산업기술연구(강원대학교 산업기술연구소 논문집), 제18집, 1998.
Research Report. The Institute of Industrial Technology, Kangwon Nat'l Univ., Korea, No. 18, 1998.

交錯 狀態의 檢出을 위한 락 待機 情報의 相互 排除

Mutual Exclusion of Lock Waiting Information for Deadlock Detection

김 상 욱* 염 상 민** 장 연 정*** 김 윤 호****
Kim, Sang-Wook Yeom, Sang-Min Jang, Yoen-Jung Kim, Yoon-Ho

ABSTRACT

The deadlock detector is a DBMS sub-component that examines periodically whether a system is in a deadlock state based on lock waiting information of transactions. The deadlock detector and transactions execute concurrently in a DBMS and read and/or write the lock waiting information simultaneously. Since the lock waiting information is a shared one, we need an efficient method guaranteeing its physical consistency by using mutual exclusion. In this paper, we propose a new method that effectively guarantees physical consistency of lock waiting information. Two primary goals of our method is to minimize the processing overhead and to maximize system concurrency.

1. 서론

데이타베이스 관리 시스템(database management system: DBMS)에서는 다수의 트랜잭션(transaction)들이 동시에 수행된다[4][11]. DBMS는 여러 트랜잭션들에 의하여 동시에 액세스되는 데이타베이스의 논리적인 일관성(logical consistency)을 보장하기 위하여 동시성 제어(concurrency control) 기능을 제공한다. 현재까지 알려진 동시성 제어 기법들은 크게 이단계 라킹 규약(two-phase locking protocol)[7][11], 타임스탬프 순서화 기법(time-stamp ordering scheme)[5], 낙관적 기법(optimistic method)[13], 다중 버전 기법(multiple version method)[16] 등으로 분류될 수 있다. 이단계 라킹 규약은 데이타를 액세스하기 전에 반드시 락을 먼저 획득하도록 함으로써 동시성을 제어하는 방법이며, 그 실용성으로 인하여 현재 가장 많은 시스템에서 널리 사용되고 있다[2][4][10][11].

* 강원대학교 정보통신공학과 교수
** 강원대학교 정보통신공학과 대학원 석사과정
*** 강원대학교 정보통신공학과 재학생
**** 강원대학교 정보통신공학과 재학생

본 연구는 강원대학교 멀티미디어연구센터를 통한 정보통신부 정보통신 우수대학원 지원사업과 강원대학교 소프트웨어 특성화 사업의 일환인 98년도 우수 연구그룹 지원사업(그룹명: 데이터 및 지식공학 연구회)의 부분적인 지원에 의한 결과임.

이단계 라킹 기법에서는 두 개 이상의 트랜잭션들이 락을 획득한 상태에서 다른 트랜잭션이 가진 락을 무한정으로 상호 대기하는 교착 상태(deadlock)가 발생할 수 있다[1][11][12]. 따라서 DBMS는 트랜잭션들이 교착 상태에 빠져 있는가를 점검하고, 교착 상태에 빠져 있는 경우 이와 연관된 트랜잭션들을 찾아냄으로써 교착 상태 문제를 해결해 주어야 한다. 교착 상태 검출기(deadlock detector)는 이와 같이 트랜잭션들이 교착 상태에 빠져 있는가를 주기적으로 검출해 주는 DBMS의 서브 컴포넌트이다.

이단계 라킹 규약을 사용하는 경우, 하나의 트랜잭션이 락을 대기한다는 것은 그 락을 이미 획득한 다른 트랜잭션의 종료를 기다린다는 의미가 된다. 락 대기 정보(lock waiting information)는 이와 같이 락과 연관된 트랜잭션들간의 상호 대기 관계를 나타내는 정보이다. 락 대기 정보는 락 획득을 요청한 트랜잭션이 대기하게 되는 경우와 락을 대기하던 트랜잭션이 그 락을 획득하게 되는 경우에 갱신된다. 또한, 주기적으로 수행되는 교착 상태 검출기는 이러한 락 대기 정보를 기반으로 대기 그래프(wait-for-graph)를 형성함으로써 시스템의 교착 상태 여부를 결정하게 된다[1][12]. DBMS내에서는 트랜잭션들과 교착 상태 검출기의 수행이 병행되며, 이들은 각각의 수행을 위

하여 락 대기 정보를 공통적으로 읽고 쓰게 된다. 따라서 공유 정보에 해당되는 락 대기 정보의 물리적 일관성(physical consistency)을 보장하기 위해서는 이에 대한 효과적인 상호 배제(mutual exclusion) 기법이 요구된다.

간단히 고려할 수 있는 방법은 래치(latch) [15][14]를 이용하여 공유 정보에 대한 상호 배제를 구현하는 것이다[2]. 이것은 각 트랜잭션이 락 대기 정보를 갱신하는 경우나 교착 상태 검출기가 락 대기 정보를 참조하는 경우 락 대기 정보 전체에 래치를 걸고, 해당 갱신 혹은 참조 작업이 종료되는 경우 이를 다시 풀어주는 방법이다. 이를 통하여 각 트랜잭션 혹은 교착 상태 검출기는 상호 배타적인 액세스를 할 수 있으며, 이 결과 락 대기 정보의 물리적인 일관성이 보장된다.

그러나 이와 같은 방법을 사용하는 경우, 각 트랜잭션 혹은 교착 상태 검출기가 락 대기 정보 전체에 대하여 상호 배타적인 액세스를 하는 동안 다른 트랜잭션들은 락 대기 정보를 전혀 액세스할 수 없다. 따라서 시스템내의 동시성(concurrency)이 크게 저하되며, 이 결과 각 트랜잭션의 응답 시간이 떨어진다. 또한, 각 트랜잭션이 락 대기 정보를 참조할 때마다 래치를 걸어 주어야 하므로 이에 대한 처리 오버헤드가 크다 [3][6][8][9].

본 연구에서는 락 대기 정보의 물리적 일관성을 효과적으로 보장하는 새로운 기법을 제안한다. 제안하는 기법은 상호 배제를 위한 오버헤드를 극소화하고, 교착 상태 검출기와 트랜잭션들 간의 불필요한 상호 간섭을 최소화함으로써 시스템의 동시성을 극대화하는 것을 주요 목표로 한다. 제안하는 기법은 현재 ETRI에서 개발하고 있는 실시간 주기억장치 DBMS를 위한 동시성 제어 관리자의 일부로서 사용 중이다.

본 논문의 구성은 다음과 같다. 제 2장에서는 관련된 연구 배경으로서 트랜잭션 테이블에서 락 대기 정보를 관리하는 방법과 이를 이용하여 교착 상태를 검출하는 과정에 관하여 간략히 설명한다. 제 3장에서는 본 연구에서 락 대기 정보를 위한 효과적인 상호 배제의 필요성을 제시하고, 상호 배제 방안이 갖추어야 할 요건을 정의한다. 제 4장에서는 제안하는 락 대기 정보의 상호 배제 기법에 관한 기본 아이디어, 알고리즘, 장단점에 관하여 상세히 논의하고, 제 5장에서는 결론을 내린다.

II. 연구 배경

본 장에서는 연구 배경 지식으로서 락 대기 정보의 관리와 교착 상태 검출에 관한 기존의 방안에 관하여 간략히 소개한다.

2.1. 락 대기 정보의 관리

트랜잭션 T_i가 모드 $mode_i$로 획득하고자 하는 락 L을 다른 트랜잭션 T_j가 이미 모드 $mode_i$와 충돌(conflict)되는 모드 $mode_j$로 획득한 경우, 트랜잭션 T_i는 더 이상 수행을 계속하지 않고 트랜잭션 T_j가 락 L을 반환할 것을 대기하게 된다. 이단계 라킹 규약에서 모든 트랜잭션들은 획득하였던 락들을 종료 시점까지 반환하지 않으므로 트랜잭션 T_i가 락 L을 대기한다는 것은 이미 이를 획득하고 있는 트랜잭션 T_j의 종료를 대기한다는 의미가 된다. 락 대기 정보는 락과 연관하여 각각의 트랜잭션이 어떤 트랜잭션의 종료를 대기하고 있는가를 표현하는 정보이다.

그림 2.1은 트랜잭션 테이블(transaction table)에서 관리되는 락 대기 정보를 나타낸다[11]. 트랜잭션 테이블은 트랜잭션 엔트리들의 집합이며, 각 트랜잭션 엔트리는 그와 대응되는 트랜잭션에 관한 정보를 유지한다. 여기서는 이러한 정보들 중에서 본 논문의 주제와 연관된 정보만을 기술한다. TransID는 해당 트랜잭션의 식별자를 의미한다. Latch는 여러 트랜잭션들이 공유하는 자원에 대한 물리적인 일관성(physical consistency) 보장을 위하여 래치를 거는 수단으로 사용된다. LockWaiting은 이 트랜잭션이 대기하는 다른 트랜잭션의 TransID를 가지는 락 대기 정보 요소들의 리스트를 가리키는 포인터를 의미한다.

한 리스트 내에 여러 개의 락 대기 정보 요소가 존재하는 이유는 이미 여러 개의 트랜잭션들이 읽기 모드로 획득한 락을 다른 트랜잭션이 쓰기 모드로 요청하는 경우가 발생하기 때문이다. 예를 들어, 트랜잭션 T_1이 락 L_1을 쓰기 모드로 획득하려고 할 때, 이미 트랜잭션 T_4, T_6, T_8이 이 락 L_1을 읽기 모드로 획득한 상태이면, 그림 2.1에서와 같이 T_1을 위한 트랜잭션 엔트리의 lockWaiting은 세 개의 락 대기 정보 요소 T_4, T_6, T_8로 구성되는 리스트를 가리키게 된다.

2) 락 대기 정보의 상호 배제 방안에 관하여 다루고 있는 구체적인 참고 문헌은 발견할 수 없었다. 이는 특별한 성능상의 고려 사항을 염두에 두지 않고 여기서 기술하고 있는 단순한 방법을 사용하기 때문으로 사료된다.

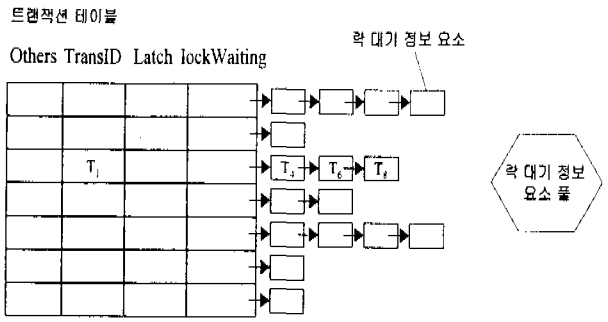

그림 2.1. 트랜잭션 테이블과 락 대기 정보.

일반적으로 이러한 락 대기 정보 요소들은 미 사용중인 요소들의 집합으로 구성되는 풀(pool) 내에서 동적으로 관리된다. 트랜잭션이 락을 대기하게 되면 하나의 락 대기 정보 요소가 이 풀로부터 할당되어 lockWaiting 리스트에서 사용되고, 대기하던 트랜잭션이 락을 획득하게 되면 리스트 내의 모든 락 대기 정보 요소들은 풀로 반환된다. 이와 같이, 락 대기 정보는 락 획득을 요청한 트랜잭션이 대기하는 경우와 대기하던 트랜잭션이 해당 락을 획득하는 경우 변경된다.

2.2. 교착 상태의 검출

교착 상태(deadlock)란 두 개 이상의 트랜잭션들이 락을 획득한 상태에서 서로 다른 트랜잭션이 획득한 락이 반환되기를 무한정 기다리는 현상이다[1][11][12]. 교착 상태는 이단계 라킹 규약을 이용한 동시성 제어 기법에서 흔히 발생하며, 이 결과 시스템의 성능을 급격히 떨어뜨리는 심각한 문제를 야기 시킬 수 있다. 따라서 DBMS는 트랜잭션들이 교착 상태에 빠져 있는가를 점검하고, 교착 상태에 빠져 있는 경우 이와 연관된 트랜잭션들을 찾아냄으로써 교착 상태에서 벗어나도록 해 주어야 한다. 교착 상태 검출기(deadlock detector)는 이와 같이 트랜잭션들이 교착 상태에 빠져 있는가를 주기적으로 검출해 주는 DBMS의 서브 컴포넌트이다.

교착 상태를 검출하는 가장 단순한 방법의 하나는 한 트랜잭션이 너무 오랜 시간 동안 락을 대기하는 경우 이를 교착 상태로 간주하는 타임 아웃 방식(time-out strategy)이다. 이 방식은 교착 상태 여부를 단순히 추측을 통하여 결정하므로 옳지 않은 판단을 할 수 있으며, 이 결과 전체 시스템의 성능 저하를 초래할 수 있다.

이러한 타임 아웃 방식의 문제점을 해결하기 위하여 제안된 것이 대기 그래프 방식(wait-for-graph strategy)이다[1][12]. 대기 그래프 방식은 트랜잭션들의 상호 대기 관계를 표현하는 대기 그래프(wait-for-graph)를 구성하고, 이 대기 그래프 상의 싸이클(cycle) 존재 여부를 파악함으로써 시스템의 상태를 파악한다. 즉, 대기 그래프 상에 싸이클이 존재하면, 그 싸이클을 형성하는 트랜잭션들이 교착 상태를 이루는 것이다. 이 방식은 대기 그래프 형성과 싸이클 점검을 위한 추가의 오버헤드가 있으나 교착 상태를 정확하게 검출할 수 있다는 장점으로 인하여 실제 DBMS에서 널리 사용되고 있다.

교착 상태 검출기는 이와 같은 대기 그래프 방식으로 시스템의 교착 상태 여부를 검출해 주는 DBMS의 서브 컴포넌트이다. 교착 상태 검출기는 주기적으로 수행되며, 그림 2.1과 같이 유지되는 락 대기 정보를 참조하여 대기 그래프를 형성하고, 이를 기반으로 교착 상태를 검출하는 역할을 한다. 교착 상태 검출기의 수행 주기는 시스템의 부하에 따라 적절하게 조정된다.

III. 연구 동기

본 장에서는 이 연구의 동기로서 해결하고자 하는 문제와 이 문제에 대한 해결 방안이 추구하는 목표를 정의한다.

3.1. 문제 정의

제 2장에서 언급한 바와 같이 트랜잭션은 락을 대기하는 경우와 대기 후 락을 획득하는 경우 락 대기 정보 요소를 갱신하게 되고, 주기적으로 수행되는 교착 상태 검출기는 이 락 대기 정보를 참조하여 교착 상태 여부를 결정하게 된다. 따라서 이 락 대기 정보는 병행 수행되는 교착 상태 검출기와 트랜잭션들이 동시에 읽고 쓰는 공유 정보이다.

이러한 공유 정보를 별도의 제어 없이 다수 프로세스들의 액세스를 허용한다면, 잘못된 수행이 발생 가능하다. 예를 들어, 교착 상태 검출기가 락 대기 정보를 액세스하는 과정에서 그림 2.1의 세 번째 엔트리의 두 번째 요소를 참조하는 중 문맥 교환(context switch)이 발생하게 되는 경우를 가정하자. 이때, 트랜잭션 T1이 락 대기를 끝낸 후 수행을 재개하는 경우, lockWaiting이 가리키는 리스트상의 요소를 요소 풀로 반환하게 된다. 이후에 교착 상태 검출기가 수

행을 재개하게 되면, 이미 풀에 반환되어 사용되지 않은 두 번째 요소가 가리키는 많은 요소들을 차례로 액세스하게 되어 올바르지 못한 결과를 초래하게 된다.

따라서 공유 정보인 락 대기 정보의 물리적인 일관성(physical consistency)을 보장하기 위한 상호 배제(mutual exclusion) 기능이 필요하다.

3.2. 해결 방안의 요건

락 대기 정보의 상호 배제를 위한 가장 단순한 방법의 하나는 락 대기 정보에 하나의 래치를 할당하여 놓은 후, 락 대기 정보를 액세스할 때마다 래치(latch)[3][15]를 획득하도록 하는 것이다. 락 대기 정보가 액세스되는 경우는 (1) 교착 상태 검출기가 대기 그래프를 형성하기 위하여 전체 락 대기 정보를 읽는 경우, (2) 트랜잭션이 다른 트랜잭션의 종료를 대기하게 되어 자신의 락 대기 정보를 갱신하는 경우, (3) 대기하던 트랜잭션이 락을 획득하게 되어 자신의 락 대기 정보를 갱신하는 경우 등이 있다. 따라서 이러한 세 가지 종류의 작업이 수행되기 직전 래치가 획득하고, 작업이 완료된 후 래치를 반환으로써 물리적인 일관성을 보장할 수 있다.

그러나 이와 같이 단순한 방식은 다음과 같은 두 가지 문제점을 유발시킨다.

첫째, 래치 처리를 위한 오버헤드의 문제이다. 각 트랜잭션은 락을 대기하거나 대기 후 락을 획득할 때마다 락 대기 정보의 갱신을 요구하므로 모든 트랜잭션은 이러한 경우 매번 래치를 획득해야 한다. 이러한 락 대기 및 락 대기 후 획득은 하나의 트랜잭션의 수행에서도 매우 빈번하게 발생하게 된다. 래치를 처리하는 비용은 락을 처리하는 비용과 비교하여 작으나, 이와 같은 빈번한 래치의 처리는 전체 트랜잭션의 수행 성능을 저하시킨다.

둘째, 동시성(concurrency)의 저하 문제이다. 교착 상태 검출기는 래치를 획득한 상태에서 락 대기 정보를 기반으로 교착 상태를 검출하게 되므로, 교착 상태 검출기가 수행되는 동안 다른 트랜잭션들은 락 대기 정보를 전혀 액세스할 수 없게 된다. 최악의 경우, 교착 상태 검출기가 수행될 때, 전체 트랜잭션들이 모두 수행을 정지할 가능성도 존재한다. 따라서 시스템의 동시성은 크게 저하되며, 이 결과 트랜잭션 응답 시간이 떨어진다.

본 연구에서 제안하고자 하는 락 대기 정보의 상호 배제 기법은 래치 처리 횟수의 극소화와 시스템 동시성의 극대화하는 것을 궁극적인 목표로 한다.

IV. 제안된 기법

본 장에서는 제안하는 락 대기 정보의 상호 배제 기법에 관하여 설명한다. 먼저, 제 4.1절에서는 제안하는 기본 아이디어를 제시하고, 제 4.2절에서는 동작을 위한 구체적인 알고리즘을 기술한다. 제 4.3절에서는 제안하는 기법의 장단점에 관하여 논의한다.

4.1. 기본 아이디어

그림 2.1에 나타난 락 대기 정보 요소 풀은 모든 트랜잭션들이 공유하는 정보이므로 이의 상호 배제를 위하여 래치가 존재한다. 즉, 각 트랜잭션이 자신의 락 대기 정보에 새로운 요소를 넣거나 존재하던 요소를 풀에 반환하는 경우에는 반드시 이 래치를 획득해야만 한다. 제안하는 기법에서는 락 대기 정보 요소 풀에 할당되어 있는 이 래치를 트랜잭션 테이블내의 락 대기 정보를 위한 상호 배제 수단으로서 사용한다. 이것은 락 대기 정보를 위한 상호 배제를 목적으로 하는 별도의 래치 처리 오버헤드를 유발시키지 않도록 하기 위한 것이다.

트랜잭션이 락 대기 정보를 갱신하고자 하는 경우에는 이미 락 대기 정보 요소를 할당받기 위하여 풀에 래치를 걸고 있을 것이므로 락 대기 정보의 상호 배제를 위한 별도의 래치 처리 작업은 불필요하다. 한편, 교착 상태 검출기가 락 대기 정보를 참조하기 위해서는 풀에 할당된 이 래치를 획득하도록 한다. 교착 상태 검출기의 수행 횟수는 트랜잭션 수행 횟수와 비교하여 매우 적으므로 이러한 오버헤드는 상대적으로 미미하다. 교착 상태 검출기가 이 래치를 사용할 수 있도록 하기 위하여 교착 상태 검출기와 대

---

3) 락과 비교한 래치의 특성은 다음과 같다[15]. 첫째, 락이 사용자 정보의 물리적 일관성을 유지하기 위하여 사용되는 반면, 래치는 시스템 정보의 물리적인 일관성을 유지하려는 목적으로 사용된다. 둘째, 락이 이단계 라킹 규약에 의거하여 트랜잭션 종료 시점까지 반환될 수 없는 것과는 달리 래치는 공유 정보에 대한 액세스 후에는 즉시 반환된다. 셋째, 래치를 거는 비용은 락을 거는 비용과 비교하여 약 10% 내외로서 상대적으로 작다. 넷째, 래치는 교착 상태 검출기가 수행될 때 검출되는 대상이 아니다. 따라서, 래치를 획득하기 전에 래치와 래치간의 교착 상태나 래치와 락과의 교착 상태 여부를 미리 점검한 후 획득해야 한다. DBMS 개발자들은 하나의 트랜잭션이 동시에 획득할 수 있는 래치의 수를 최대 2개에서 3개 이하로 제한하고 있다.

응되는 엔트리를 트랜잭션 테이블 내에 할당한다.

트랜잭션 테이블의 각 엔트리가 가리키는 락 대기 정보 리스트에 미리 정해진 numStaticElements 만큼의 락 대기 정보 요소를 시스템 초기화 시에 미리 할당시킨다. 이는 트랜잭션이 래치를 획득하는 수를 줄임으로써 동시성을 극대화하기 위한 것이다. 즉, 트랜잭션이 다른 트랜잭션들의 종료를 대기하게 될 때, 자신이 대기하는 트랜잭션들의 수가 numStaticElements 미만인 경우에는 풀로부터 락 대기 정보 요소를 할당받을 필요가 없으며, 자신이 대기하는 다른 트랜잭션의 수가 numStaticElements를 초과하는 경우에 한하여 락 대기 정보 요소를 추가적으로 할당받게 된다. 따라서 이러한 락 대기 정보 요소의 사전 할당 방식을 사용함으로써 교착 상태 검출기와 트랜잭션들간의 래치 경쟁을 극소화 할 수 있으며, 이 결과 동시성은 극대화된다[4].

1. numStaticElements를 지정한다.
2. 트랜잭션 테이블 내에 교착 상태 검출기를 위한 하나의 엔트리를 할당하고, 엔트리의 각 필드를 초기화한다.
3. 트랜잭션 테이블의 나머지 각 엔트리에 대하여
   3.1 numStaticElements 만큼의 락 대기 정보 요소를 풀로부터 할당하여 리스트를구성시킨다.
   3.2 각 요소의 값을 NULL로 초기화 시킨다.
   3.3 엔트리의 lockWaiting 필드가 이 리스트를 가리키도록 한다.

알고리즘 4.1. 시스템 초기화 과정.

4.2. 동작 알고리즘

본 절에서는 제안하는 기법의 수행을 위하여 필요한 알고리즘을 상세히 기술한다.

알고리즘 4.1은 시스템 초기화 과정을 나타낸 것이다. 먼저, 시스템의 특성을 고려하여 numStaticElements를 지정한다. 트랜잭션 테이블 내에 교착 상태 검출기를 위한 엔트리를 하나 생성하고, 각 필드를 초기화한다. 트랜잭션 테이블내의 모든 엔트리에 대하여, numStaticElements 만큼의 락 대기 정보 요소를 풀로부터 할당하여 리스트를 구성하고, 각 요소의 값을 NULL로 초기화한 후, 엔트리의 lockWaiting 필드가 이 리스트를 가리키도록 한다.

알고리즘 4.2는 트랜잭션의 락 대기 처리 방법을 나타낸 것이다. 이러한 락 대기의 처리는 트랜잭션이 락을 요청하였을 때 그 락이 다른 트랜잭션에 의하여 이미 획득된 상태인 경우와 락 대기 도중 이 락을 이 트랜잭션 보다 앞서 요청하여 대기 중이던 다른 트랜잭션이 먼저 획득한 경우에 필요하다. 먼저, 해당 락

1. 대기하고 있는 락을 앞서 획득한 트랜잭션의 수를 체크한다.
2. IF (해당 락을 획득한 트랜잭션 수가 현재 lockWaiting이 가리키는 리스트내의 요소 수를 초과한 경우)
   2.1 락 대기 정보 요소 풀에 래치를 건다.
   2.2 초과되는 만큼의 요소를 락 대기 정보 요소 풀로부터 할당받는다.
   2.3 할당 받은 요소들을 트랜잭션 테이블내 해당 엔트리의 lockWaiting 필드가 가리키는 리스트에 연결시킨다.
   2.4 해당 락을 획득한 트랜잭션들의 TransID를 lockWaiting 필드가 가리키는 리스트내의 각 요소에 넣는다.
   2.5 대기 정보 요소 풀에 걸었던 래치를 푼다.
3. ELSE
   3.1 해당 락을 획득한 트랜잭션들의 TransID를 lockWaiting 필드가 가리키는 리스트내의 각 요소에 넣는다.

알고리즘 4.2. 트랜잭션의 락 대기 처리 과정.

을 앞서 획득한 트랜잭션의 수를 체크한다. 만약, 해당 락을 획득한 트랜잭션 수가 현재 lockWaiting이 가리키는 리스트내의 요소 수를 초과하면, 락 대기 정

4) 물론, 교착 상태 검출기와 트랜잭션들이 미리 할당되어 있는 락 대기 정보 요소 값을 함께 액세스하는 경우가 발생한다. 그러나 락 정보 요소 리스트의 형태 변화 없이 원자적인 요소 값만을 읽고 쓰는 이러한 경우에는 상호 배제가 불필요하다.

보 요소 풀에 래치를 걸고 초과되는 만큼의 요소를 락 대기 정보 요소 풀로부터 할당시킨다. 그리고 할당받은 요소들을 트랜잭션 테이블의 해당 엔트리의 lockWaiting 필드가 가리키는 리스트에 연결하고, 해당 락을 획득한 트랜잭션들의 TransID를 lockWaiting 필드가 가리키는 리스트내의 각 요소에 넣는다. 끝으로 락 대기 정보 요소 풀에 걸었던 래치를 풀어준다. 해당 락을 획득한 트랜잭션 수가 현재 lockWaiting이 가리키는 리스트내의 요소 수를 초과하지 않는 경우에는, 해당 락을 획득한 트랜잭션들의 TransID를 lockWaiting 필드가 가리키는 리스트내의 각 요소에 넣는다.

1 대기하는 트랜잭션 엔트리의 lockWaiting이 가리키는 리스트에서 종료된 트랜잭션의 TransID를 찾아 값을 NULL로 리셋시킨다.

알고리즘 4.3. 트랜잭션의 락 획득 처리 과정.

알고리즘 4.3은 트랜잭션의 락 획득 처리 과정을 나타낸 것이다. 만약, 락을 획득한 트랜잭션 엔트리의 lockWaiting 필드가 가리키는 리스트의 락 대기 정보 요소 수가 numStaticElements를 초과하면, 먼저 락 대기 정보 요소 풀에 래치를 건다. 그런 후, 초과되는 만큼의 요소들을 lockWaiting 필드가 가리키는 리스트로부터 떼어내어 락 대기 정보 요소 풀에 반환시키고, 대기 정보 요소 풀에 걸었던 래치를 풀어준다. 또한, lockWaiting 필드가 가리키는 리스트내의 모든 요소 값을 NULL로 리셋 시킨다. 해당 트랜잭션 엔트리의 lockWaiting 필드가 가리키는 리스트내의 요소 수가 numStaticElements를 초과하지 않는 경우에는, lockWaiting 필드가 가리키는 리스트내의 모든 요소 값을 NULL로 리셋 시킨다.

알고리즘 4.4는 트랜잭션의 락 반환 처리 과정을 나타낸 것이다. 락을 반환하는 트랜잭션을 대기하는 트랜잭션 엔트리의 lockWaiting 필드가 가리키는 리스트에서 종료된 트랜잭션의 TransID를 찾아 값을 NULL로 리셋 시킨다. 유의할 것은 이 요소를 풀에 즉시 반환하지 않는다는 것이다. 즉, 풀로의 반환은 이 트랜잭션이 락을 획득할 수 있을 때까지 지연함으로써 락 대기 정보 요소 풀에 대한 래치를 걸지 않도록 하기 위한 것이다. 이 결과, 잦은 래치 처리의 오버헤드를 제거할 수 있으며, 동시성도 증가시킬 수 있다.

알고리즘 4.5는 교착 상태 검출기의 처리 과정을 나타낸 것이다. 먼저, 락 대기 정보 요소 풀에 래치를 걸고, 트랜잭션 테이블내 각 엔트리의 lockWaiting

1. IF (해당 트랜잭션 엔트리의 lockWaiting 필드가 가리키는 리스트내의 요소 수가 numStaticElements를 초과하는 경우)
   1.1 락 대기 정보 요소 풀에 래치를 건다.
   1.2 초과하는 요소들을 lockWaiting 필드가 가리키는 리스트에서 떼어낸다.
   1.3 떼어낸 요소들을 락 대기 정보 요소 풀에 반환시킨다.
   1.4 대기 정보 요소 풀에 걸었던 래치를 푼다.
2. lockWaiting 필드가 가리키는 리스트내의 모든 요소 값을 NULL로 리셋 시킨다.

알고리즘 4.4. 트랜잭션의 락 반환 처리 과정.

필드가 가리키는 락 대기 정보를 자신의 로컬 메모리에 복제한다. 락 대기 정보 요소 풀에 걸었던 래치를 풀어 준 후, 로컬 메모리에 복제해 놓은 락 대기 정보를 이용하여 시스템의 교착 상태 여부를 검출한다. 여기서 로컬 메모리의 복제본을 이용하여 교착 상태를 검출하는 이유는 락 대기 정보 풀에 걸어놓은 래치를 잡고 있는 시간의 최소화를 위해서이다. 즉, 락 대기 정보의 복제하는 시간이 교착 상태를 검출하는 시간보다 훨씬 짧으므로 우선 복제한 후 즉시 래치를 반환해 줌으로써 다른 트랜잭션이 이 래치를 바로 획득할 수 있다. 이 결과, 시스템의 동시성이 극대화 될 수 있다.

4.3. 논의 사항

제안하는 기법은 락 대기 정보에 대한 상호 배제를 위하여 락 대기 정보 요소 풀에 할당된 래치를 이용한다. 이와 같이, 별도의 래치를 사용하지 않으므로 래치 처리를 위한 오버헤드를 요구하지 않는다. 래치를 처리하는 비용은 락을 처리하는 비용에 비하

여 작으나, 고속의 데이타 처리가 가능한 시스템에서는 이러한 래치 처리를 위한 오버헤드가 큰 비중을 차지한다. 제안하는 기법은 래치의 수를 최소화함으로써 트랜잭션 수행에 필요한 래치 획득을 줄일 수 있으며, 이 결과 빠른 응답 시간을 기대할 수 있다.

또한, 제안하는 기법은 각 트랜잭션을 위한 락 대기 정보 요소들을 주어진 만큼 미리 할당함으로써 각 트랜잭션이 수행 시 획득해야 하는 래치의 수를 최소화하는 방안을 사용한다. 이 결과, 각 트랜잭션은 자신이 대기하는 다른 트랜잭션의 수가 미리 할당된 요소 수를 초과하는 경우에만 래치를 획득하게 된다. 따라서 교착 상태 검출기와 트랜잭션들간의 래치 경쟁을 극소화 할 수 있으며, 이 결과 시스템의 동시성은 극대화된다. 특히, 응용 시스템의 특성에 따라 사전에 할당되는 락 대기 정보 요소의 수를 조정함으로써 대부분의 트랜잭션들은 락 대기 정보 요소의 동적

1. 락 대기 정보 요소 풀에 래치를 건다.
2. 트랜잭션 테이블내 각 엔트리의 lockWaiting 필드가 가리키는 락 대기 정보를 자신의 로컬 메모리에 복제한다.
3. 락 대기 정보 요소 풀에 걸었던 래치를 푼다.
4. 단계 2에서 복제했던 락 대기 정보를 이용하여 교착 상태에 있는 트랜잭션들을 출해 낸다.

알고리즘 4.5. 교착 상태 검출기의 처리 과정.

인 할당 없이 정적인 할당만으로 락 대기 정보를 구성할 수 있다. 이러한 트랜잭션들은 교착 상태 검출기의 수행 여부에 전혀 영향을 받지 않고 독립적으로 수행할 수 있도록 해 준다. 이러한 동시성의 극대화는 DBMS의 성능 향상에 일조할 수 있다.

V. 결론

이단계 라킹 규약은 다수의 트랜잭션들의 수행이 병행되는 데이타베이스 환경에서 데이타의 논리적 일관성을 보장하기 위하여 널리 사용되는 동시성 제어 기법이다. 이단계 라킹 규약을 사용하는 시스템에서는 두 개 이상의 트랜잭션들이 락을 잡은 상태에서 서로 다른 트랜잭션이 가진 락을 무한정으로 상호 대기하는 교착 상태가 발생할 수 있다.

교착 상태 검출기는 트랜잭션들 간의 상호 대기 관계를 나타내는 락 대기 정보를 기반으로 시스템 내에 교착 상태가 발생하였는가를 주기적으로 검출하고 해결하는 DBMS의 서브 컴포넌트이다. 트랜잭션들과 교착 상태 검출기의 수행이 병행되며, 이들은 각각의 수행을 위하여 락 대기 정보를 동시에 읽고 쓰게 된다. 따라서 이러한 락 대기 정보의 물리적 일관성을 보장하기 위한 상호 배제 기법이 요구된다.

본 연구에서는 락 대기 정보의 물리적 일관성을 효과적으로 보장하는 새로운 기법을 제안한다. 제안하는 기법에서는 별도의 래치 처리 오버헤드를 유발시키지 않도록 하기 위하여 락 대기 정보 요소 풀을 위한 래치를 락 대기 정보를 위한 상호 배제 수단으로서 사용한다. 또한, 락 대기 정보 요소의 사전 할당 방식을 사용함으로써 교착 상태 검출기와 트랜잭션들간의 래치 경쟁을 극소화시킨다. 이 결과, 제안하는 기법은 처리 비용의 극소화와 동시성의 극대화라는 두 가지 요건을 모두 만족시킴으로써 고성능 시스템에 만족할 만한 좋은 성능을 제공할 수 있다.

참고 문헌

[1] R. Agrawal, M. Carey, and D. DeWitt, "Deadlock Detection is Cheap" *ACM SIGMOD Record*, Vol. 13, No. 2., 1983.

[2] R. Agrawal, M. Carey, and M. Livny, "Models for Studying Cuncurrency Control Performance: Alternatives and Implications," *In Proc. Intl. Conf. on Management of Data, ACM SIGMOD*, May., 1985.

[3] A. Ammann, M. Hanrahan, and R. Krishnamurthy, "Design of a Memory Resident DBMS," *In Proc. Intl. Conf. on COMPCON*, Feb., 1985.

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

[5] P. Bernstein, N. Goodman, "Timestamp-Based Algorithms for Concurrency Control in

Distributed Systems," *In Proc. Intl. Conf. on Very Large Data Bases, VLDB.,* 1980.

[6] D. DeWitt. et al, "Implementation Techniques for Main Memory Database Systems," *In Proc. Intl. Conf. on Management of Data, ACM SIGMOD,* 1984.

[7] K. P. Eswaran et al., "The Notion of Consistency and Predicate Locks in a Database System," *Comm. of the ACM*, Vol. 19, No. 11, Nov., 1976.

[8] H. Garchia-Molina and K. Salem, "High Performance Transaction Processing with Memory Resident Data," *In Proc. Intl. Workshop on High Performance Transaction Systems*, Dec., 1987.

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

[10] J. Gray, R. Lorie, and G. Putzolu, "Granularity of Locks in a Shared Data Base," *In Proc. Intl. Conf. on Very Large Data Bases*, Sept., 1975.

[11] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufman Publishers., 1993.

[12] B. Jiang, "Deadlock Detection is Really Cheap," *ACM SIGMOD Record*, Vol. 17, No. 2, pp. 2-13, June., 1988.

[13] H. Kung and J. Robinson, "On Optimistic Methods for Concurrency Control," *ACM Trans. on Database Systems*, Vol. 6, No. 2., 1981.

[14] C. Mohan and F. Levine, "ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging," *IBM Research Report RJ* 6846., 1989.

[15] C. Mohan et al., "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," *ACM Trans. on Database Systems,* Vol. 17, No. 1, Mar., 1992.

[16] C. Papadimitriou and P. Kanellakis, "On Concurrency Control by Multiple Versions," *ACM Trans. on Database Systems,* Vol. 9, No. 1, pp. 89-99., 1984.

[17] A. Silberschatz and P. B. Galvin, *Operating System Concepts,* Fourth Edition, Addison-Wesley Publishing Company., 1998.