변화에 따른 *conflict-ratio* 결과값

(Fig. 4) *conflict-ratio* to IAT of locking protocols



(그림 5) 잠금 프로토콜들의 *IAT* 변화에 따른 재시작 비율

(Fig. 5) Restart ratio to IAT locking protocols

(그림 4)와 (그림 5)는 서로 다른 IAT 값에 대한 프로토콜들의 충돌 비율과 재시작 비율을 보여 준다. 프로토콜 PCLP의 재시작 비율은 늘 0이기 때문에 (그림 5)에는 프로토콜 PCLP가 나타나지 않는다. 프로토콜 PBLP와 프로토콜 DPLP의 충돌 비율이 비록 다른 프로토콜들의 그것보다 낮다고 하더라도, 재시작의 갯수는 프로토콜 ABLP와 프로토콜 PILP에 비해서 훨씬 큰 것으로 관찰되었다. 프로토콜 PBLP와 프로토콜 DPLP에서는 데이타 충돌을 풀기 위해서 트랜잭션들을 재시작하지만, 프로토콜 ABLP와 프로토콜 PILP에서 재시작의 유일한 기준은 교착상태이다.

6. 결 론

본 논문에서는 기존의 알고리즘들이 트랜잭션에 부여하였던 우선순위를 특정 데이타 항목에 부여하여 접근하는 트랜잭션중 가장 높은 우선순위의 트랜잭션을 먼저 수행토록 함으로써 교착상태를 예방할 수 있는 점에 착안하여 데이타 항목에 우선순위를 부여하는 데이타 우선순위에 기초한 잠금 프로토콜(DPLP:Data Priority Based Locking Protocol)을 제안하고, 이 알고리즘의 성능을 실시간 데이타베이스 시스템에서 기존 잠금 프로토콜의 성능과 비교분석하였다. 비교 기준으로는 트랜잭션 도착 시간차(IAT: inter arrival time)에 따른 트랜잭션의 성공 비율(success ratio), 평균 지연(average-lateness), 충돌 비율(conflict ratio), 재시작 비율(restart ratio)을 채택하였다. 성능 비교의 결과는 본 논문에서 제안하는 데이타 우선순위에 기초한 잠금 프로토콜이 기존의 잠금 프로토콜들보다 우수한 성능을 보임을 확인하였다.

자료의 총 수(*db_size*)는 200으로 비교적 작은 크기의 데이타베이스를 대상으로 하였는 데, 이는 높은 데이타 충돌을 유발하기 위함이었으며, 보다 큰 데이타베이스 시스템의 접근이 잦은 부분(Hot Spot)으로 고려하려는 의도도 있었다. 데이타베이스 크기의 변화는 알고리즘간의 성능 비교에서는 그리 독자적인 중요성을 갖지 않는다. 이는 〈표 10〉의 여러 변수가 결과적으로는 독립적인 결과를 낳지 않으며, 데이타 충돌을 유도한다는 관점에서 볼 때, 한 변수는 다른 변수와 종속적인 면을 갖는 점에서 기인한다.

시뮬레이션을 통하여 얻은 결론들을 다음과 같이

요약한다.

실시간 성능의 관점에서, 프로토콜 PILP와 프로토콜 PBLP는 둘 다 2단계 잠금의 기본적 버전인 프로토콜 ABLP보다 우수하였다. 프로토콜 PILP는, 자신이 블럭킹하고 있는 모든 높은 우선순위 트랜잭션들 중 가장 높은 우선 순위로, 낮은 우선순위의 트랜잭션을 수행하게끔 허용한다. 프로토콜 PBLP는 잠금된 데이타 중 어느 하나가 보다 높은 우선순위의 트랜잭션에 의해서 요청될 때, 낮은 우선순위의 트랜잭션을 포기한다. 보다 높은 우선순위의 트랜잭션을 블럭킹하는 것 보다는 낮은 우선순위의 트랜잭션을 포기함으로써 보다 많은 트랜잭션들이 그들의 마감시간 이내에서 수행되도록 도와줄 수 있다는 결론을 유도하면 프로토콜 PBLP가 프로토콜 PILP보다 성능이 좋다고 결론 지을 수 있다. 프로토콜 PCLP는 우리가 시뮬레이션한 알고리즘 중 가장 낮은 성능을 보였다. 데이타 충돌이 없음에도 불구하고 잠금을 요구하는 트랜잭션들을 블럭킹 할 수 있는 프로토콜 PCLP의 규칙이 너무 제약적이어서 RTDBMS에서는 구현할 수가 없다. 프로토콜 PCLP는 한 순간에 단지 몇 개의 트랜잭션만을 활성화 함으로써 낮은 동시성과 자원 이용도를 지원한다. 특히, 높은 트랜잭션 부하의 경우에 대부분의 트랜잭션들은 마감시간을 맞추지 못하며 큰 지체 시간을 갖는다.

본 논문에서 제안한 새로운 데이타 우선순위에 기반한 잠금 프로토콜은, 만약 각 트랜잭션의 데이타 접근 리스트를 트랜잭션의 수행에 우선해서 스케줄러가 알 수 있다면, 프로토콜 PBLP에 의해서 달성된 수준보다 높은 실시간 성능이 가능하다는 점을 증명하였다. 프로토콜 DPLP는 트랜잭션 부하, 데이타충돌, 자원 충돌의 여러 수준하에서 모든 다른 잠금 프로토콜들 보다 성능이 우수하였다.

초고속 정보통신망의 구축과 더불어 향후의 컴퓨팅 환경이 분산처리 및 네트워크 컴퓨팅 환경으로 변화할 것임을 고려할 때, 또한 공동작업을 위한 그룹웨어의 개발이 현실화 되고 있음을 감안할 때, 분산된 환경에서의 트랜잭션 스케줄링 알고리즘은 이러한 기술 개발에 필수적인 것으로 판단되며, 이를 위한 연구개발이 지속적으로 수반되어야 할 것으로 생각한다.

참 고 문 헌

[1] K. P. Eswaren, J. N. Gray, "The Notion of Consistency and Recovery in a Database System", Comm. of ACM, Vol. 19, No. 11, 1976, pp. 624-633.

[2] Y. Tay, N. Goodman, R. Suri, "Locking Performance in Centralized Databases", ACM Transactions on Database Systems, Vol. 10, No. 4, 1985, pp. 415-462.

[3] P. A. Bernstein, N. Goodman, "Timestamp Based Algorithms for Concurrency Control in Distributed Database Systems", 6th International Conference on Very Large Data Bases, 1980, pp. 285-300.

[4] H. T. Kung, J. T. Robinson, "On Optimistic Methods for Concurrency Control", ACM TODS, Vol. 6, No. 2, 1981, pp. 213-226.

[5] R. Abbott, H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation", 14th International Conference on Very Large Data Bases, 1988, pp. 1-12.

[6] L. Sha, R. Rajkumar, J. Lehoczky, "Concurrency Control for Distributed Real-Time Databases", ACM SIGMOD Record, March 1988, pp. 82-98.

[7] L. Sha, R. Rajkumar, J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", IEEE Transactions on Computers, Vol. 39, No. 9, 1990, pp. 1175-1185.

[8] L. Sha, R. Rajkumar, S. H. Son, C. H. Chang, "A Real-Time Locking Protocol", IEEE Transactions on Computers, Vol. 40, No. 7, 1991, pp. 793-800.

[9] J. Huang, J. A. Stankovic, D. Towsley, K. Ramamritham, "Experimental Evaluation of Real-Time Transaction Processing", 10th Real-Time Systems Symposium, 1989, pp. 144-153.

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

[11] J. Huang, J. A. Stankovic, K. Ramamritham, D.

Towsley, "On Using Priority Inheritance in Real-Time Database," 12th Real-Time System Symposium, 1991, pp. 210-221.

[12] R. Abbott, H. Garcia-Molina, "Scheduling I/O Requests with Deadlines: A Performance Evaluation," 11th Real-Time Symposium, 1990, pp. 113-124.

윤 석 환

1982년 2월 아주대학교 산업공학과(공학사)
1984년 2월 건국대학교 산업공학과(공학석사)
1992년 3월~현재 아주대학교 산업공학과 박사과정
1992년 8월 품질관리 기술사 자격 취득
1986년~현재 한국전자통신연구소 책임연구원
관심분야: 그룹웨어, S/W 공학, 생산정보시스템

이 재 영

1988년 2월 서울대학교 물리학과(이학사)
1990년 5월 The Johns Hopkins Univ.(이학석사)
1994년 8월 The Johns Hopkins Univ.(이학박사)
1994년 12월~현재 한국전자통신연구소 시각언어연구실
관심분야: 그룹웨어, 분산처리, S/W 공학

박 치 항

1974년 2월 서울대학교 응용물리학과(이학사)
1980년 2월 한국과학원 전산학과(이학석사)
1987년 12월 파리 6대학 전산학과(공학박사)
1974년 2월~1978년 2월 한국과학기술연구소 연구원
1978년 2월~1985년 7월 한국전자기술연구소 선임연구원
1985년 7월~현재 한국전자통신연구소 책임연구원 멀티미디어연구부장
관심분야: 멀티미디어, 분산시스템, 그룹웨어, 네트워크 컴퓨팅, 에이전트 아키텍쳐, 가상현실