데이타베이스 관리 시스템에서의 적응형 로크 상승

(Adaptive Lock Escalation in Database Management Systems)

장 지 웅 † 이 영 구 †† 황 규 영 ††† 양 재 헌 †††

(Ji-Woong Chang) (Young-Koo Lee) (Kyu-Young Whang) (Jae-Heon Yang)

요 약 데이타베이스 관리 시스템에서 한계 이상의 로크요청이 발생하는 경우에는 트랜잭션이 철회된다. 최악의 경우에는 트랜잭션들이 연속적으로 철회되어 시스템이 정지된 것과 같이 어느 트랜잭션도 완료하지 못하는 현상이 발생할 수 있다. 이 문제점을 해결하기 위하여 로크상승을 사용하지만 기존의 로크상승 방법들은 문제를 완전히 해결하지는 못한다.

본 논문에서는 이 문제를 해결하기 위하여 적응형 로크상승 기법을 제안한다. 먼저 로크상승에 대한 체계적인 모델과 로크자원의 부족으로인한 트랜잭션 철회의 주 발생 원인인 상승불가능 로크의 개념을 제안한다. 또한 상승불가능 로크의 수를 제어하기 위한 해결책으로 준로크상승, 로크블로킹, 선택적 강제수행의 개념을 제안하고, 이를 적응형 로크상승 기법에 적용한다. 적응형 로크상승 기법은 불필요한 트랜잭션 철회를 감소시키며 과다한 로크요청 상황에서 시스템의 성능을 단계적으로 저하시키면서 시스템이 정지되는 현상이 발생하지 않음을 보장한다.

적응형 로크상승 기법의 성능을 검증하기 위하여 시뮬레이션을 통한 실험을 수행하였다.실험결과 적응형 로크상승 기법은 기존의 로크상승 방법을 사용하는 경우에 비하여 트랜잭션의 철회와 평균 응답시간을 크게 줄이고, 단위시간당 트랜잭션 처리율을 향상시켰다. 특히 동시에 수행할 수 있는 트랜잭션의 수가 16에서 256배 이상 증가하는 것을 보였다.

본 논문은 모호하게 인식되던 로크자원 관리 측면에서의 로크상승의 역할을 체계적으로 규명하고 상세한 작동원리를 명확히 했다는 점에서 커다란 의의가 있다. 기존의 로크상승 방법들은 과다한 로크요청이 발생할 때의 문제를 사용자 또는 시스템관리자의 책임으로 처리한다. 반면에 적응형 로크상승 기법은 과다한 로크요청이 발생할 때의 문제를 자동적으로 조절하므로 사용자의 부담을 크게 감소시킨다. 따라서 최근에 많은 관심이 모아지고 있는 자체조율(self-tuning)이 가능한 데이타베이스 관리 시스템 개발에 공헌할 것이다.

Abstract Since database management systems(DBMSs) have limited lock resources, transactions requesting locks beyond the limit must be aborted. In the worst case, if such transactions are aborted repeatedly, the DBMS can become paralyzed, i.e., transactions execute but cannot commit. Lock escalation is considered a solution to this problem. However, existing lock escalation methods do not provide a complete solution.

In this paper, we propose a new lock escalation method, adaptive lock escalation, that solves most of the problems. First, we propose a general model for lock escalation and present the concept of the unescalatable lock, which is the major cause making the transactions to abort.

Second, we propose the notions of semi lock escalation, lock blocking, and selective relief as the mechanisms to control the number of unescalatable locks. We then propose the adaptive lock escalation

· 본 연구는 첨단정보기술연구센터를 통하여 한국과학재단의 지원을 받았음.

† 학생회원 : 한국과학기술원 전자전산학과
jwchang@mozart.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전자전산학과
yklee@mozart.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전자전산학과 교수
kywhang@cs.kaist.ac.kr
jhyang@cs.kaist.ac.kr

논문접수 : 2000년 6월 9일
심사완료 : 2001년 5월 23일

method using these notions. Adaptive lock escalation reduces needless aborts and guarantees that the DBMS is not paralyzed under excessive lock requests. It also allows graceful degradation of performance under those circumstances.

Third, through extensive simulation, we show that adaptive lock escalation outperforms existing lock escalation methods. The results show that, compared to the existing methods, adaptive lock escalation reduces the number of aborts and the average response time, and increases the throughput to a great extent. Especially, it is shown that the number of concurrent transactions can be increased more than 16~256 fold.

The contribution of this paper is significant in that it has formally analyzed the role of lock escalation in lock resource management and identified the detailed underlying mechanisms. Existing lock escalation methods rely on users or system administrator to handle the problems of excessive lock requests. In contrast, adaptive lock escalation releases the users of this responsibility by providing graceful degradation and preventing system paralysis through automatic control of unescalatable locks. Thus adaptive lock escalation can contribute to developing self-tuning DBMSs that draw a lot of attention these days.

1. 서 론

데이타베이스 관리 시스템에서 여러 트랜잭션(transaction)이 데이타베이스를 동시에 액세스하는 경우 일관성(consistency)이 결여된 데이타가 생성되지 않도록 트랜잭션간의 상호작용을 제어하는 작업을 동시성 제어라고 한다[1]. 동시성 제어 방법 중에서도 가장 널리 사용되는 방법이 로킹기법(locking)이다. 로킹기법은 데이타를 액세스하기 전에 로크를 획득함으로써 충돌을 발생시킬 수 있는 다른 트랜잭션의 수행을 사전에 방지한다.

대부분의 경우 로크 정보는 주기억장치 상에서 관리되므로 물리적인 제약에 의하여 시스템에서 제공할 수 있는 로크의 갯수에는 한계가 있다.[1] 일반적으로 데이타베이스 관리 시스템에서 제공할 수 있는 로크의 갯수는 시스템을 시작할 때 시스템 관리자에 의해 설정된다[2]. 따라서 운용 중에 동시에 트랜잭션들이 많은 수의 로크를 요청하거나 다수의 트랜잭션들이 수행되면 시스템에서 로크자원이 부족해질 수 있다.

로크자원이 부족하게 되면 로크자원을 할당받지 못한 트랜잭션은 철회된다. 본 논문에서는 이러한 현상을 로크자원 고갈현상이라고 부른다. 이러한 현상이 심해지면 트랜잭션이 철회와 재시작을 반복하면서 완료되지 못하는 순환 재시작(cyclic restart)[1][3]이 발생한다. 순환 재시작은 교착상태에 비하여 발견이 어려울 뿐만 아니라 시스템 자원을 낭비한다는 면에서 더욱 좋지 않은 현상이다[1]. 최악의 경우에는 모든 트랜잭션이 순환 재시작 상태가 되어 어느 트랜잭션도 완료되지 않는 상태가 발생할 수 있다. 본 논문에서는 이러한 상황을 활성정지(live halt)라고 부르기로 한다. 매우 많은 수의 로크자원이 있더라도 많은 트랜잭션이 동시에 수행되면 활성정지가 발생할 수 있다. 활성정지는 결과적으로 시스템이 정지한 것과 동일한 치명적인 문제이다.

이러한 문제를 해결하는 방법으로는 계층적 로킹기법[4]을 기반으로 한 로크상승(lock escalation)이 있다[1]. 계층적 로킹기법은 하나의 시스템에서 계층화된 여러 크기의 로크단위를 제공함으로써 트랜잭션이 사용할 로크단위를 선택할 수 있도록 하는 방법이다[4]. 로크단위란 데이타의 일관성 유지를 위하여 원자적(atomically)으로 로크가 획득되는 데이타의 집단을 의미한다[4]. 로크단위의 예로는 데이타베이스, 화일, 페이지, 레코드 등이 있다. 로크상승은 계층적 로킹기법에서 여러 개의 하위수준 로크를 반환하는 대신 그 하위수준 로크들을 포함하는 하나의 상위수준 로크를 획득하는 방법이다[1]. 그러므로 로크상승을 이용하여 사용하는 로크의 갯수를 줄일 수 있다. 로크상승은 로크자원이 고갈되는 경우 해결책으로 사용할 수 있으나 이로 인해 로크의 단위가 커져서 동시성이 저하되는 문제가 있으므로 불필요한 로크상승은 가급적 피하는 것이 좋다.

기존의 로크상승 기법들[2][5]은 수행 중인 트랜잭션들이 사용하는 로크자원의 총 사용량은 고려하지 않고 각 트랜잭션이 사용하는 로크자원의 갯수에 따라 로크상승 여부를 결정하는 일종의 지역적 접근법을 사용한다. 그러나 지역적 접근법은 로크자원의 총 사용량을 고려하지 않기 때문에 다음과 같은 세가지 문제가 발생한다.

1) 본 논문에서는 트랜잭션에 의하여 획득되지 않고 로크 풀(lock pool)에 존재하는 로크를 트랜잭션이 획득한 로크와 구분하기 위하여 로크자원이라는 용어를 사용한다.

첫째, 사용되지 않는 로크자원이 충분히 남아있는 경우 로크상승을 수행할 필요가 없음에도 불구하고 불필요하게 로크상승을 수행하여 동시성을 저하시킬 수 있다. 둘째, 총 로크사용량이 한계값에 도달하여 로크상승이 꼭 필요한 경우에도 적은 수의 로크만을 획득한 트랜잭션은 로크상승을 수행하지 않으므로 불필요하게 트랜잭션이 철회된다. 셋째, 로크자원의 부족으로 인하여 트랜잭션들이 연속적으로 철회되어 활성정지가 발생할 수 있다.

이러한 문제점을 해결하기 위하여 시스템 내에서 사용되고 있는 로크의 총 갯수에 따라 로크상승의 수행여부를 결정하는 전역적 접근법을 생각할 수 있다. 본 논문에서는 이 방법을 기본 적응형 로크상승 기법(simple adaptive lock escalation) 이라고 부른다. 이 방법은 로크자원의 총 사용량이 일정한 수준에 도달할 때까지는 로크상승을 수행하지 않으므로 불필요한 로크상승을 줄일 수 있고, 각 트랜잭션이 획득한 로크의 갯수와 관계없이 로크상승을 수행하므로 로크상승이 필요한데도 수행하지 않는 문제점이 발생하지 않는다. 그러나 로크자원의 부족으로 인한 활성정지를 방지하지는 못한다. 왜냐하면, 많은 트랜잭션들이 로크를 공유하고 있을 때에는 로크모드의 충돌로 인하여 로크상승을 할 수 없는 경우가 있어서 이 때에는 트랜잭션이 철회할 수 밖에 없기 때문이다.

기존의 상용 데이타베이스 관리 시스템에서는 이러한 문제를 자동적으로 해결하지 못하고 사용자 또는 시스템 관리자가 로크자원의 갯수를 보다 큰 값으로 재설정하거나 트랜잭션의 길이를 보다 짧게 조절하도록 하여 처리하고 있다[2]. 그러나 이러한 방법은 사용자의 부담을 가중시키므로 옳바른 해결책이라고 볼 수 없다. 많은 상용 DBMS에서 로크상승을 사용하고 있으나 로크상승에 대한 정형적인 모델은 발표된 바가 없다. 본 논문에서는 로크상승의 작동원리를 분석하여 로크상승에 대한 정형적인 모델을 제안한다. 특히 로크모드의 충돌로 인하여 로크상승을 수행할 수 없는 상승불가능 로크 개념을 제안한다. 그리고 상승불가능 로크가 로크자원 고갈현상을 발생시키는 주요원인임을 보인다. 또한, 상승불가능 로크의 발생원인을 분석하여, 상승불가능 로크의 갯수를 조절하는 적응형 로크상승 기법을 제안한다.

적응형 로크상승 기법은 동일한 로크자원을 사용할 때 동시에 수행될 수 있는 트랜잭션의 수를 크게 증가시킨다. 기존의 방법들은 과다한 로크요청이 발생하는 경우에 로크자원 고갈현상으로 인하여 시스템의 성능이 급격하게 저하되고, 최악의 경우에는 활성정지가 발생한다. 그러나 적응형 로크상승 기법은 로크자원 고갈현상의 발생을 억제하고, 단계적으로 트랜잭션의 순차수행을 유도하여 시스템의 성능을 단계적으로 저하(graceful degradation)시켜 활성정지의 발생을 방지한다.

본 논문의 구성은 다음과 같다. 제 2 장에서는 관련 연구로서 계층적 로킹기법과 로크상승에 대하여 설명하고, 기존의 로크상승 기법에 대하여 설명한다. 제 3 장에서는 먼저 직관적으로 쉽게 생각할 수 있는 기본 적응형 로크상승 기법을 제안한다. 제 4 장에서는 로크상승 연산을 위한 로크의 특성을 분석하여 로크상승의 모델을 제안하고, 이 모델을 기반으로하여 트랜잭션의 연속적인 철회가 발생하는 것이 상승불가능 로크의 증가로 인한 것임을 규명한다. 제 5 장에서는 제안한 로크상승 모델하에서 상승불가능 로크의 증가 원인을 분석하여 각각의 원인에 대한 해결책을 제안하고, 그 해결책을 사용하는 적응형 로크상승 기법을 제안한다. 제 6 장에는 실험을 통하여 적응형 로크상승 기법의 우수성을 입증하고, 마지막으로 제 7 장에서 본 논문의 결론을 맺는다.

2. 관련 연구

로크상승 기법은 계층적 로킹기법[4]을 지원하는 데이타베이스 관리 시스템에서 사용할 수 있는 방법이다. 본 장에서는 제 2.1 절에서는 로크의 단위와 계층적 로킹기법에 대하여 설명하고, 제 2.2 절에서는 로크상승의 개념과 효과에 대하여 설명한다. 그리고 제 2.3 절에서는 기존의 로크상승 기법들을 설명하고 문제점을 지적한다.

2.1 로크단위와 계층적 로킹기법

데이타베이스 관리 시스템에서 로킹을 사용하여 동시성을 제어하는 경우 시스템의 성능을 좌우하는 중요한 결정 요소 중의 하나가 로크단위이다[6]. 로크단위가 큰 경우에는 적은 수의 로크로 많은 데이타에 대한 액세스 권한을 획득할 수 있으므로 로킹 오버헤드가 작으나 동시성이 저하된다. 반면에 로크단위가 작은 경우에는 큰 경우에 비하여 같은 양의 데이타를 액세스할 때에 보다 많은 수의 로킹이 필요하므로 로킹 오버헤드가 크지만 불필요한 데이타에 대한 로킹을 줄일 수 있으므로 동시성이 높아진다[4][7][8]. 참고문헌 [8], [9], [10]에서는 시뮬레이션을 통하여 트랜잭션의 성격에 따라 어떤 크기의 로크단위를 사용하는 것이 보다 좋은 성능을 낼 수 있는지에 관한 연구를 수행하였다.

계층적 로킹기법[4]은 트랜잭션의 성격에 따라 로크크기를 결정할 수 있도록 하나의 데이타베이스 관리 시스템에서 여러 크기의 로크단위를 제공하는 방법이다.

이 방법에서는 로크의 단위들이 계층 구조를 이룬다. 상위수준의 로크단위에 대한 로크를 획득한 경우 그 하위수준 로크단위에 대하여 묵시적으로 같은 모드의 로크가 획득된 것으로 간주한다. 이 때 실제로 로킹된 로크는 명시적 로크(explicit lock)[11]라고 하고, 명시적으로 로킹된 로크단위의 하위수준 로크단위에 대하여 묵시적으로 인정되는 로크는 묵시적 로크(implicit lock)[11]라고 한다. 그리고 서로 다른 수준의 로크단위에 대한 로크들 간에 충돌을 일으키는지 여부를 효율적으로 판단하기 위하여 로킹하고자 하는 로크단위의 모든 상위 로크단위에 대해 의도 모드(intention mode) 로크를 획득한다.

계층적 로킹기법에서 사용되는 로크의 모드는 크게 공유 모드(S), 독점 모드(X), 의도 모드(I)로 구분되고, 의도 모드는 다시 의도 공유 모드(IS)와 의도 독점 모드(IX)로 구분된다. 공유 모드 로크는 해당 로크단위와 그 하위수준의 모든 로크단위에 대하여 읽기 연산만을 허용하고 갱신, 삽입, 삭제 연산은 허용하지 않는다. 독점 모드 로크는 해당 로크단위와 그 하위수준의 모든 로크단위에 대하여 읽기, 갱신, 삽입, 삭제 연산을 모두 허용한다. 의도 모드 로크는 해당 로크단위의 하위수준 로크단위에 대하여 공유 또는 독점 모드의 로크를 획득할 의도가 있음을 의미한다.

표 1은 계층적 로킹기법에서 사용되는 각 로크 모드 간의 호환성을 나타낸 것이다[1]. 표 1에서 행은 이미 획득된 로크모드를 의미하고, 열은 새로 요청된 로크의 모드를 의미한다. 표의 각 항목은 이미 획득된 로크모드와 요청된 로크모드 사이에 호환성이 있는지 여부를 나타낸 것으로, T는 호환성이 있음을 의미하고, F는 호환성이 없음을 의미한다.

표 1 로크모드의 호환성

	IS	IX	S	X
IS	T	T	T	F
IX	T	T	F	F
S	T	F	T	F
X	T	F	F	F

계층적 로킹기법을 사용하는 대부분의 데이타베이스 관리 시스템에서 로크의 단위로 화일과 레코드를 사용한다. 그러므로 본 논문에서는 이해를 돕기 위하여 상위수준 로크단위와 하위수준 로크단위라는 용어 대신 각각 화일과 레코드라는 용어를 사용하기로 한다. 그러나 본 논문에서 제안하는 모델과 로크상승 기법은 2단계 이상의 로크단위 계층을 사용하는 경우에도 적용할 수 있다.

2.2 로크상승의 개념

계층적 로킹기법을 사용할 때 대부분의 경우 많은 레코드를 액세스하는 트랜잭션은 파일에 대한 로크를 획득하는 것이 좋다. 반면에 적은 양의 레코드를 액세스하는 트랜잭션은 레코드에 대한 로크를 사용하는 것이 좋다. 그러나 트랜잭션이 얼마나 많은 수의 레코드를 액세스할 지는 특별한 정보가 없는 한 예측할 수가 없다. 그러므로 화일에 대한 로크는 특별한 정보가 있는 경우에만 사용하고, 일반적으로는 레코드에 대한 로크를 사용한다[1].

많은 데이타를 액세스할 때 레코드 로크를 사용하면 매우 많은 수의 로킹으로 인해 로킹 오버헤드가 지나치게 커진다. 특히, 주기억장치의 한계로 인하여 DBMS에서 제공할 수 있는 로크의 개수에는 한계가 있으므로, 과다한 로크 요청이 발생하면 로크자원이 부족하게 된다[1]. 로크자원이 부족하게 되면 로크를 요청한 트랜잭션은 로크자원을 할당받지 못하여 철회된다. 트랜잭션의 철회는 그 트랜잭션이 수행했던 모든 연산을 무효화하여야 하므로 오버헤드가 큰 작업이며, 잦은 트랜잭션 철회는 데이타베이스 관리 시스템의 성능을 크게 저하시킨다.

이러한 현상을 방지하기 위하여 사용하는 방법이 로크상승이다. 로크상승이란 계층적 로킹기법에서 한 트랜잭션이 획득한 여러 개의 레코드 로크들을 하나의 화일 로크로 대체하고, 레코드 로크들은 반환하는 일련의 작업을 의미한다[1][12]. 로크상승은 다음과 같이 화일 로크모드 변환 연산과 레코드 로크 반환 연산의 2 단계로 이루어진다.

1. 화일 로크모드 변환 연산: 화일에 대하여 획득된 의도 모드의 로크를 그 모드에 따라 각각 의도 공유 모드는 공유 모드로, 의도 독점 모드는 독점 모드로 변환하는 연산.

2. 레코드 로크 반환 연산: 로크모드 상승 연산이 수행된 화일에 속한 레코드 로크를 모두 반환하는 연산.

로크상승은 항상 수행할 수 있는 것이 아니라 수행할 수 없는 경우도 있다. 왜냐하면, 화일 로크모드 변환 연산을 수행하면 로크모드가 변환되는데 이때 이미 로킹되어 있는 다른 트랜잭션의 로크모드와 충돌이 발생할 수 있기 때문이다. 예를 들어, 두 트랜잭션 T1과 T2가 화일 F에 대하여 각각 의도 공유 모드와 의도 독점 모드로 로킹했다고 가정하자. 의도 공유 모드와 의도 독점 모드는 충돌이 발생하지 않으므로 두 트랜잭션은 모두

로크를 획득한 상태이다. 이 상태에서 T2가 로크상승을 수행한다고 하자. T2는 화일 로크모드 변환 연산에서 의도 독점 모드를 독점 모드로 변환하려고하지만 독점 모드는 T1이 획득한 의도 공유 모드와 호환되지 않으므로 충돌이 발생하여 화일 로크모드 변환 연산이 수행되지 않는다. 이러한 로크상승의 가능 여부에 대해서는 제 4 절에서 자세히 설명하기로 한다.

2.3 기존의 로크상승 기법

기존의 로크상승 기법들은 각 시스템에 따라서 임시변통적으로(ad hoc) 설계[1]되었기 때문에 각 방법들을 개별적으로 설명하기는 어렵다. 따라서, 본 절에서는 기존의 로크상승 기법을 특징에 따라 트랜잭션 및 화일별 로크상승 기법과 트랜잭션별 로크상승 기법으로 분류하여 설명한다.

2.3.1 트랜잭션 및 화일별 로크상승 기법

트랜잭션 및 화일별 로크상승 기법(LETF: Lock Escalation based on locks per Transaction and File)은 하나의 트랜잭션이 하나의 화일에 대하여 임계값 이상의 레코드 로크를 획득한 경우에 해당 화일에 대하여 로크상승을 수행하는 방법이다[5]. 본 논문에서는 이 때 사용되는 임계값을 **화일별 로크상승 임계값**이라고 부르기로 한다. 화일별 로크상승 임계값은 모든 화일에 대하여 일률적으로 적용된다. 이 방법은 다음과 같은 몇 가지 문제점이 있다.

첫째, 로크자원이 충분히 남아있는 경우에도 로크상승이 수행된다. 어떤 트랜잭션이 화일별 로크상승 임계값 이상의 로크를 획득하였더라도 동시에 수행되는 트랜잭션의 수가 적으면 굳이 로크상승을 수행할 필요가 없다. 왜냐하면 여분의 로크자원이 많이 있으므로 로크자원이 부족해질 가능성이 적기 때문이다. 이러한 현상은 총 로크사용량을 고려하지 않고 각 트랜잭션이 개별적으로 로크상승의 수행여부를 결정하기 때문에 발생하는 문제점이다.

둘째, 로크자원이 부족한 경우에는 화일별 로크상승 임계값 이상의 로크를 획득하지 않은 경우라도 로크상승이 필요하다. 그러나 트랜잭션 및 화일별 로크상승 기법에서는 총 로크사용량을 고려하지 않으므로 로크상승을 수행하지 않는다. 그리하여 다수의 트랜잭션들이 동시에 수행되면, 각 트랜잭션이 획득한 로크의 갯수는 화일별 로크상승 임계값이하라고 하더라도 로크자원 고갈 현상이 발생할 수 있다.

2.3.2 트랜잭션별 로크상승 기법

트랜잭션별 로크상승 기법(LET: Lock Escalation based on locks per Transaction)은 트랜잭션이 화일에 관계없이 기 정의된 임계값 이상의 로크를 획득한 경우에, 액세스 중인 화일 중에 하나를 선택하여 로크상승을 수행하는 방법이다[2]. 본 논문에서는 이때 사용되는 임계값을 **트랜잭션별 로크상승 임계값**이라고 부르기로 한다. 트랜잭션별 로크상승 임계값은 시스템 관리자가 DBMS를 시작하기 전에 설정하며 모든 트랜잭션에 대하여 공통으로 적용된다. 트랜잭션별 로크상승 기법도 총 로크사용량을 고려하지 않는다는 점에서는 트랜잭션 및 화일별 로크상승 기법과 같으므로 동일한 문제가 발생한다.

이러한 문제를 해결하기 위하여 총 로크사용량을 고려하는 방법을 병행하기도 한다[2]. 총 로크사용량이 시스템에서 제공하는 로크 한계와 같아지면, 즉 로크자원이 부족한 경우에는 로크를 요청한 트랜잭션이 트랜잭션별 로크상승 임계값 이상의 로크를 획득하지 않았더라도 자신이 액세스하고 있는 화일 중에서 하나를 선정하여 로크상승을 수행하는 것이다.

그러나 이와 같이 총 로크사용량을 고려하더라도 문제는 완전하게 해결되지 않는다. 그 이유는 다음과 같다. 특정 트랜잭션이 로크상승을 수행하려고 하더라도 더이상 로크상승을 수행할 수 없는 경우가 있다. 트랜잭션이 액세스하고 있는 모든 화일에 대하여 이미 로크상승을 수행하였거나, 또는 액세스하고 있는 모든 화일이 로크상승을 시도할 때 로크모드의 충돌이 발생하는 경우이다. 이 때에는 다른 트랜잭션이 로크상승을 수행하면 된다. 그러나 트랜잭션별 로크상승 기법에서는 로크를 요청한 트랜잭션을 철회한다. 이러한 현상이 발생하는 이유는 로크상승을 트랜잭션별로 개별적으로 수행하기 때문이다.

3. 기본 적응형 로크상승 기법

총 로크사용량을 고려하지 않는 기존의 로크상승 기법들과는 달리 수행 중인 트랜잭션들이 사용하고 있는 로크의 총합이 임계값을 초과하는 경우, 적당한 트랜잭션과 화일을 선정하여 로크상승을 수행하는 방법을 생각할 수 있다. 본 논문에서는 이 방법을 **기본 적응형 로크상승 기법**이라고 하고, 이 때 로크상승의 기준이 되는 임계값을 **총 로크사용량 기반 로크상승 임계값**이라고 부르기로 한다.

총 로크사용량 기반 로크상승 임계값은 시스템에서 제공할 수 있는 총 로크자원의 수보다 어느정도 작은 값으로 설정한다. 그 이유는 다음과 같다. 총 로크사용량 기반 로크상승 임계값을 시스템의 총 로크자원의 수와 동일하게 설정하면, 더 이상 로크상승을 수행할 수

없는 경우에는 트랜잭션이 반드시 철회하게 된다. 그러나 총 로크사용량 기반 로크상승 임계값을 총 로크자원 수보다 작게 설정하면, 남은 로크자원을 사용하면서 로크상승이 가능해질 때까지 기다릴 수 있다.

이 방법은 총 로크사용량을 고려하고 전역적으로 로크상승을 수행할 트랜잭션과 화일을 결정하므로 불필요하게 트랜잭션이 철회하는 문제는 발생하지 않는다. 그러나 기본 적응형 로크상승 기법은 더 이상 로크상승을 수행할 수 없을 때까지 여분으로 남겨둔 로크자원이 사용되지 않는다. 그러므로 총 로크자원의 수를 총 로크사용량 기반 로크상승 임계값 수준으로 설정한 것과 유사한 효과가 발생한다. 이것은 결과적으로 로크상승을 일찍 수행하여 동시성을 저하시킨다. 또한, 여분의 로크자원을 모두 소모한 후에도 로크자원이 부족한 경우에는 여전히 순환재시작과 활성정지가 발생한다.

4. 로크상승 모델

본 절에서는 문제의 체계적인 해결을 위하여 로크상승의 관점에서 로크의 특성을 분석하고 로크상승에 대한 정형적 모델을 제안한다. 분석의 단순화를 위하여 다음을 가정한다. 트랜잭션은 레코드 수준의 로크를 사용한다고 가정한다. 즉, 화일에 대하여는 IS 또는 IX 모드의 로크만을 요청하며, 화일에 대한 S 또는 X 모드의 로크는 로크상승을 통해서만 획득된다. 로크의 반환은 로크상승으로 인한 반환을 제외하면 트랜잭션 종료시에 반환하는 엄정 2단계 로킹 규약[13]을 따르는 것으로 가정한다.

화일의 상태는 로크상승가능 여부에 따라 **자유 상태**, **상승가능 상태**, **상승불가능 상태**, 그리고 **상승완료 상태**의 네가지 상태로 구분된다. 각각을 다음과 같이 정의한다.

정의 1: 어떤 트랜잭션에 의해서도 로킹되지 않은 화일은 **자유 상태**에 있다고 정의한다. 자유 상태의 화일에 대해서는 레코드 수준의 로크가 존재하지 않는다.

정의 2: 로크모드의 충돌없이 로크모드 상승 연산이 수행될 수 있는 화일은 **상승가능 상태**에 있다고 정의한다. 그리고 상승가능 상태의 화일에 속한 레코드 로크들을 **상승가능 로크**라고 정의한다.

정의 3: 로크모드 상승 연산을 시도할 때 로크모드의 충돌이 발생하는 화일은 **상승불가능 상태**에 있다고 정의한다. 그리고 상승불가능 상태의 화일에 속한 레코드 로크들을 **상승불가능 로크**라고 정의한다.

정의 4: 로크상승이 되어 공유 또는 독점 모드로 화일 수준에서만 로킹된 화일은 **상승완료 상태**에 있다고 한다. 상승완료 상태인 화일에 대해서는 명시적인 레코드 로크가 존재하지 않는다.

화일의 상태는 해당 화일을 로킹한 로크모드들의 조합에 의하여 구분된다. 표 2는 화일의 상태와 그 상태에 해당하는 로크모드의 조합을 나타낸 것이다. 표에서 첫번째 열은 화일의 상태를 의미하고, 두번째 열은 로크모드의 조합을 의미한다. 두번째 열에서 $\{IX\}^{1+}$는 IX 모드의 로크가 하나 이상 획득되어 있는 상태를 의미하며, $\{IX\}^{2+}$는 IX 모드의 로크가 두개이상 획득되어 있는 상태를 의미한다. 또한 $\{IX\}^{1+}$, $\{IS\}^{1+}$는 각각 하나 이상의 IX 모드 로크와 하나 이상의 IS 모드 로크가 동시에 획득되어 있는 상태를 의미한다. 본 표에서 나타난 로크모드의 조합은 주어진 가정 하에서 가능한, 즉 서로 충돌이 발생하지 않는, 모든 로크모드의 조합을 나타내고 있다.

표 2 화일에 대한 로크모드의 조합에 따른 화일의 상태

화일의 상태	화일에 대한 로크모드의 조합
자유 상태	
상승완료 상태	$\{X\}^{1}$
	$\{S\}^{1+}$
상승불가능 상태	$\{IX\}^{2+}$
	$\{IX\}^{1+}$, $\{IS\}^{1+}$
상승가능 상태	$\{IX\}^{1}$
	$\{IS\}^{1+}$
	$\{S\}^{1+}$, $\{IS\}^{1+}$

로크모드의 조합은 새로운 로크의 획득, 로크의 반환, 그리고 로크상승에 의하여 변환되고 그에 따라 화일의 상태도 변화한다. 그림 1은 화일의 상태가 변화하는 것을 상태도(state diagram)로 표현한 것이다. 그림에서 각 노드는 화일에 대한 로크모드의 조합을 의미한다. 단, NL(No Lock)은 해당 화일에 대하여 획득된 로크가

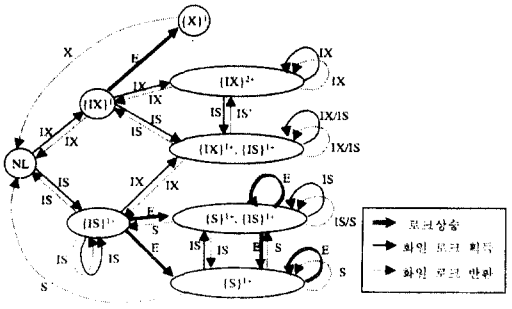


그림 1 화일의 상태 변화도

없다는 것을 의미한다.

상태도에는 다섯가지 종류의 화살표가 존재한다. 두줄 화살표는 로크상승을 의미하고,한줄 실선 화살표와 점선 화살표는 각각 로크의 획득과 반환을 의미한다. 각 화살표에 표시된 이름은 해당 화살표가 나타내는 로크의 모드 또는 로크상승을 의미한다. 예를들어, IS로 표시된 한줄 실선 화살표는 IS 모드의 로크 획득을 의미하고, E로 표시된 두줄 화살표는 로크상승을 의미한다.

예 1은 화일의 상태 변화를 설명하기 위하여 읽기 전용 트랜잭션 T1과 갱신 트랜잭션 T2가 화일 F를 동시에 액세스하는 경우에 F의 상태 변화를 설명한 것이다.

예 1: 자유 상태인 화일 F에 대하여 읽기 전용 트랜잭션 T1과 갱신 트랜잭션 T2가 다음과 같은 순서로 수행된다고 하자.

단계 1. T1 : F에 대한 IS 모드 로크 획득

단계 2. T1 : F에 속한 레코드에 대한 S 모드 로크 획득

단계 3. T2 : F에 대한 IX 모드 로크 획득

단계 4. T2 : F에 속한 레코드에 대한 X 모드 로크 획득

단계 5. T1 : 수행 완료(commit)

단계 6. T2 : F에 대한 로크상승 수행

단계 7. T2 : 수행 완료(commit)

단계 1에서 F의 상태는 그림 1에 표현된 것과 같이 노드 [NL]에서 노드[$\{IS\}^{1+}$]로 변환된다. 이는 자유 상태에서 상승가능 상태로 변환된 것을 의미한다. 단계 2에서는 화일 수준에 대한 로크모드의 조합에 변화가 없다. 그러므로 화일의 상태도 변환되지 않는다. 단계 3에서는 [$\{IS\}^{1+}$]에서 노드 [$\{IX\}^{1+}$, $\{IS\}^{1+}$]로 변환된다. 이는 상승가능 상태에서 상승불가능 상태로 변환된 것을 의미한다. 단계 4에서도 단계2와 마찬가지로 화일 수준에 대한 로크모드의 조합에 변화가 없다. 그러므로 화일의 상태도 변환되지 않는다. 단계 5에서는 트랜잭션 T1이 완료하여 F에 대한 로크를 반환하므로 노드 [$\{IX\}^{1+}$, $\{IS\}^{1+}$]에서 노드 [$\{IX\}^{1}$]로 변환된다. 이는 상승불가능 상태에서 상승가능 상태로 변환된 것을 의미한다. 단계 6에서는 T2가 로크상승을 수행하므로 노드 [$\{IX\}^{1}$]에서 [$\{X\}^{1}$]로 변환된다. 이는 상승가능 상태에서 상승완료 상태로 변환된 것을 의미한다. 마지막으로 단계 7에서 T2가 완료하면 F에 대한 X모드의 로크가 반환되어 노드 [$\{X\}^{1}$]에서 [NL]로 변환된다. 이는 상승완료 상태에서 자유 상태로 변환된 것을 의미한다.

그림 1에서 [$\{IX\}^{1}$] 또는 [$\{IS\}^{1+}$]와 [$\{S\}^{1+}$, $\{IS\}^{1+}$]는 모두 상승가능 상태라는 점에서는 동일하지만 약간의 차이가 있다. 화일의 상태가 [$\{IX\}^{1}$] 또는 [$\{IS\}^{1+}$] 상태인 경우에는 새로운 로크의 획득으로 인하여 상승불가능 상태로 변환될 수 있다. [$\{IX\}^{1}$] 상태인 경우에는 새로운 IX 모드 또는 IS 모드의 로크가 획득될 때 상승불가능 상태인 [$\{IX\}^{2+}$] 또는 [$\{IX\}^{1+}$, $\{IS\}^{1+}$] 상태로 변환된다. [$\{IS\}^{1+}$] 상태인 경우에는 새로운 IX 모드의 로크가 획득될 때 상승불가능 상태인 [$\{IX\}^{1+}$, $\{IS\}^{1+}$] 상태로 변환된다. 그러나 [$\{S\}^{1+}$, $\{IS\}^{1+}$] 상태는 상승불가능 상태로 변환되지 않는다. 새로운 IX 모드의 로크가 요청되더라도 기존의 S 모드 로크와 충돌이 발생하여 로크가 획득되지 않기 때문이다.

이와 같이, 상승가능 상태는 상승불가능 상태로 변환될 수 있는 상태와 변환될 수 없는 상태로 구분될 수 있다. 각각의 상태를 다음과 같이 정의한다.

정의 5: 상승가능 상태 중 화일에 대한 새로운 로크의 획득으로 인하여 상승불가능 상태로 변환될 수 있는 상태를 **불안전 상승가능 상태**라고 정의한다.

정의 6: 상승가능 상태 중 화일에 대한 새로운 로크가 획득되더라도 상승불가능 상태로 변환되지 않는 상태를 **안전 상승가능 상태**라고 정의한다.

그림 2는 그림 1을 화일의 상태를 위주로 간략하게 표현한 것이다. 그림에서 각 노드는 화일의 상태를 의미하고, 각 화살표는 그림 1에서와 같은 의미를 가진다. 단, 레코드 로크의 획득과 반환을 표시하는 끝이 동그란 실선과 점선을 추가하였다. 이는 제 5.1 절에서 상승불가능 로크의 증가원인을 설명하기 위한 것으로서 레코드 로크의 획득과 반환은 화일의 상태 변화에 영향을 미치지 않는다.

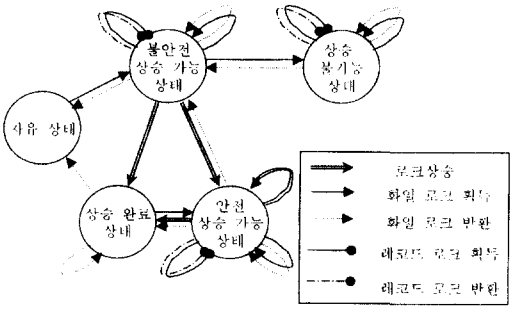


그림 2 화일의 간략한 상태 변화도

그림 2에서 로크상승의 몇가지 특징을 알 수 있다. 첫째, 로크상승은 상승가능 상태의 화일에 대해서만 수행

할 수 있다. 상승완료 상태 또는 자유 상태의 화일에 대해서는 로크상승의 정의에 의하여 로크상승을 수행할 수 없다. 또한, 상승불가능 상태의 화일에 대해서는 로크상승을 시도하더라도 로크모드의 충돌이 발생하여 대기하게 되며, 충돌을 발생시킨 모든 로크가 반환되어 화일의 상태가 상승가능으로 변환된 후에야 로크상승 연산이 수행된다. 그러므로 상승불가능 상태의 화일에 대하여 로크상승을 수행하는 것은 무의미하다.

둘째, 전역적 접근법을 사용하여 로크상승을 수행하면, 상승가능 로크가 존재하는 동안에는 언제든지 로크상승을 수행하여 로크자원을 반환 받을 수 있다. 그러므로 로크자원의 부족으로 인하여 트랜잭션이 철회하는 현상이 발생하지 않는다. 그러나 상승가능 로크가 존재하지 않는 경우, 즉 모든 레코드 로크가 상승불가능 로크인 경우에는, 로크자원이 부족해지더라도 더이상 로크상승을 수행할 수 없으므로 트랜잭션이 철회된다. 그러므로 전역적 접근법을 사용하여 로크상승을 수행하는 경우 상승불가능 로크의 증가가 로크자원의 부족으로 인한 트랜잭션의 철회를 발생시키는 원인임을 알 수 있다.

셋째, 로크상승은 단순히 총 로크사용량을 기준으로 수행하는 것보다는 상승불가능 로크의 갯수를 기준으로 수행하여야 한다. 왜냐하면 로크사용량이 동일하더라도 상승불가능 로크의 갯수에 따라 로크자원의 부족으로 인한 트랜잭션의 철회가 발생할 가능성이 달라지기 때문이다.

5. 적응형 로크상승 기법

적응형 로크상승 기법은 기본적으로는 상승불가능 로크의 갯수에 따라 로크상승의 수행 여부를 결정하는 방법이다. 즉, 수행 중인 트랜잭션들이 획득한 상승불가능 로크의 총합이 로크상승 임계값을 초과하는 경우에 로크상승을 수행하는 방법이다. 이 방법은 기본 적응형 로크상승 기법에 상승불가능 로크 개념을 도입하여 확장한 방법이라고 할 수 있다. 그러나 이러한 기본적인 확장 이외에도 상승불가능 로크의 증가원인을 해결함으로써 성능을 더욱 향상시킨다. 본 장에서는 먼저 제안한 로크상승 모델하에서 상승불가능 로크의 증가원인을 분석한다. 그리고 기존의 로크상승이 상승불가능 로크의 증가에 미치는 영향을 분석하여 이를 보완한 새로운 해결방법을 제안한다.

5.1 상승불가능 로크의 증가 원인

그림 3은 그림 2에서 상승불가능 상태와 관계 있는 부분만을 다시 그린 것이다. 그림 3에서 상승불가능 상태와 관련이 있는 화살표 중 원인 1, 2, 3으로 표시된 실선 화살표가 로크의 증가 원인을 나타낸다. 각각의 화살표가 나타내는 상승불가능 로크의 증가 원인은 다음과 같다.

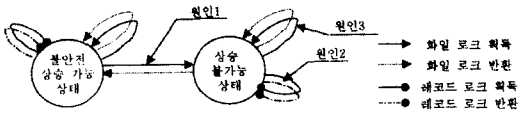


그림 3 상승불가능 상태의 상태 변화도

원인 1. 상태 변환: 화일의 상태가 불안전 상승가능 상태에서 상승불가능 상태로 변환됨에 따라 해당 화일에 속한 레코드에 대한 로크들이 모두 상승가능 로크에서 상승불가능 로크로 변환되어 상승불가능 로크가 증가한다.

원인 2. 상승불가능 로크의 추가 획득: 상승불가능 상태의 화일을 액세스 중인 트랜잭션이 추가적으로 해당 화일에 속한 레코드 로크를 획득함으로써 상승불가능 로크가 증가한다.

원인 3. 트랜잭션 수 증가: 상승불가능 상태의 화일을 액세스하는 트랜잭션의 수가 증가하여 증가한 트랜잭션들이 향후 해당 화일에 속한 레코드 로크를 획득할 때에 원인 2에 의하여 상승불가능 로크가 증가한다.

기존의 연구에서는 로크상승을 단순히 로크의 사용량을 조절하기 위한 도구로만 간주하였다. 즉, 다수의 레코드 로크를 하나의 화일 로크로 대체하고 레코드 로크를 반환함으로써 로크사용량을 감소시키는 방법으로 사용하였다. 그러나 제 4 장에서 제안한 로크상승 모델에 의하면 로크상승이 상승불가능 로크의 증가에 미치는 영향을 분석할 수 있다.

그림 3에서 상승불가능 상태에서 시작되는 두줄 화살표는 불안전 상승가능 상태를 상승완료 또는 안전 상승가능 상태로 변환시키는 로크상승에 해당하는 것이다. 이는 상승불가능 로크의 세가지 증가 원인 중 상태 변환에 의한 증가 원인(원인 1)을 제거하는 역할을 한다. 반면에 안전 상승가능 상태에서 화일에 대한 로크상승은 단순히 로크자원을 반환하는 역할만을 수행한다. 그러므로 상승불가능 로크의 증가를 억제하기 위해서는 불안전 상승가능 상태의 화일에 대하여 로크상승을 수행하는 것이 필요하다. 기존의 로크상승 기법들이나 기본 적응형 로크상승 기법은 모두 화일의 상태를 고려하지 않고 로크상승을 수행하므로 상승불가능 로크의 증가를 효과적으로 억제할 수 없다.

5.2 상승불가능 로크의 증가에 대한 해결책

5.2.1 준로크상승

로크상승은 화일의 상태 변환에 의한 상승불가능 로크의 증가(원인 1)를 방지할 수 있다. 그러나 로크상승을 수행한 트랜잭션이 완료할 때까지 로크단위가 커진 상태로 유지하여야 하므로 동시성이 저하된다. 본 절에서는 상승불가능 로크의 증가원인 1을 해결하면서도 위와 같은 로크상승의 단점을 개선하여 필요에 따라 로크단위를 레코드 수준으로 환원할 수 있는 준로크상승을 제안한다.

불안전 상승가능 상태의 화일에 대하여 로크상승을 수행함으로써 상승불가능 로크의 증가를 억제하는 것은 로크상승의 2단계 연산 중 첫번째에 해당하는 화일 로크모드 변환 연산에 의한 것이다. 왜냐하면, 화일 로크모드 변환 연산이 수행되면 해당 화일에 대한 로크모드의 조합이 달라지고 이에 따라 화일의 상태가 안전 상승가능 상태로 변환되기 때문이다.

반면에, 두번째 단계에 해당하는 레코드 로크 반환 연산은 상승불가능 로크의 증가를 억제하는 역할을 수행하지 않는다. 단지, 사용되고 있던 로크자원을 반환하는 역할만을 수행할 뿐이다. 그러므로 상승불가능 로크의 증가를 억제하기 위해서는 단지 화일 로크모드 변환 연산만을 수행하는 것으로 충분하다.

이로써 로크상승을 수행함에 있어서 화일 로크모드 변환 연산과 레코드 로크반환 연산을 반드시 연속적으로 수행해야만하는 것은 아니라는 것을 알 수 있다. 즉, 상승불가능 로크의 증가를 억제하기 위해서는 화일 로크모드 변환 연산만을 수행하면 되고, 로크자원이 부족하여 로크자원의 반환이 필요한 경우에만 레코드 로크 반환 연산을 수행하면 된다. 이러한 개념을 바탕으로 준로크상승을 다음과 같이 정의한다.

정의 7: 레코드 로크반환 연산은 수행하지 않고 화일 로크모드 변환 연산만을 수행하는것을 **준로크상승**(*semi lock escalation*)이라고 정의한다. 준로크상승을 수행하더라도 해당 화일에 속한 레코드 로크는 계속 획득하여야 한다. 왜냐하면 레코드 로크 반환 연산이 수행기 전에는 로크하강을 수행할 수 있도록 하기 위한 것이다.

준로크상승을 이용하여 다음과 같은 로크상승 기법을 생각할 수 있다. 수행 중인 트랜잭션들이 획득한 상승불가능 로크의 총합이 로크상승 임계값을 초과하는 경우에는 로크상승 대신 먼저 준로크상승을 수행함으로써 상승불가능 로크의 증가를 억제한다. 그리고 로크자원이 부족하게된 경우에만 이미 준로크상승이 수행된 화일에 대하여 레코드 로크 반환 연산을 수행한다. 이 방법은 상승불가능 로크의 증가를 억제한다는 관점에서는 준로크상승 대신 바로 로크상승을 수행한 것과 동일한 효과를 갖는다. 뿐만아니라 아직 레코드 로크반환 연산을 수행하지 않은 경우에는 상승불가능 로크의 갯수가 감소하여 로크상승 임계값 이하가 되면 로크단위를 작게 환원할 수 있으므로 동시성을 향상시킬 수 있다.

5.2.2 로크블로킹

본 절에서는 이해를 돕기 위하여 먼저 상승불가능 로크의 증가원인 3을 해결하는 방법인 로크블로킹에 대하여 설명하고, 원인 2에 대한 해결 방법에 대해서는 제 5.2.3 절에서 설명한다. 왜냐하면, 로크블로킹은 개념상 준로크상승과 유사하기 때문이다. 이제 상승불가능 로크의 증가원인 1과 3의 공통점을 살펴보고 이에따라 원인 3을 해결하는 로크블로킹에 대하여 설명한다.

상승불가능 로크의 증가원인 중 원인 1과 3은 모두 새로운 화일 로크의 획득에 의해 발생한다. 원인 1은 상승가능 상태의 화일에 대하여 그 상태를 상승불가능 상태로 변환시키는 새로운 화일 로크의 획득에 의한 것이고, 원인 3은 상승불가능 상태의 화일에 대하여 새로운 화일 로크가 획득됨으로써 발생한다. 그러므로 이 두원인은 모두 새로운 화일 로크의 획득을 금지함으로써 제거할 수 있다. 준로크상승은 상승가능 상태의 화일에 대한 새로운 로크 획득을 금지하는 방법으로 로크모드의 조합을 변환시키는 방법을 사용한 것이라고 할 수 있다.

준로크상승은 상승불가능 상태의 화일에 대해서는 적용할 수 없으므로 원인 3을 해결하는 방법으로는 사용할 수 없다. 그러나 유사한 방식으로 원인 3을 해결하는 방법을 생각할 수 있다. 상승불가능 상태의 화일에 대한 새로운 로크 획득을 금지(또는 허용)하는 방법으로 로크 블로킹 설정 연산과 로크 블로킹 해제 연산을 다음과 같이 정의한다.

정의 8: 주어진 상승불가능 상태의 화일에 대하여, 새로운 로크의 획득을 금지하는 연산을 **로크 블로킹 설정 연산**이라고 한다. 또한, 설정된 로크 블로킹을 해제하여 새로운 로크의 획득을 허용하는 연산을 **로크 블로킹 해제 연산**이라고 정의한다. 로크 블로킹 설정 연산이 수행된 화일의 상태를 **로크 블로킹 설정 상태**라고 정의한다.

로크블로킹 설정 연산을 수행하면 해당 화일에 대한 추가적인 로크 획득이 금지되므로해당 화일을 액세스하는 트랜잭션 수가 더이상 증가하지 않는다. 이에 따라 상승불가능 로크의 증가원인 3에 의한 상승불가능 로크의 증가는 더이상 발생하지 않는다.

5.2.3 선택적 강제수행

본 절에서는 상승불가능 로크의 증가원인 2에 대한 해결책을 설명한다. 원인 1과 3이 화일에 대한 로크의 획득에 의하여 발생하는 것과는 달리 원인 2는 레코드 로크의 획득에 의하여 발생한다. 그러므로 준로크상승과 로크블로킹을 사용하여 모든 화일에 대한 추가 로크 획득을 금지하더라도 원인 2에 의하여 상승불가능 로크가 계속 증가할 수 있다. 이에 따라 로크자원이 부족하게 되어 트랜잭션이 철회함으로써 순환재시작과 활성정지가 발생할 수 있다.

이 현상은 단순히 상승불가능 로크의 추가 획득을 금지하여 트랜잭션을 대기시키는 방법만으로는 해결되지 않고 반드시 트랜잭션을 철회하여야 한다. 그 이유는 다음과 같다. 더 이상 상승가능 상태의 화일이 존재하지 않는 경우에는, 모든 레코드 로크의 획득이 곧 상승불가능 로크의 획득을 의미한다. 그러므로 추가적으로 레코드 로크를 요청하는 트랜잭션들이 모두 대기하게 된다. 심한 경우에는 수행 중인 모든 트랜잭션이 대기할 수도 있다. 이 경우 로크자원을 반환 받아야 대기 중인 트랜잭션들이 로크를 획득할 수 있는데 더 이상 로크상승을 수행할 수는 없으므로 트랜잭션을 철회할 수밖에 없다.

이 때 철회할 트랜잭션을 잘못 선정하면 트랜잭션들이 순환재시작 상태가 되어 활성정지가 발생할 수 있다. 본 논문에서는 활성정지를 방지하기 위하여 철회할 트랜잭션을 결정하는 방법으로 선택적 강제수행을 제안한다.

정의 9: 선택적 강제수행(selective relief)은 로크자원의 부족으로 철회할 트랜잭션을 선정함에 있어서 특정 트랜잭션을 선정대상에서 제외하여 그 트랜잭션의 완료를 보장하는 방법이다. 이 때, 완료가 보장되는 트랜잭션을 **불사 트랜잭션**(*immortal transaction*)이라고 정의한다. 불사 트랜잭션은 강제적으로 로크상승을 수행할 수 있도록 철회할 트랜잭션을 선정한다. 즉, 불사 트랜잭션은 액세스하는 모든 화일에 대하여 로크상승을 수행하고, 이 때 로크모드의 충돌을 발생시키는 로크를 획득하고 있는 트랜잭션들은 모두 철회한다. 단, 불사 트랜잭션이 레코드 로크가 아닌 화일 로크를 획득하려고 하는 경우에는 로크모드의 충돌을 발생시킨 트랜잭션들을 모두 철회한다.

이렇게 함으로써 불사 트랜잭션은 더이상 레코드 로크를 획득할 필요가 없다. 그러므로 로크모드의 충돌이나 로크자원의 할당 때문에 대기할 필요가 없게 된다. 선택적 강제수행을 사용하면 불사 트랜잭션은 철회되지 않고 수행을 계속하므로 활성정지가 발생하지 않음을 보장한다.

또한, 상승불가능 로크의 갯수를 감소시키는 역할을 한다. 그 이유는 다음과 같다. 불사 트랜잭션이 수행하는 로크모드의 충돌을 발생시키는 트랜잭션을 철회하고 로크상승을 수행함으로써 상승불가능 상태였던 화일이 상승가능 상태로 변환된다. 이로 인하여 해당 화일에 속한 레코드에 대한 로크들도 모두 상승불가능 로크에서 상승가능 로크로 변환된다.

5.3 상승불가능 로크에 기반한 로크상승 기법

적응형 로크상승 기법에서는 상승불가능 로크의 증가를 억제하기 위하여 준로크상승과 로크블로킹을 사용한다. 이러한 억제방법에도 불구하고 상승불가능 로크가 계속 증가하여 로크자원이 부족해지는 경우에는 선택적 강제수행을 사용하여 활성정지를 방지한다. 본 절에서는 적응형 로크상승 기법의 알고리즘을 설명한다.

그림 4는 적응형 로크상승 기법의 기본 알고리즘을 나타낸 것이다.알고리즘은 세부분으로 구성된다. 첫번째 부분(단계 A)은 트랜잭션이 로크를 요청할 때마다 수행되고, 두번째 부분(단계 B)은 모든 트랜잭션이 대기상태일 때 데몬 프로세스에 의해 수행되며, 마지막 부분(단계 C)은 트랜잭션이 로크를 반환할 때마다 수행된다. 단계 A와 C는 로크를 요청 또는 반환한 트랜잭션에 의해 수행된다. 그러나 단계 B는 모든 트랜잭션이 대기상태인 일종의 교착상태이므로 데몬 프로세스에 의해 수행된다.

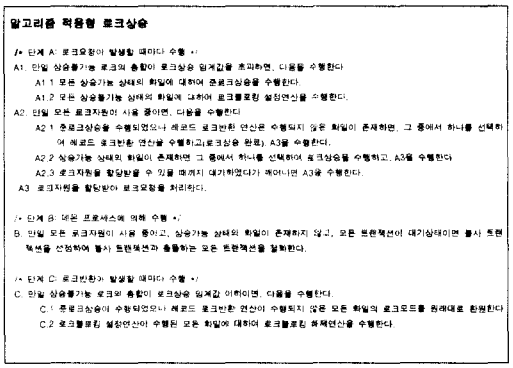


그림 4 적응형 로크상승 알고리즘

첫번째 부분(단계 A)의 단계 A.1은 상승불가능 로크의 총합이 로크상승 임계값을 초과한 경우에 준로크상승(A1.1)과 로크블로킹 설정연산(A1.2)을 수행하는 것을 나타낸 것이다. 단계 A1.1에서 상승가능 상태의 화일이 상승불가능 상태로 변환되는 것을 방지하기 위하여, 모든 상승가능 상태의 화일에 대하여 준로크상승을 수행한다(원인 1제거). 그리고 단계 A1.2에서는 상승불가능

상태의 화일을 액세스하는 트랜잭션의 수가 증가하는 것을 방지하기 위하여, 모든 상승불가능 상태의 화일에 대하여 로크블로킹 설정연산을 수행한다(원인 3제거).

단계 A2는 더이상 여분의 로크자원이 존재하지 않는 경우의 처리방법을 나타낸 것이다.단계 A2.1은 준로크상승만 수행되고 레코드 로크 반환연산은 아직 수행되지 않은 화일이 존재하는 경우이다. 이 경우에는 이 화일들 중에서 하나를 선정하여 레코드 로크 반환연산을 수행함으로써 로크자원을 반환받고, 단계 A3을 수행한다. 단계 A2.2는 A2.1의 조건은 만족되지 않지만 상승가능 상태의 화일이 존재하는 경우이다. 이 경우에는 이 화일들 중에서 하나를 선정하여 로크상승을 수행함으로써 로크자원을 반환 받고, 단계 A3을 수행한다. 단계 A2.1과 A2.2에서 안전 상승가능 상태의 화일과 한번에 가장 많은 수의 로크자원이 반환될 수 있는 화일을 우선적으로 선정한다. 이는 로크상승으로 인한 동시성저하를 줄이고, 로크상승의 횟수를 줄이기 위한 것이다. 단계 A2.3은 A2.1과 A2.2의 조건이 모두 만족하지 않는 경우이다. 이 경우에는 로크상승을 이용하여 로크자원을 반환받을 수가 없다. 그러므로 로크를 요청한 트랜잭션은 로크자원이 반환될 때까지 대기하였다가 로크자원을 할당받을 수 있게되면 깨어나서 단계 A3을 수행한다. 단계 A3은 여분의 로크자원이 존재하는 경우이므로 로크자원을 할당하여 로크요청을 처리한다.

두번째 부분(단계 B)은 더 이상 여분의 로크자원이 존재하지 않고, 모든 트랜잭션이 대기상태이며, 상승가능 상태의 화일이 존재하지 않는 경우에만 수행된다. 이 경우는 로크상승을 수행할 수 없고, 모든 트랜잭션이 대기 상태이다. 이는 일종의 교착상태라고 할 수 있다. 그러므로 데몬 프로세스가 선택적 강제수행 개념을 이용하여 불사 트랜잭션을 선정하고[2] 불사 트랜잭션과 로크모드의 충돌을 발생시키는 모든 트랜잭션을 철회한다.

세번째 부분(단계 C)은 상승불가능 로크의 총합이 로크상승 임계값 이하로 감소하는 경우의 처리방법을 나타낸 것이다. 이 때에는 더 이상 상승불가능 로크의 증가를 억제할 필요가 없다. 그러므로 단계 C.1에서는 준로크상승이 수행되었으나 레코드 로크반환 연산은 수행되지 않은 모든 화일의 로크모드를 원래대로 환원한다. 즉, 로크하강을 수행하는 것이다. 단계 C.2에서는 단계 A.2에서 수행된 로크블로킹 설정연산을 해제하기 위하여 로크블로킹 해제연산을 수행한다. 이로 인하여 준로크상승과 로크블로킹에 의해 대기 중이던 모든 트랜잭션이 수행을 계속하게 된다.

기본 적응형 로크상승 기법에서는 단순히 총 로크사용량이 로크상승 임계값을 초과하는 경우에 로크상승을 수행한다. 반면에 적응형 로크상승 기법에서는 상승불가능 로크의 총 갯수가 임계값을 초과하는 경우에 준로크상승과 로크블로킹을 수행하여 상승불가능 로크의 증가를 억제한다. 그러므로 상승불가능 로크의 갯수가 임계값을 초과하지 않으면 로크자원이 모두 사용될 때까지 로크상승을 수행하지 않는다. 이러한 특징으로 인하여 기본 적응형 로크상승 기법에서와 같이 로크자원의 총 갯수를 줄인 것과 같은 효과는 나타나지 않는다. 이 경우에 적응형 로크상승 기법의 오버헤드는 단지 상승불가능 로크의 총합을 유지하는 것과 로크요청 및 반환연산을 수행할 때 상승불가능 로크의 총합이 로크상승 임계값을 초과하는지를 확인하는 것뿐이다. 이 연산은 매우 간단한 연산이므로 오버헤드가 작다.

6. 성능 평가

본 절에서는 적응형 로크상승 기법과 기존의 로크상승 기법의 성능을 시뮬레이션을 통하여 비교한다. 먼저, 제 6.1 절에서는 시뮬레이션을 수행하기 위한 시스템 구조와 실험 환경에 대하여 설명하고, 제 6.2 절에서는 실험 결과를 설명한다.

6.1 시스템 구조 및 실험 환경

본 논문에서 사용한 시스템의 구조는 그림 5와 같이 크게 클라이언트와 서버로 구성된다. 클라이언트는 트랜잭션 생성자를 포함하는데 트랜잭션 생성자는 트랜잭션을 생성하는 역할을 하며, 하나의 트랜잭션이 완료되면 다시 트랜잭션을 생성하는 작업을 반복한다.

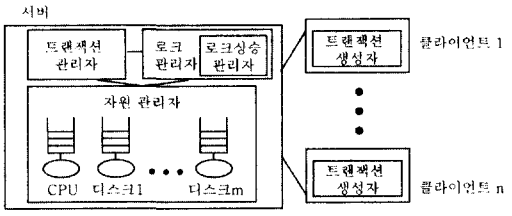


그림 5 실험을 위한 시스템 구조

서버는 크게 트랜잭션 관리자, 로크 관리자, 그리고 자원 관리자로 구성된다. 트랜잭션 관리자는 클라이언트의 요청에 의해 생성된 트랜잭션의 수행을 담당한다. 로

2) 불사 트랜잭션을 선정하는 방법은 여러가지가 있을 수 있으나 본 논문의 연구범위가 아니므로 생략한다. 본 논문의 실험에서는 가장 간단한 방법으로서 수행 중인 트랜잭션 중 가장 먼저 시작된 트랜잭션을 불사트랜잭션으로 선정하였다.

크 관리자는 트랜잭션이 요청하는 로킹 작업을 수행하고, 교착상태를 검출한다. 또한 로크 관리자는 로크상승 관리자를 포함하여 기 정의된 방법에 따라 로크상승을 수행한다. 자원 관리자는 하나의 CPU와 m개의 디스크에 대한 트랜잭션의 요청을 처리한다. 각 디스크에 대한 요청은 FCFS 방식으로 처리한다. 본 논문에서는 로크 자원의 사용량 변화를 관측할 때, 디스크에 의한 병목현상으로 그 결과가 왜곡되는 것을 피하기 위하여 실험 4를 제외한 모든 실험은 디스크 큐에서의 대기가 없는 것으로 가정한다. 그리고 실험 4에서 디스크 큐에서의 대기를 고려한 경우에 대한 실험을 수행한다.

DBMS에서 제공할 수 있는 총 로크자원의 갯수는 1000개로 설정한다. 실험을 위한 데이타베이스는 100개의 화일로 구성되며, 실험 3을 제외한 모든 실험에서 각 화일은 100000개의 레코드로 구성된다. 실험 3에서는 각 화일에 속한 레코드의 갯수가 서로 다른 경우를 실험한다.

이제 트랜잭션 모델에 대하여 설명한다. 트랜잭션은 계층적 로킹기법을 사용하여 로크를 획득한다. 계층적 로킹기법에서 사용하는 로크단위로는 화일과 레코드의 2단계 로크단위를 사용하는 것으로 가정한다. 이것은 실제로 대부분의 상용 데이타베이스 관리 시스템에서 사용하고 있는 로크단위이다. 트랜잭션은 크게 읽기전용 트랜잭션과 갱신 트랜잭션으로 구분한다. 읽기전용 트랜잭션은 액세스하는 모든 화일에 대하여 IS 모드의 로크를, 갱신 트랜잭션은 IX 모드의 로크를 요청하고, 화일에 대한 S 또는 X 모드의 로크는 로크상승으로 인해서만 획득되는 것으로 가정한다. 또한 로크상승으로 인한 로크의 반환을 제외한 로크의 반환은 트랜잭션이 완료 또는 철회할 때 일괄적으로 이루어지는 엄정 2단계 로킹 규약을 따르는 것으로 가정한다. 트랜잭션은 교착상태가 발생하여 희생자로 선정되거나 로크자원을 할당받지 못하는 경우에만 철회되며, 트랜잭션에 의한 자발적인 철회는 발생하지 않는 것으로 가정한다. 철회된 트랜잭션은 바로 재시작한다.

트랜잭션이 액세스하는 레코드 갯수의 분포는 평균을 100으로하는 지수분포를 가정한다. 이 값은 로크상승을 수행하지 않는 경우 트랜잭션이 획득하는 레코드 로크 갯수와 동일하다. 그리고 실험 1, 3, 4에서는 각 트랜잭션이 두개의 화일을 액세스하도록 설정하며, 실험 2에서 트랜잭션이 액세스하는 화일의 갯수가 성능에 미치는 영향을 분석한다.

트랜잭션의 연산은 크게 트랜잭션의 CPU 연산, 디스크 I/O, 그리고 로킹 및 로크상승연산으로 구성된다. 실험의 각 로크상승 기법들에 있어서 공통적으로 CPU 연산과 디스크 I/O 에 소요되는 시간의 비를 2:1로 설정하였다.[3] 로킹과 로크상승은 CPU 연산과 디스크 I/O에 소요되는 시간의 비에 포함하지 않고 시뮬레이션이 아닌 실제 구현을 통하여 전체 소요된 시간에 포함하였다. 그러므로 각 로크상승 기법의 오버헤드는 실험 결과에 반영되어 있다.

실험은 로크상승을 하지 않는 경우(No Escalation으로 표기), 트랜잭션 및 화일별 로크상승 기법(LETF로 표기), 트랜잭션별 로크상승 기법(LET로 표기), 기본 적응형 로크상승 기법(Simple로 표기), 그리고 적응형 로크상승 기법(Adaptive로 표기)의 다섯가지 방법에 대하여 수행한다. 이 때, 각 로크상승 기법의 로크상승 임계값은 LETF의 경우에는 한 트랜잭션이 하나의 화일에 대하여 획득하는 평균 로크 갯수의 80%로 설정하고, LET의 경우에는 한 트랜잭션이 획득하는 평균 로크 갯수의 80%로 설정한다. Simple과 Adaptive의 경우에는 로크상승 임계값을 전체 로크 갯수의 80%로 설정한다.

모든 실험은 총 10000개의 트랜잭션이 완료될 때까지의 성능을 측정하되 동시에 수행하는 트랜잭션의 갯수를 증가시키면서 각 로크상승 기법의 성능을 평가한다. 실험 1에서는 읽기전용 트랜잭션과 갱신 트랜잭션의 비에 의한 영향을 검토하기 위하여 그 비가 8:2인 경우와 2:8인 경우의 성능을 평가한다. 실험 2에서는 다른 조건은 실험 1과 동일한 상태에서 트랜잭션이 액세스하는 화일의 갯수를 1, 5, 10으로 증가시키면서 트랜잭션이 액세스하는 화일의 갯수에 따른 영향을 측정한다. 실험 3에서는 화일의 크기가 동일하지 않고 편중된 경우의 성능을 평가한다. 실험 4에서는 다른 조건은 실험 1과 동일한 상태에서 적은 수의 디스크에 데이타베이스가 분산되어 저장되어 있고 디스크 큐를 고려했을 때의 성능을 평가한다.

성능 평가의 척도로는 트랜잭션당 평균 철회 횟수 ($=\frac{\text{총 철회 횟수}}{\text{완료된 트랜잭션의 수}}$), 트랜잭션의 평균 응답시간, 그리고 단위시간당 트랜잭션 처리율($=\frac{\text{완료된 트랜잭션의 수}}{\text{총 철회 횟수}}$)을 사용한다. 트랜잭션당 평균 철회 횟수는 하나의 트랜잭션이 완료할 때까지 평균적으로 철회된 횟수이다. 이

3) 한국과학기술원에서 개발한 DBMS인 Odysseus를 사용하여 간단한 트랜잭션을 실험한 결과 CPU 연산과 디스크 I/O에 소요되는 시간의 비는 약 1:1정도이다. 본 논문에서는 보다 복잡한 장기 트랜잭션의 경우를 고려하여 이 비를 2:1로 설정하였다. 1:1과 1:2인 경우도 실험을 수행한 결과 그 추세는 유사하게 나타났으나 본 논문에서는 생략한다.

값이 낮을수록 철회가 적게 발생한 것을 의미한다. 트랜잭션의 응답시간은 어떤 트랜잭션이 처음으로 시작되어 완료할 때까지 걸린 시간으로서 철회가 많거나 로크 충돌로인한 대기 시간이 길수록 응답시간이 길어진다. 단위시간당 트랜잭션 처리율은 시스템의 처리량(throughput)을 나타내는 지표로서 이 값이 클수록 높은 성능을 의미한다.

본 실험에서는 사용하는 로크상승 기법에 따라서 활성정지가 발생할 수 있다. 활성정지는 그 발생 여부를 판정하는 것이 매우 어렵다. 그러므로 본 실험에서는 1000회의 트랜잭션 철회가 발생할 동안 1회의 트랜잭션 완료도 발생하지 않은 경우에 활성정지로 간주하였다. 이러한 방법은 활성정지를 정확하게 판정할 수는 없다. 그러나 위의 조건이 만족되는 경우에는 단위시간당 트랜잭션 처리율이 거의 0에 가깝게 된다. 이는 실제적으로는 활성정지와 다름없는 상황이라고 할 수 있다.

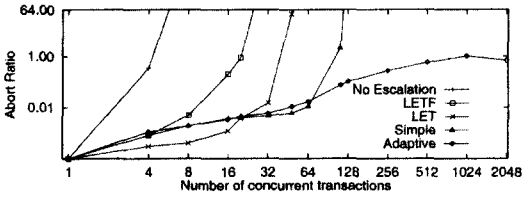
6.2 실험 결과

실험 1. 읽기전용 트랜잭션과 갱신 트랜잭션의 비율을 변화한 경우

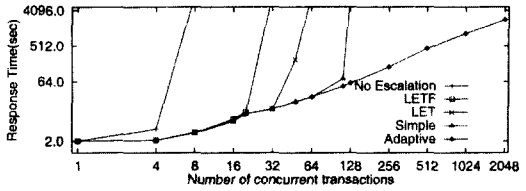
그림 6은 읽기전용 트랜잭션과 갱신 트랜잭션의 비가 8:2인 경우에 각 로크상승 기법들의 성능을 나타낸 것이다. 그림 6의 (a)는 트랜잭션당 평균 철회 횟수를 나타낸 것이고, (b)는 트랜잭션의 평균 응답 시간을 나타낸 것이며, (c)는 단위시간 당 트랜잭션 처리율을 나타낸 것이다.

그림에서 동시에 수행하는 트랜잭션의 갯수가 1인 경우, 즉 트랜잭션들이 순차적으로 수행하는 경우의 단위시간당 트랜잭션 처리율은 0.5이다. 그리고 모든 로크상승 기법들에서 공통적으로 동시에 수행하는 트랜잭션의 갯수가 증가함에 따라, 처음에는 단위시간당 트랜잭션 처리율이 증가하지만 일정 수준에 도달하면 더이상 증가하지 않고 나중에는 오히려 감소하는 것을 볼 수 있다. 동시에 수행하는 트랜잭션의 수가 크게 증가하면서 성능이 감소하는 이유는 다수의 트랜잭션이 동시에 수행됨에 따라 트랜잭션의 로크사용량이 증가하고 로크자원이 부족하게되어 로크자원을 할당받지 못한 트랜잭션들이 연속적으로 철회되기 때문이다. 트랜잭션이 철회되면 재시작하여 완료되어야하므로 자연적으로 응답시간이 길어진다. 또한 트랜잭션이 철회되면서 그 동안 수행한 작업이 무효화되므로, 결과적으로 불필요한 작업을 수행한 것이 되어 단위시간 당 트랜잭션 처리율도 감소한다.

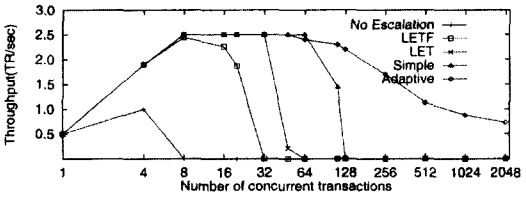
No Escalation은 8개의 트랜잭션이 동시에 수행될 때에 이미 활성정지가 발생하여 트랜잭션의 철회와 응답



(a) 트랜잭션당 평균 철회 횟수



(b) 트랜잭션 평균 응답 시간



(c) 단위시간 당 트랜잭션 처리율

그림 6 읽기전용 트랜잭션과 갱신 트랜잭션의 비가 8:2인 경우에 로크상승 기법들의 성능

시간이 무한대로 증가하고, 단위시간 당 트랜잭션 처리율은 0이 된다. LETF, LET, Simple은 각각 32, 64, 그리고 128개의 트랜잭션이 동시에 수행될 때 활성정지가 발생한다. 반면에, Adaptive는 2048개의 트랜잭션이 동시에 수행되는 경우에도 활성정지가 발생하지 않는다. 특히, 동시에 수행되는 트랜잭션이 매우 많은 경우에는 거의 대부분의 트랜잭션이 로크 충돌에 의해서 대기하거나 로크자원의 부족으로 인하여 대기 상태에 있게되고, 매우 적은 수의 트랜잭션들만이 수행을 하게 된다. 최악의 경우에는 불사 트랜잭션만이 수행된다. 이는 결과적으로는 각 트랜잭션들이 하나씩 순서적으로 수행된 것과 유사하다. 그림에서 2048개의 트랜잭션이 동시에 수행되는 경우의 단위시간당 트랜잭션 처리율(0.73)은 트랜잭션들이 순차적으로 수행된 경우의 처리율과 유사하다는 것을 알 수 있다.

No Escalation의 경우를 제외한 나머지 방법들은 모두 8개의 트랜잭션이 동시에 수행될 때까지는 비슷한 수준의 성능을 보인다. 그러나 타 로크상승 기법들의 경우 동시에 수행하는 트랜잭션이 증가함에 따라 시스템

의 성능이 급격하게 저하되는데 반하여 Adaptive는 매우 완만하게 단계적으로 성능이 저하된다. 그러므로 그림 6의 (c)에서 보는 바와 같이 No Escalation과 비교할 때 256배이상, LET나 Simple과 비교할 때에도 16배 이상의 트랜잭션을 동시에 수행시킬 수 있다. 본 실험에서는 2048개의 트랜잭션을 동시에 수행시키는 수준에서 그쳤다. 그러나 Adaptive는 활성정지가 발생하지 않으므로 이론적으로는 무한대의 트랜잭션을 동시에 수행시킬 수 있다.

LETF가 LET에 비하여 낮은 성능을 보이는 이유는 LETF는 총 로크사용량을 전혀 고려하지 않고 로크상승 임계값에 따라서만 로크상승을 수행하는 반면에, LET는 로크상승 임계값을 넘지 않더라도 로크자원이 부족하면 로크상승을 수행하기 때문이다. 이러한 결과로부터 총 로크사용량을 고려한 로크상승이 성능에 큰 영향을 미치는 것을 알 수 있다.

그림 7은 읽기전용 트랜잭션과 갱신 트랜잭션의 비가 2:8인 경우에 각 로크상승 기법들의 단위시간당 트랜잭션 처리율을 나타낸 것이다. 그림에서 전체적인 추세는 그림 6과 유사하지만 기존의 로크상승 기법들은 동시에 수행하는 트랜잭션의 수가 많아질수록 그림 6에 비하여 급격하게 성능이 저하되어 동시에 수행될 수 있는 트랜잭션의 수가 약 25%정도 감소한다. 이러한 현상이 발생하는 이유는 갱신 트랜잭션은 화일에 대하여 IX 모드의 로크를 획득하므로, 상승불가능 상태의 화일이 증가하여 로크상승을 수행할 수 없는 경우가 많기 때문이다. 반면에, 적응형 로크상승 기법은 유사한 성능을 보인다. 이는 적응형 로크상승 기법이 상승불가능 로크의 갯수에 기반하여 상승불가능 상태의 화일이 증가하는 것을 효과적으로 조절하고 있음을 의미한다.

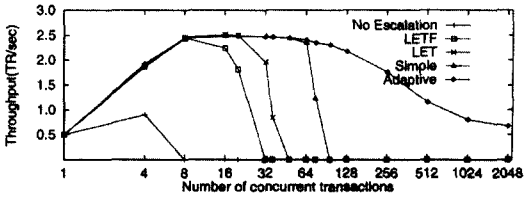
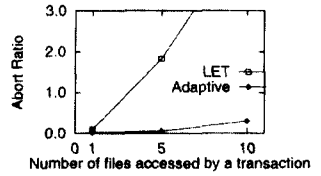


그림 7 읽기전용 트랜잭션과 갱신 트랜잭션의 비가 2:8인 경우에 로크상승 기법들의 단위시간 당 트랜잭션 처리율

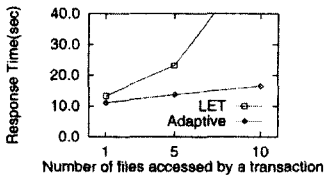
실험 2. 한 트랜잭션이 액세스하는 화일의 갯수가 증가하는 경우

그림 8은 읽기전용 트랜잭션과 갱신 트랜잭션의 비가 8:2이고 32개의 트랜잭션이 동시에 수행될 때 하나의 트랜잭션이 액세스하는 화일의 갯수에 따른 LET와 Adaptive의 성능을 나타낸 것이다. 이 때 각 트랜잭션이 액세스하는 총 레코드의 갯수는 동일하다.

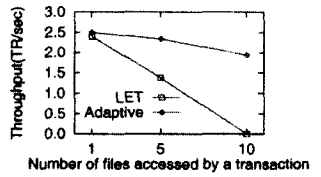
그림에서 LET와 Adaptive 모두 각 트랜잭션이 액세스하는 레코드의 갯수는 동일한데도 불구하고 트랜잭션이 액세스하는 화일의 갯수가 증가함에 따라 트랜잭션의 철회와 응답시간이 증가하고, 단위시간당 트랜잭션 처리율이 감소하는 것을 알 수 있다. 그 이유는 트랜잭션이 액세스하는 화일의 갯수가 증가할수록 IX 모드로 로킹되는 화일의 수가 증가하고, 이에따라 상승불가능 상태의 화일이 증가하여 로크상승을 수행할 수 없는 경우가 많아지기 때문이다. 이 현상은 타 로크상승 기법에서도 유사하게 발생한다. 그러나 Adaptive는 상승불가능 로크의 갯수를 효과적으로 조절하여 성능저하가 매우 작은 것을 볼 수 있다.



(a) 트랜잭션 당 평균 철회 횟수



(b) 트랜잭션 평균 응답 시간



(c) 단위시간 당 트랜잭션 처리율

그림 8 읽기전용 트랜잭션과 갱신 트랜잭션의 비가 8:2이고 32개의 트랜잭션이 동시에 수행되는 경우에 액세스하는 화일의 갯수에 따른 LET와 Adaptive의 성능

실험 3. 화일의 크기가 편중된 경우

그림 9는 읽기전용 트랜잭션과 갱신 트랜잭션의 비가 8:2이고 화일의 크기가 편중된 경우에 각 로크상승 기법의 단위시간당 트랜잭션 처리율을 나타낸 것이다. 데이타베이스에 존재하는 화일의 크기가 서로 다른 경우를 알아보기 위하여 화일에 속하는 레코드 갯수의 분포를 평균이 100000이고 가 0인 Zipf 분포를 따르는 경우에 대하여 실험을 수행하였다. Zipf 분포는 파라미터로 0과 1사이의 값인 를 가지며, 가 1일 때에는 균일 분포이고, 0일 때에는 극단적으로 편중(bias) 분포가 된다. 따라서 앞에서 소개한 실험 1,2,3은 모두 가 1인 경우의 실험에 해당한다.

그림 9는 그림 6(C)와 유사한 결과를 보인다. 적응형 로크상승 기법은 화일 크기의 분포와 관계없이 항상 우수한 성능을 보인다는 것을 알 수 있다. 트랜잭션 당 평균 철회 횟수와 트랜잭션 평균 응답 시간도 유사한 결과를 보이므로 지면 관계상 생략한다.

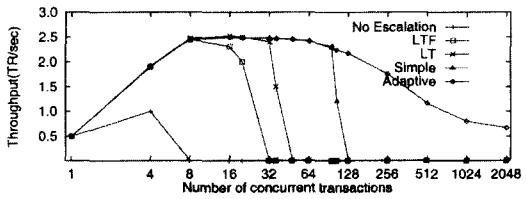


그림 9 적응형 로크상승 기법에서 화일의 크기가 편중되고, 읽기전용 트랜잭션과 갱신 트랜잭션의 비가 8:2일 때 단위시간 당 처리율

실험 4. 디스크 큐에서의 대기를 고려한 경우

그림 10은 읽기전용 트랜잭션과 갱신 트랜잭션의 비가 8:2이고, 한 디스크에 20개의 화일이 저장되어 총 다섯개의 디스크에 데이타가 분산되어 저장된 경우에 디스크 큐에서의 대기까지 고려할 때 각 로크상승 기법의 단위시간당 트랜잭션 처리율을 나타낸 것이다. 디스크가 한 개인 경우와 열 개인 경우에 대한 실험에서도 유사한 추세가 나타났다. 본 논문에서는 다섯 개의 디스크를 사용한 경우에 대해서만 설명한다. 본 실험에서는 디스크 액세스에 대한 요청이 FCFS 방식으로 처리되는 것으로 가정하였다.

그림 10에서 디스크 큐에서의 대기 시간이 길어지기 때문에 전체적인 성능은 실험 1에 비하여 낮게 나타난다. 실험 1에서는 단위시간당 트랜잭션 처리율의 최고값이 2.5였으나, 본 실험에서는 1.9로 약 25%정도 감소하였다. 그러나 전체적인 추세는 그림 6과 유사하여 적응형 로크상승 기법이 디스크 큐에서의 대기와 관계없이 항상 우수한 성능을 보인다는 것을 알 수 있다. 트랜잭션 당 평균 철회 횟수와 트랜잭션 평균 응답 시간도 유사한 결과를 보인다.

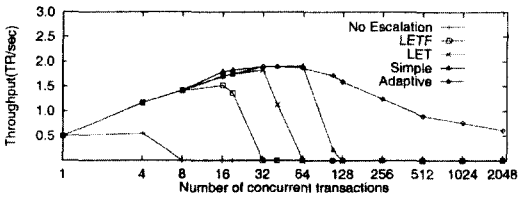


그림 10 읽기전용 트랜잭션과 갱신 트랜잭션의 비가 8:2이고, 디스크 큐에서의 대기를 고려했을 때 단위시간 당 처리율

7. 결 론

기존의 로크상승 기법은 로크상승 여부를 트랜잭션별로 결정한다. 그로 인하여 불필요하게 로크상승을 수행하여 동시성을 저하시키거나 반드시 로크상승이 수행되어야 하는 경우에도 로크상승을 수행하지 못하고 트랜잭션을 철회시키는 문제가 있다. 또한, 매우 많은 로크 요청이 발생할 때, 로크자원의 부족으로 인하여 트랜잭션이 반복적으로 철회되어 DBMS가 마비되는 문제가 있다. 본 논문에서는 이러한 문제를 체계적인 방법으로 해결하였으며, 그 공헌은 다음과 같다.

첫째, 로크상승에 대한 모델을 제안하였다. 기존의 연구에서는 로크상승에 대한 명확한 모델이 없이 로크상승을 단순히 로크자원을 반환받는 방법으로 사용하였다. 본 논문에서는 로크상승에 대한 모델을 정립하여 로크상승의 역할을 규명하고 보다 체계적으로 문제를 해결하였다. 제안한 모델 하에서 로크자원의 부족으로 트랜잭션의 철회를 발생시키는 주 원인인 상승불가능 로크의 개념을 제시하고, 상승불가능 로크의 증가원인을 규명하였다.

둘째, 상승불가능 로크의 증가원인을 해결하는 방법으로서 준로크상승, 로크블로킹, 그리고 선택적 강제수행의 개념을 제안하였다. 그리고 이를 이용한 적응형 로크상승 기법을 제안하였다.

셋째, 실험을 통하여 적응형 로크상승 기법이 기존의 로크상승 기법에 비하여 성능이 매우 우수함을 입증하였다. 적응형 로크상승 기법은 트랜잭션 철회와 평균 응답시간을 감소시키는 동시에 단위시간당 트랜잭션 처리율을 향상시킨다. 또한, 과다한 로크 요청이 발생하는 경우에 시스템의 성능이 급격하게 감소하는 기존의 로

크상승 기법들과는 달리 시스템의 성능을 단계적으로 저하시켜 트랜잭션들이 순차적으로 수행되도록 유도한다. 그리하여 결과적으로 동시에 수행할 수 있는 트랜잭션의 수가 16배에서 256배이상 증가하는 것을 보였다.

본 논문은 모호하게 인식되던 로크자원 관리 측면에서의 로크상승의 역할을 규명하고 상세한 작동원리를 명확히했다는 점에 있어서 커다란 의의가 있다. 기존의 로크상승 기법들은 과다한 로크 요청이 발생할 때의 문제를 사용자 또는 시스템 관리자의 책임으로 처리한다. 반면에, 적응형 로크상승 기법은 이를 자동적으로 조절하여, 시스템의 성능이 급격하게 변화하지 않고 활성정지가 발생하지 않으므로 사용자의 부담을 크게 줄여 준다.

참 고 문 헌

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

[2] IBM, IBM DB2 Universal Database Administration Guide, Version 6, ftp://ftp.software.ibm.com/ps/products/db2/info/vr6/htm/db2d0/index.htm, 2000.

[3] Bernstein, P. and Schkolnick, M., Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987.

[4] Gray, J., Lorie, R., and Putzolu, G., Granularity of Locks in a Shared Data Base, In Proc. Int'l Conf. on Very Large Data Bases, Boston, pp. 428-451, Sept. 1975.

[4] UniSQL, Database Administration Guide (All Products), 1996.

[6] Bell, D. and Grimson, J., Distributed Database Systems, Addison-Wesley, 1992.

[7] Papadimitriou, C., The Theory of Database Concurrency Control, Computer Science Press, 1986.

[8] Ries, D, R. and Stonbraker, M. R, Locking Granularity Revisited, ACM Trans. on Database Systems, Vol. 4, No. 2, pp. 210-227, 1979.

[9] Kohler, W., Wilner, K., and Stankovic, J., An Experimental Comparision of Locking Policies in a Testbed Database System, In Proc. Int'l Conf. on Management of Data, ACM SIGMOD, San Jose, California, pp. 108-119, May, 1983.

[10] Ries, D, R. and Stonbraker, M. R, Effects of Locking Granularity in a Database Management System, ACM Trans. on Database Systems, Vol. 2, No. 3, pp. 233-246, 1977.

[11] Korth, H., Deadlock Freedom Using Edge Locks, ACM Trans. on Database Systems, Vol. 7, No. 4, pp. 632-652, 1982.

[12] Carey, M., Granularity Hierarchies in Concurrency Control, In Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 156-165, 1983.

[13] Eswaran, K. P. et al., The Notion of Consistency and Predicate Locks in a Database System, Comm. of the ACM, Vol. 19, No. 11, pp. 624-633, Nov. 1976.

장 지 웅

정보과학회논문지 : 데이타베이스
제 28 권 제 3 호 참조

이 영 구

정보과학회논문지 : 데이타베이스
제 28 권 제 2 호 참조

황 규 영

정보과학회논문지 : 데이타베이스
제 28 권 제 1 호 참조

양 재 헌

1985년 서울대학교 전산학과(학사). 1987년 서울대학교 전산학과(석사). 1987년 ~ 1989년 한국전기통신공사 사업지원단 전임연구원. 1989 년 1994 University of Maryland at College Park 전산학과(박사). 1994년 ~ 1996 Mills College Oakland, CA, USA, 조교수. 1996년 ~ 현재 한국과학기술원 조교수. 관심분야 OS, Distributed System, Network Virtual Environment, Networking, Distributed Object 입니다.