선택모델을 이용한 동시성제어 알고리즘 비교 분석

양 기 철†

요 약

본 논문에서는 여러 가지 동시성제어 알고리즘들의 비교기준을 설정하는데 어떠한 항목들이 필요한지 알아보고 선택모델을 작성한다. 또한 작성된 선택모델을 이용하여 동시성제어 알고리즘들을 비교 분석한다. 분석 결과는 동시성제어 알고리즘이 필요한 시스템 개발시 사용자 환경에 맞는 최선의 알고리즘을 선택 할 수 있도록 하는데 크게 기여할 것이다.

Comparison Analysis of Concurrency Control Algorithms by using Selection Models

Gi-Chul Yang†

ABSTRACT

Comparison criteria and a selection model for concurrency control algorithms has been presented in this article. In addition, a comparison analysis has been performed with the developed comparison model. The result of the analysis can be utilized to select the best fitting concurrency control algorithm to the user's existing system environment.

키워드 : 동시성제어(Concurrency Control), 선택모델(Selection Model)

1. 서 론

동시성제어는 공유데이터를 사용하거나 병행 처리되는 프로세스들을 서로 간섭받지 않고 운영되도록 관리하는 작업이다. 대부분 두 개 이상의 트랜잭션이 동시에 작동하면 이들은 서로에게 영향을 주게된다. 예를 들어, 하나의 프로그램이 끝나기 전에 다른 프로그램이 중간에 공유데이터를 조작하게 되면 처음 프로그램은 오류 없이 동작하였더라도 틀린 결과를 산출할 것이다. 이러한 잘못된 결과의 도출을 방지하는 작업이 동시성제어이다. 이렇듯 동시성제어는 자원을 공유하는 모든 시스템에서 필요한 중요한 작업이다.

동시성제어는 Locking, 상호배제등 여러 가지 방법을 이용하여 수행된다. 이들 각각의 방식들도 중앙집중식 또는 분산방식으로 나뉘게 된다. 이렇듯 다양한 동시성제어 알고리즘이 존재하는데 이들은 각각 장단점을 가지고 있다. 따라서 어떤 경우에 어떤 알고리즘이 최적의 알고리즘인지 쉽게 판단하기 어렵다. 그러므로 시스템 환경에 맞는 최선의 동시성제어 알고리즘을 선택할 수 있도록 하기 위해서 동시성제어 알고리즘의 총체적인 비교연구는 중요한 연구과제중의 하나이다.

본 논문에서는 여러 가지 동시성제어 알고리즘들의 비교기준을 설정하는데 어떠한 항목들이 필요한지 알아보고 이를 이용하여 동시성제어 알고리즘들의 선택모델을 만든다. 이는 시스템 개발시 최선의 동시성제어 알고리즘을 선택 할 수 있도록 하는데 크게 기여할 것이다. 다음 장에서는 동시성제어 알고리즘 관련 연구를 살펴보고 3장에서 동시성제어 알고리즘들의 비교기준을 설정하는데 필요한 항목들을 설정한다. 4장에서는 동시성제어 알고리즘 선택모델을 만들고 5장에서는 선택모델을 이용하여 알고리즘들을 비교 분석하고 6장에서 결론을 맺는다.

2. 관련 연구

동시성제어는 1960년대 중반 Ben-Ari, Brinch Hansen, Holt등이 운영체제 분야에서 연구하기 시작하였다. Serializability는 Bernstein, Gray, Papadimitriou등에 의해 연구되었고 Two Phase Locking(2PL)은 1976년 Eswaran에 의해 소개되었다. 그후 Bernstein, Eswaran, Papadimitriou등에 의해 2PL의 정확성이 증명되었다. 분산 시스템에서의 Locking의 효율에 관해서 Ries, Stonebraker, Thanos, Garcia-Molina등이 연구하였으며 특히 Ries와 Stonebraker는 계층

※ 본 연구는 정보통신부 대학기초 연구지원 사업의 지원을 받았음.
† 종신회원 : 목포대학교 정보공학부 교수
논문접수 : 2002년 8월 16일, 심사완료 : 2003년 3월 20일

적 locking도 연구하였다[1].

Locking에 의한 동시성 제어는 Two Phase Locking(2PL)이 대표적인데 2PL은 공용데이터를 충돌 없이 사용하고자 할 때 사용하는 간단한 방법으로 각 데이터 항목에 lock이 있어 lock이 걸려있지 않은 데이터는 바로 사용할 수 있으나 lock이 걸려있는 데이터는 lock이 풀리기 전에는 사용할 수 없도록 한다. 따라서 하나의 트랜잭션이 어떤 데이터를 사용하면 다른 트랜잭션은 그 데이터를 사용할 수 없게 할 수 있어 트랜젝션의 선형순서화를 가능하게 한다.

이러한 2PL 규칙을 따르면 트랜젝션들을 선형순서화 하여 수행시킬 수 있으나 Deadlock의 위험이 있다는 것이 단점이다. 이러한 Deadlock을 다루는 방법으로 임계시간(Time out)을 설정하는 방법이 있다. 즉 어떤 트랜젝션이 너무 오래 기다리면 Deadlock이 발생했다고 가정하고 그 트랜젝션을 abort 시키는 것이다. 이는 틀린 결과를 내는 오류를 발생시키지는 않지만 실행 효율을 떨어뜨릴 수 있다.

여기서 임계시간을 길게 잡으면 abort 시키는 트랜젝션의 수는 줄일 수 있지만 Deadlock을 발견하는데 더 오랜 시간이 걸린다. 또 다른 방법은 Waits-for-graph(WFG)를 사용하는 것이다. 이는 트랜젝션간에 lock의 기다림 관계를 그래프로 표현하고 이 그래프에 cycle이 있을 경우 deadlock이 발생했음을 알아낸다. 이 경우에는 cycle을 얼마나 자주 검사해야되는가가 문제이다. 자주 검사하면 deadlock 은 빨리 발견할 수 있으나 그만큼 비용이 많이 들고 반대로 검사를 자주하지 않으면 deadlock이 발생해도 장시간 발견할 수 없게된다

트랜젝션의 abort는 deadlock 때문에 발생한다. Deadlock의 발생을 막기 위해서 모든 트랜젝션이 필요로 하는 lock을 모두 얻은 다음에 연산을 수행하는 방법이 보수적 2PL (Conservative 2PL) 방법이다. 따라서 이 경우에는 deadlock이 발생하지 않아 트랜젝션을 abort할 필요도 없다. Strict 2PL은 트랜젝션이 끝날때 그 트랜젝션이 걸었던 lock을 동시에 푼다. Strict 2PL은 복원가능(Recoverable)하고 단계적 abort(Cascade abort)를 피할 수 있다.

Locking을 사용하지 않는 방법중의 하나로 Timestamp Ordering(TO)을 사용하는 방법이 이 있는데 이는 Shapiro, Thomas등에 의해 1977년과 1979년 각각 연구되었다. TO 기법은 각 연산에 붙어있는 timestamp를 이용하여 충돌을 피한다. 따라서 만약 연산 Pi[χ]와 Pj[χ]가 충돌할 경우 각각의 timestamp ts(Ti)와 ts(Tj)를 비교하여 빠른 것을 먼저 수행한다. Timestamp를 사용하면 timestamp를 기록할 공간이 추가로 필요하게되어 기억장치를 많이 사용하게 된다. 데이터 항목의 크기가 작을 경우에는 더 많은 기억장치 공간을 필요로 한다.

한편 종료하지 않는 프로그램(예 : 시스템 관리 프로그램)은 순차수행이 적용되지 않는다. 따라서 이러한 프로그램에 대한 동시성제어는 상호배제의 생성을 통하여 이루어진다. 상호배제 생성 알고리즘도 중앙집중식과 분산 방식으로 구분되고 각각의장단점이 있다. 첫 번째 방법은 하나의 노드가 모든 권한을 갖고 상호배제를 이루어내는 방법이고 두 번째 방법은 하나의 분산 시스템내에 속해있는 모든 노드들이 다 함께 상호배제를 이루는데 참여하는 방법이다.

위에서 말하는 첫 번째 방법은 간단하기는 하나 커다란 단점이 있다. 상호배제를 위한 모든 권한을 하나의 노드가 갖고 있어서 이 노드에 고장이 발생할 경우 시스템 전체가 멈추어야 하며 이러한 경우를 대비하여 또 다른 노드에 권한을 이양할 수 있는 별도의 조치가 마련되어야 한다. 따라서 이러한 별도 조치까지 고려한다면 매우 복잡한 방법이 된다. 두 번째 방법은 어떤 노드에 고장이 발생하더라도 별도의 조치가 필요 없이 나머지 노드들이 상호배제를 이루어 낼 수 있다.

분산방식 상호배제생성 알고리즘도 크게 두 가지로 나눌 수 있다. 첫째는 Time Stamp를 사용한 방법으로 Lamport가 개발한 알고리즘이다[4]. Lamport의 알고리즘은 N개의 노드를 갖는 네트워크에서 $3\times(N-1)$개의 메시지를 사용하는 분산방식 상호배제생성 알고리즘이다. 이는 Logical Clock을 사용하여 노드간의 상호배제를 이루었으나 노드고장은 고려하지 않았다.

Ricart & Agrawala의 알고리즘도 Sequence number와 Node number를 Logical Clock으로 사용한다[10]. 노드 중 Critical Section에 들기를 원하는 노드는 Request 메시지를 다른 모든 노드에 보내고 Request 메시지를 받은 노드는 Reply 메시지를 보낸다. Request 메시지를 받았지만 Critical Section에 들어가고 싶으면 받은 Request 메시지에 있는 Logical Clock과 자신의 Request 메시지에 있는 Logical Clock을 비교하여 우선순위를 결정한다. 이렇게 하면 $2\times(N-1)$개의 메시지를 사용하여 노드 간의 상호배제를 이룰 수 있다.

Maekawa는 Request 메시지를 다른 모든 노드에 보내지 않고 특정그룹에 속한 노드에게만 보냄으로서 상호배제 생성 시 필요한 메시지 숫자를 줄였다[8]. 이렇게 하여 N개의 노드를 갖는 네트워크에서 $C\sqrt{N}$개의 메시지로 상호배제를 이룰 수 있게 하였다(이때 $3 \le C \le 5$). 하지만 특정노드 그룹의 생성이 쉽지 않고 노드 고장시 더욱 복잡한 알고리즘이 필요하다.

Raymond는 전체 시스템을 트리 형태로 구성하여 Log N개의 메시지로 상호배제를 생성할 수 있도록 하였다[14]. 하지만 하나의 노드는 두 개 이상의 Request 메시지를 보낼 수 없으며 두 개의 서로 인접한 노드가 동시에 고장일 경우 대책이 없다.

둘째는 토큰을 사용한 방법인데 토큰을 사용하는 방법은 전체 시스템 내에 유일한 토큰이 존재하고 그 토큰을 소유한 노드는 Critical Section에 들어갈 수 있다. Suzuki & Kasami, yang, Kumar & Place, Singhal 등이 Token을 사

용한 알고리즘을 개발한바 있다.

Suzuki & Kasami는 각 노드에 크기가 N인 Array RN을 보유하고 가장 큰 Sequence number를 저장하도록 하고 새로운 Request (j, n) 메시지가 들어오면 우선순위 결정을 위하여 RN[j] := max(RN[j], n)으로 변경한다[13]. 그리고 PRIVILEGE 메세지를 받으면 그 노드는 Critical Section에 들어갈 수 있다.

Yang, Kumar & Place는 Request 메시지를 우선순위에 따라 저장하는 하나의 Queue를 토큰으로 사용하여 그 Queue를 갖는 노드가 Critical Section에 들어갈 수 있도록 하였다[3]. 이 알고리즘의 특징은 Request 메시지는 하나만 보내도 되고 각 노드에서 보내진 Request는 하나의 Queue에 저장된다는 것이다. 이렇게 하면 적은 메시지만 가지고도 상호배제를 이루어 낼 수 있다. 이때 네트워크상에 어떤 노드에 그 Queue가 있는지를 알아야 하며 이는 각 노드에 있는 Local State에 있는 기록을 이용한다.

3. 비교항목 설정

본 절에서는 기존의 연구들에서 얻은 동시성제어 알고리즘 관련정보들을 알아보고 비교연구에 필요한 항목이 무엇인지 알아보기로 한다. Munz와 Krenz는 다음과 같은 사실을 알아냈다. 2PL에서 lock의 범위(granularity)가 작을수록 트랜젝션의 대기시간이 줄어들어 동시에 수행될 수 있는 트랜젝션의 수는 증가한다. 한편 lock의 범위(granularity)가 커지면 deadlock이 줄어든다. 또한 deadlock은 프로세스의 수와 데이터 항목(lockable unit)의 수가 비슷할 때 가장 빈번히 일어남을 알 수 있다[9]. Ries와 Stonebreaker는 lock의 범위(granularity)가 작아지면 동시성은 증가하지만 lock을 관리해야되는 일이 많아져 시스템의 성능은 떨어진다는 사실을 알아냈다[11, 12]. Lin과 Nolte는[5]에서 [11, 12]의 연구를 확장하여 2PL에서 퀴리와 갱신기의 성능을 연구하였다. 연구결과 read와 write의 빈도는 lock의 충돌 확률과 deadlock발생에 거의 영향을 주지 않는 것으로 나타났다. 또한 read와 write의 빈도는 갱신기의 응답 시간에도 별로 영향을 미치지 않고 트랜젝션의 평균 크기가 작으면 트랜젝션의 응답시간이 약간 개선됨을 알았다.

또한, [5]에서 평균 트랜젝션 크기가 작을 때 timestamp를 사용한 방식이 2PL 방식보다 우수한 성능을 나타내는 것을 보았다. 하지만 평균 트랜젝션 크기가 클 경우에는 반대로 2PL 방식이 timestamp를 사용한 방식보다 성능이 좋았다. 또한 트랜젝션의 크기가 작으면 충돌이 일어날 확률이 작아지고 충돌은 큰 트랜젝션과 더 많이 일어남을 알았다. 트랜젝션의 크기와 관련한 결론적인 사실은 크기가 큰 트랜젝션은 abort 시키는 것보다 차례를 기다리는 것이 더 좋은 결과를 기대할 수 있다는 것이다.

Irani와 Lin은 [2]에서 lock을 사용한 동시성 서버에서 데이터 항목의 크기가 클수록 locking에 대한 부담이 줄어 결과적으로 성능이 우수한 것으로 나타났다. [15]에서는 큐잉 네트워크 모델을 사용하여 lock의 충돌과 deadlock 이 시스템의 성능에 미치는 영향을 분석하였는데 시스템의 응답시간은 deadlock보다는 lock의 충돌에 더 많은 영향을 받는 것으로 나타났다.

이러한 연구 결과에 기반 하여 동시성제어 알고리즘들을 비교할 때 고려하여야될 사항을 알아보면 다음과 같다.

① Lock의 범위(granularity)
② 응답시간과 Throughput간의 관계
③ 서로 다른 시스템 부하에서 자원활용도
④ 동시 수행율과 Throughput간의 관계
⑤ Deadlock 발생비율 및 deadlock 진단에 필요한 일의 양
⑥ 트랜젝션의 크기
⑦ 트랜젝션에서의 read와 write 비율
⑧ 다단계 Roll-Back
⑨ 트랜젝션 대기시간과 Throughput간의 관계
⑩ i/o 처리 시간
⑪ CPU 사용 시간
⑫ CPU Locking 처리 시간
⑬ CPU 복원 시간 (Recovery)
⑭ 트랜젝션 수
⑮ DB 크기
⑯ 평균 트랜젝션 Blocking 수
⑰ Roll-Back 트랜젝션 수
⑱ 평균 대기시간
⑲ 동시수행 트랜젝션의 평균 수
⑳ 평균 Deadlock 수
㉑ 시스템 Throughput
㉒ 사용되는 메시지 수(상호배제 사용시)

위에서 ①번부터 ⑨번까지는 사용자 입장에서 고려 하여야할 사항이고 ⑩번부터 ⑬번까지는 시스템 관련사항 그리고 ⑭번 ⑮번은 시스템의 부하가 얼마나 큰가와 관련된 내용 그리고 ⑯번부터 ㉑번 까지는 출력에 관한 사항이다. ㉒번은 분산방식 상호배제 사용시 중요한 고려 대상이다.

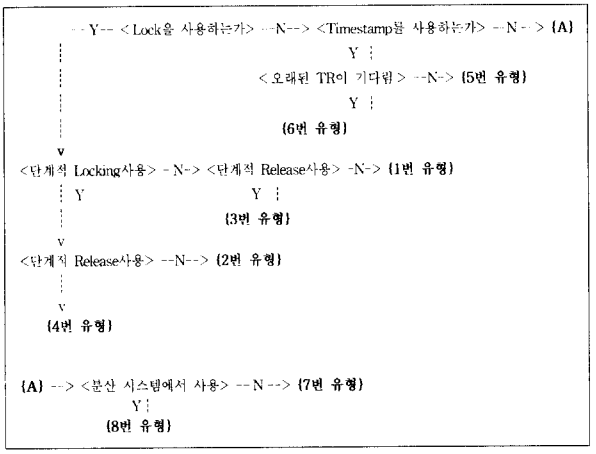
4. 동시성 제어 알고리즘 선택모델

본 절에서는 앞 절의 내용을 참고하여 동시성 알고리즘 선택모델을 설계한다. 우선 앞 절에서 살펴본 여러 가지 고려사항 중에서 중요한 것들(시스템의 성능에 영향을 많이 미치는 요소들)을 추출하고 이들을 사용한 선택모델을 작성하도록 한다. 중요한 고려사항은 트랜젝션의 Granunality(lock의 범위), 트랜제션의 평균적 인 크기, 응답시간, Throughput 등을

들 수 있다. 이들에 관련된 내용을 정리하면 다음과 같다.

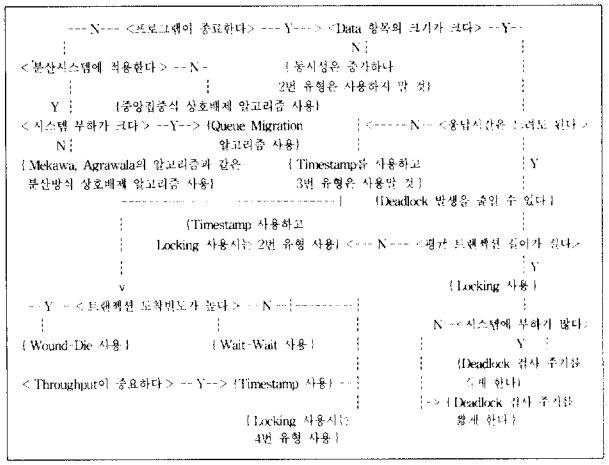
트랜젝션의 Granunality 크기가 크면 응답시간이 느려지고 Granunality 크기가 작으면 동시성은 증가하나 시스템 성능이 떨어짐으로 Lock을 단계적으로 걸고 동시에 푸는 방법 사용하지 말아야 한다. 평균 트랜젝션의 크기가 크면 Locking 방법 사용이 좋고 작으면 Timestamp 방법 사용하고 Locking 방법 사용시는 Lock을 동시에 걸고 풀때는 단계적으로 푸는 방식을 사용하는 것이 좋다. 응답시간이 중요한 경우는 Timestamp 방식이 Locking 보다 우수하나 Locking 방식 사용시는 Lock의 크기를 줄이고 단계적으로 Lock을 걸고 동시에 푸는 방식을 사용하지 않는 것이 좋다. 한편 응답시간이 중요하지 않는 경우에는 트랜제션의 크기를 크게 하여 Deadlock 발생을 줄이는 것이 좋다. 트랜젝션의 도착빈도가 높으면 Wait-Die 방식을 사용하는 것이 효과적이고 트랜젝션의 도착빈도가 낮으면Wound-Wait 방식을 사용하는 것이 효과적이다. 또한, Throughput이 중요하면 Time stamp 방식을 사용하고 Throughput이 중요하면서 트랜젝션의 도착 빈도가 낮을때는 Locking 방식 사용도 가능하나 이때는 단계적으로 Lock을 걸고 풀때도 단계적으로 푸는 방식을 사용하는 것이 좋다.

위의 내용을 고려한 선택모델은 다음과 같이 구성 할 수 있다.



알고리즘 유형별 상황 선택을 위한 선택모델

- **1번 유형** : Lock을 동시에 걸고 동시에 Release 하는 Two-Phase Locking 방식으로 Data contention이 적을 때 비교적 좋은 효율을 내는 알고리즘이다.
- **2번 유형** : Lock을 동시에 걸고 단계적으로 Release 하는 Two-Phase Locking 방식으로 트랜젝션 길이가 짧을 때 사용하면 좋다. 데이터 항목의 크기가 작을 때는 사용하지 말아야 한다. 대체적으로 효율이 떨어지는 방식이다.
- **3번 유형** : Lock을 단계적으로 걸고 동시에 Release 하는 Two-Phase Locking 방식으로 빠른 응답 시간을 요구하는 시스템 개발시에는 사용하지 않는 것이 좋다.
- **4번 유형** : Lock을 단계적으로 걸고 단계적으로 Release 하는 Two-Phase Locking 방식으로 트랜젝션의 도착빈도가 높고 Throughput이 주요할 때 사용하면 좋다.
- **5번 유형** : Wound-Wait 방식으로 Roll-Back을 가장 많이 사용하는 방식이며 트랜젝션의 도착 빈도가 Exponential일때만 2번 유형 보다 효율이 좋다.
- **6번 유형** : Wait-Die 방식으로 4번 유형보다 대기시간이 적고 응답시간이 빠르다.
- **7번 유형** : 중앙집중식 상호배제 생성 알고리즘으로 종료하지 않는 프로그램에서 동시성제어를 위해서 필요한 알고리즘이다. 구현이 용이하나 중앙 노드의 고장시 시스템 전체의 작동이 멈추게 된다.
- **8번 유형** : 분산방식 상호배제 생성 알고리즘으로 종료하지 않는 프로그램에서 동시성제어를 위해서 필요한 알고리즘이다. 분산시스템 개발시 사용된다.



사용자(외부환경) 상황별 알고리즘 선택을 위한 선택모델

위의 선택모델을 사용하면 알고리즘 유형에 따른 최적의 적용 상황이나 외부환경의 상황별 최적 알고리즘 선택시 큰 도움이 될 것이다. 보통의 경우 트랜젝션의 평균 길이는 짧고 데이터 항목의 크기는 작으며 Timestamp를 사용하는 것이 Throughput을 높일 수 있다.

5. 동시성 제어 알고리즘 비교 분석

본 장에서는 지금까지 살펴본 내용을 바탕으로, 여러 가지 동시성 제어 알고리즘 유형을 세분하여 비교 분석하여 보기로 한다. 동시성 제어 알고리즘은 Lock을 사용하는 것, Timestamp를 사용하는 것으로 나누어 비교 분석한다.

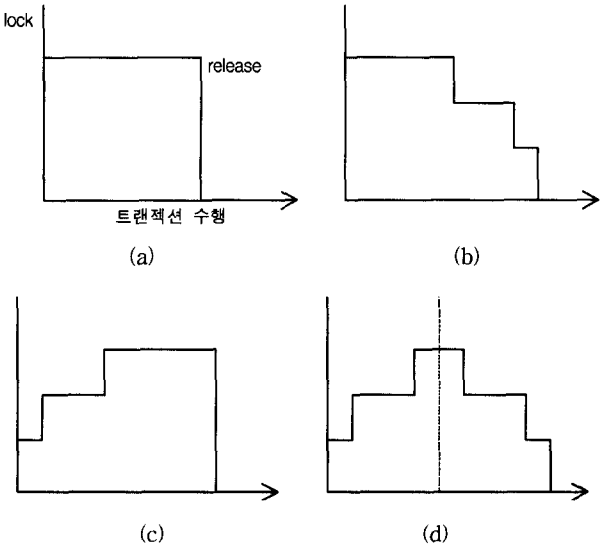
5.1 Lock을 이용한 방법

Two-Phase Locking(2PL)은 4가지 유형의 알고리즘으로 다시 세분할 수 있는데 이는 Lock을 걸때와 풀때 동시에 하

는지 단계적으로 하는지에 따라 다르다.

- 2PL의 4가지 유형은 다음과 같다.
  ① 필요한 모든 lock을 동시에 걸고 동시에 푸는형
  ② 필요한 모든 lock을 동시에 걸지만 풀때는 단계적으로 하는형
  ③ 필요한 모든 lock을 단계적으로 거고 동시에 푸는형
  ④ 필요한 모둔 lock을 단계적으로 걸고 단계적으로 푸는형

이들은 다음과 같이 그림으로 나타 낼수 있다.



위에서 보듯이 (a)번과 (b)번 유형은 lock을 동시에 걸기 때문에 (c)번과 (d)번 유형에 비하여 더 많은 트랜젝선이 충돌할 수 있다. 또한 (a)번 (b)번 유형이 (c)번 (d)번 유형보다 lock을 가지고 있는 시간이 더 길다. (b)번과 (c)번의 경우는 어느 것이 lock을 더 오래 가지고 있는지 단정 하기는 어렵다. 하지만 (a)번 유형이 lock을 가지고 있는 시간은 가장 길고 (d)번 유형이 lock을 가지고 있는 시간이 가장 짧다는 것은 분명하다. Lock을 가지고 있는 시간이 짧으면 시스템의 throughput이 증가하는 것은 쉽게 알 수 있다. 이는 [4, 5]번에서도 언급되었다.

Deadlock의 경우 (c)번과 (d)번 유형에서 발생 가능하다. Deadlock이 자주 발생하지 않는 경우는 (c)번 유형이 (a)번이나 (b)번 유형보다 성능이 좋다. 단계적인 roll-back 이 적으면 (d)번 유형이 가장 효율이 좋다. [10]과 [11]에서는 blocking이 roll-back보다 더 throughput에 영향을 크게 미친다 는 것을 알았다. 트랜젝션의 도착 빈도가 높은 경우 (d)번 유형의 응답시간이 가장 작다. 응답시간은 (b)번 유형이 (a)번 유형보다 크다. 트랜젝션들의 크기가 대부분 적을 때 (b)번 유형이 가장 좋은 성능을 보인다. Deadlock의 발생 확률은 (d)번 유형보다 (c)번 유형이 높다. Deadlock의 발견을 위한 검사는 자주 하지 않는 것이 좋다.

응답시간과 단계적 roll-back의 관계를 알아보면 다음과 같다. 시스템 내에서 트랜젝션의 동시수행률이 높아지면 through put도 어느 정도 까지는 높아지나 어느 수준에 도달하면 오히려 떨어진다. 따라서 트랜젝션의 동시수행률은 시스템 성능 측정시 중요한 변수는 못된다. read와 write의 빈도 역시 시스템의 성능에 큰 영향은 미치지 않는다. 결론 적으로 locking을 사용하는 방법에서 모든 조건을 만족하는 알고리즘은 없지만 일반적으로 (d)번 유형이 우수한 것을 알 수 있다. 트랜젝션의 도착 빈도가 높으면 (a)번 유형이 (b)번이나 (c)번 유형보다 우수하고 시스템의 성능은 roll-back보다 blocking에 더 많은 영향을 받는다. (c)번 유형이 (d)번 유형보다 구현하기가 쉬워 더 자주 사용되고 있다.

5.2 Timestamp를 이용한 방법

Timestamp를 이용하는 방식에서는 트랜젝션은 blocking이 아닌 roll-back에 의해 충돌을 해결한다. 따라서 blocking이 시스템의 throughgput이나 응답시간에 중대한 영향을 미친다면 timestamp 방식이 lock을 사용하는 방식보다 효율적이라고 생각할 수 있다. [10, 11]의 연구결과에 의하면 blocking이 roll-back보다 더 큰 영향을 시스템에 미친다고 하였다.

Wound-Wait(WW) 알고리즘과 **ait-Die(WD)** 알고리즘에서는 충돌이 blocking 이나 roll-back 두 가지에 의해 해결될 수 있다. WD 방식에서는 두 트랜젝션의 충돌시 더 오래된 트랜젝션이 기다리고, WW 방식에서는 그 반대이다. 결론적으로 timestamp를 이용한 방식이 다른 방식보다 성능 면에서 약간 우수하고 자원 이용률에서는 WD나 WW보다 효율이 낮다.

트랜젝션의 도착 빈도가 낮으면 응답시간에는 알고리즘간 차이가 거의 없다. 그러나 트랜젝션의 도착 빈도가 높으면 WD의 응답시간이 WW보다 빠르다. Timestamp를 사용하는 알고리즘이나 WW, WD 모두 트랜젝션의 도착 빈도가 높으면 트랜젝션의 도착 분포가 Exponential일 때 throughput이 가장 낮고 트랜젝션의 도착 분포가 Hyper-exponential 일 때 throughput이 가장 높고 roll-back은 가장 낮다. 트랜젝션의 도착 분포에서는 어느 분포이던지 WD가 WW보다 roll-back이 적어 더 성능이 좋음을 알 수 있다. 하지만 roll-back이 많아진다고 해서 꼭 throughput이 적어지는 것은 아니다. 이는 timestamp를 상용하는 알고리즘의 roll-back숫자가 WD보다 많지만 throughput은 TS가 더 높은 것에서도 알 수 있다.

앞의 (d)번 유형은 timestamp를 사용하는 알고리즘보다 응답시간이 더 오래 걸린다. 이는 timestamp를 사용하면 기다리지 않고 roll-back하기 때문이다. 따라서 roll-back을 효율적으로 구현 할 수 있으면 blocking을 사용하는 것보다 좋은 성능을 얻을 수 있다. WD는 (d)번 유형 보다 대기시간이 적고 응답시간이 빠르다. WW의 성능은 WD나 time stamp를 사용하는 알고리즘 또는 (d)번 유형 보다 좋자 않다. 이는

WW가 roll-back을 가장 많이 하기 때문이다. WW와 (c)번 유형을 비교하여 보면 트랜잭션의 도착 분포가 exponential 일 때 만 WW가 좋음을 알 수 있다.

Throughput은 timestamp를 사용하는 알고리즘이나 WD보다 (d)번 유형이 높으나 트랜잭션의 도착 빈도가 아주 높아지면 deadlock이 많아져서 throughput이 급격히 떨어진다. 응답시간은 timestamp를 사용하는 알고리즘이나 WD나 별 차이가 없다.

결론적으로 lock을 사용하는 (b)번 유형이 효율이 가장 떨어지고 실제로 deadlock과 단계적 roll-back은 많이 일어나지 않으므로 시스템의 성능에 큰 영향은 미치지 않지만 트랜잭션의 대기 시간은 시스템의 성능에 크게 영향을 미칠 수 있다는 것을 알 수 있다. Roll-back과 blocking을 비교하여 보면 blocking을 사용하는 것 보다 roll-back을 사용하는 것이 더 효율적이고 트랜잭션의 대기 시간을 줄이면 시스템의 성능을 높일 수 있다는 것을 알 수 있다. 또한 단계적 locking에서 deadlock은 자주 발생하지 않고 blocking이 발생할 때마다 deadlock을 검사할 필요는 없다. 또한 트랜잭션이 전부 read이거나 전부 write가 아니면 read와 write의 비율은 시스템 성능에 크게 영향을 주지 않는다.

6. 결 론

본 논문에서는 여러 가지 동시성제어 알고리즘들의 비교 기준을 설정하는데 어떠한 항목들이 필요한지 알아보고 이를 동시성제어 알고리즘들을 비교하는데 필요한 모델을 설정하였다. 또한 설정된 선택모델을 이용하여 여러 가지 동시성제어 알고리즘들을 비교 분석하였다. 본 논문의 연구 결과는 사용자의 환경에 가장 적합한 동시성제어 알고리즘을 선택하는데 유용하게 활용될 수 있을 것이다.

참 고 문 헌

[1] Berstein P., V. Hadzilacos & N. Goodman, Concurrency Control and Recovery in Database Systems, Addison Wesley, 1987.

[2] Irani, K. B. and Lin, H., "Queuing Network Models for Concurrent Transaction Processing in a Database System," ACM-SIGMOD, pp134-142, May, 1979.

[3] Kumar, Place and Yang, An Efficient Algorithm for Mutual Exclusion Using Queue Migration in Computer Networks, IEEE Trans. on Knowledge and Data Engineering, Vol.3, No.3, September, 1991.

[4] Lamport, L., Time Clocks and the Ordering of Events in a Distributed System, CACM, 21, 7, July, 1978.

[5] Lin, W. K. and Nolte, J., "Read only Transactions and Two-Phase Locking," Proc. of 2nd Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, pp.85-93. July, 1982.

[6] Lin, W. K. and Nolte, J., "Basic Timestamp, Multiple Version Timestamp, and Two-Phase Locking," 9th Int. Conf. on VLDB, Florence, Italy, pp.109-119, October, 1983.

[7] Luk, W. and Wong, T., Two New Quorum Based Algorithms for Distributed Mutual Exclusion, 17th international conference on Distributed Computing Systems, Boltimore, 1997.

[8] Maekawa, M., A SQRT(N) Algorithm for Mutual Exclusion in Decentralized Systems, ACM Trans. on Comp. Sys., 3, 2, May, 1985.

[9] Munz, R. and Krenz, G., "Concurrency in Database Systems - A Simulation Study," Proc. INT. Conference on MOD., pp. 111-120, August, 1977.

[10] Ricart, G. and Agrawala, A., An Optimal Algorithm for Mutual Exclusion in Computer Networks, CACM, 24, 1, January, 1981.

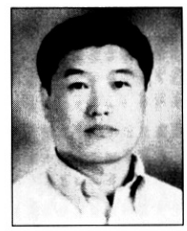
[11] Rise, D. R. and Stonebraker, M., "Effect of Granularity in a Database Management System," ACM TODS, Vol.2, No.3, pp.233-246. September, 1977.

[12] Rise, D. R. and Stonebreaker, M., "Locking Granularity Revisited," ACM Trans on Database Systems, Vol.4, No.2, pp.210-227, June, 1979.

[13] Suzuki, I. and Kasami, T., A Distributed Mutual Exclusion Algorithm, ACM Trans. on Comp. Sys. 3, 4, November, 1985.

[14] Raymond, K., A Tree-Based Algorithm for Distributed Mutual Exclusion, ACM Trans. on Comp. Sys. 7, 1, February, 1989.

[15] Thomasian, A., "An Iterative Solution to the Queuing Network Model of a DBMS with Dynamic Locking," Proc. of the 1982 Computer Measurement Group, San Diago, pp. 252-261, 1982.



양 기 철

e-mail : gcyang@mokpo.ac.kr

1982년 전남대학교 계산통계학과(학사)

1986년 미국 University of Iowa 전산학(석사)

1993년 미국 University of Missouri 전산학(박사)

2001년 영국 Heriot-Watt University 객원교수

1993년~현재 목포대학교 정보공학부 부교수

관심분야 : 인공지능, 자연어처리, 시맨틱 웹, 분산처리 등