Hotspot 테이블을 위한 새로운 공간 비축 방법

(A New Space Reservation Method for Hotspot Tables)

이 강 우 † 김 형 주 ††
(Kang-Woo Lee)(Hyoung-Joo Kim)

요 약 여러 새로운 DBMS 응용들 중에 주요 응용분야인 워크 플로우, 그룹웨어, 주문 처리 응용에서는 다수의 트랜잭션들에 의해 동일 테이블에 대한 레코드 삽입과 삭제 연산이 빈번히 발생한다. 그러나 기존의 공간 비축 문제들은 이러한 환경에 적합하지 않아 그대로 사용하는 경우 성능상의 많은 문제점을 유발하게 된다.

본 논문에서는 로크 테이블에 페이지의 비축 공간 상태를 정확히 추적할 수 있는 추가의 자료를 두어, 기존의 방법보다 정확하고, 적은 부하의 비축 공간 검사를 수행할 수 있는 새로운 공간 비축 방법을 제안하였다. 또한, 시뮬레이션 실험을 통해, 동일 테이블을 동시에 접근하는 트랜잭션의 수가 많은 경우에 제안된 방법이 기존의 방법들보다 매우 우수한 성능을 가짐을 보였다.

Abstract Workflow, groupware and order inventory systems are emerging as new important DBMS applications. In those applications, many concurrent transactions insert and delete records on the same table. Those applications, however, might have several performance problems, since previous space reservation methods do not support this new type of access pattern well

This paper proposes a new space reservation method which extends the lock table that keeps track of the reserved pages and which guarantees correct reserved space check with low checking overhead. This paper also shows the proposed method provides better performance than the previous ones when there are many concurrent insertions and deletions on the table by simulation studies.

1. 연구 동기 및 배경

데이타베이스 시스템(DBMS)의 기능과 성능이 향상됨에 따라 보다 다양한 응용분야에서 DBMS를 사용하게 되었다. 그러나 많은 응용분야의 데이타 접근 형태는 기존의 그것과 상이하여 기존의 DBMS를 그대로 사용하는 경우 성능상의 많은 문제점이 유발된다[2]. 이에 따라 많은 DBMS 연구자들은 이러한 응용분야를 지원하기 위해 노력을 기울여왔다[1, 3, 8, 10].

· 이 연구는 '96년도 통상산업부 공업기반과제 연구비(과제번호:961-28-3)와 정보통신부 산학연 과제 지원에 의한 결과임
† 비 회 원 : 서울대학교 전산과학과
kwlee@candy.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학과 교수
hjk@im4u.snu.ac.kr
논문접수 1997년 11월 27일
심사완료 : 1998년 10월 15일

많은 응용분야에서 여러 트랜잭션들이 특정 테이블에 대해 동시에 삽입과 삭제를 수행하는 경우를 볼 수 있다. 워크플로우(workflow), 그룹웨어(groupware) 및 큐를 이용한 트랜잭션 처리가 그러한 데이타베이스 응용분야의 예이다. 워크플로우나 그룹웨어의 경우는 하나의 장시간 작업 과정을 여러 작은 트랜잭션의 네트워크 연결로 구성하게 되는데, 이때 트랜잭션들 사이의 데이타 교환은 큐(queue) 또는 유사한 역할을 하는 테이블에 의해 구현된다. 이러한 큐 또는 테이블에서 빈번한 데이타 접근 형태는 레코드의 갱신보다는 레코드의 삽입과 삭제 연산이다. 즉, 한 트랜잭션이 다음 트랜잭션에 전달할 데이타를 큐에 넣으면 다음 트랜잭션이 큐로부터 이 데이타를 꺼내어 사용하는 작업이 빈번히 발생한다. 또한 최근의 ORACLE[15]과 같은 상용 DBMS에서는 지금까지 주로 트랜잭션 모니터(transaction monitor)에서 해왔던 지연 트랜잭션(delayed transaction)의 구현

Copyright (C) 2005 NuriMedia Co., Ltd.

을 큐를 사용하여 개발할 수 있도록 한다[7]. TPC-C [5] 벤치마크에서도 이러한 접근 형태를 볼 수 있다. TPC-C 벤치마크는 가상적인 주문 처리 과정을 모델로 사용하고 있다. 사용자로부터 주문을 받아 처리하는 트랜잭션은 사용자 주문 레코드를 지정된 주문 테이블에 삽입하고, 주문 처리 트랜잭션은 해당 테이블에서 주문 레코드를 꺼내는(읽고 테이블에서 삭제하는) 작업을 수행한다. 여기서도 여러 사용자 트랜잭션들과 주문 처리 트랜잭션들이 주문 테이블에 삽입과 삭제를 빈번히 수행한다. 뿐만 아니라, 이외의 기존의 여러 응용에서도 특정 테이블에 많은 수의 트랜잭션들이 동시에 삽입과 삭제 연산을 수행하는 경우를 자주 볼 수 있다.

레코드를 테이블에 삽입/삭제함에 있어 중요한 고려 사항으로, 레코드 삽입시 삽입 대상 페이지를 선택하는 문제와 레코드 삭제시 삭제된 공간을 비축(space reservation)하는 문제가 있다. 이 두 가지 문제는 테이블에 레코드가 삽입 또는 삭제될 때 시스템의 성능을 좌우하는 중요한 요소가 된다. 삽입 대상 페이지 선택 문제는 [11]에 의해 심도있게 연구되어, 기존의 페이지 선택 기법의 단점을 극복하는 여러 가지 방법을 제시하고 벤치마크를 통해 다수의 레코드가 추가되는 경우 기존의 방법보다 우수한 성능을 가짐을 보였다. 공간 비축 문제의 경우는 비교적 최근에 해결 방법이 발표되었다[1] [9, 14]. 그러나 이러한 공간 비축 방법들은 여러 트랜잭션이 동시에 한 페이지 내의 공간을 비축하지 않는 것을 가정하여, 다수의 트랜잭션이 동시에 동일 테이블에 삽입과 삭제를 수행하는 경우에는 좋지 않은 성능을 보이게 된다.

본 논문에서는 데이타 페이지의 정확한 비축 공간 정보를 추적하여, 기존 방법의 문제점을 해결하는 새로운 공간 비축 방법을 제안한다. 또한 시뮬레이션 실험을 통해 제안된 방법이 다수의 트랜잭션들이 동일 테이블에 대한 삽입과 삭제를 수행하는 경우 기존의 비축 공간 방법보다 우수한 성능을 가짐을 보인다.

본 논문의 구성은 다음과 같다. 2절에서는 공간 비축 문제에 대해 자세히 설명하고, 3절에서는 기존에 발표된 공간 비축 방법에 대해 설명하고, 이들을 새로운 응용분야에 그대로 사용하는 경우에 발생하는 문제점에 대해 언급한다. 4절에서는 본 논문에서 제안하는 새로운 공간 비축 방법을 설명하고 기존의 방법들과 비교한다. 5절에서는 시뮬레이션 실험을 통해 제안된 방법과 기존의 방법들의 성능을 수량적으로 비교하고 이를 통한 결론을 도출한다. 마지막 6절에서는 논문의 내용을 요약하고 향후 연구 과제에 대해 언급하기로 한다.

1) 공간 비축 문제의 해결책은 이미 여러 상용 시스템들에서 사용되었으나, 문헌을 통한 발표는 비교적 최근에 이루어졌다.

2. 공간 비축 문제

한 트랜잭션이 레코드를 삭제하여 생성되는 빈 공간[2]은 해당 트랜잭션이 철회되는 경우 무효화(undo) 작업에 의해 이 공간이 다시 사용되어야 하기 때문에, 삭제 연산을 수행한 트랜잭션이 완료(commit)될 때까지 이 빈 공간은 다른 트랜잭션에 의해 사용될 수 없다. 이렇게 가용 공간임에도 불구하고 잠시동안 다른 트랜잭션에 의해 사용할 수 없는 공간을 비축 공간(reserved space)라 부르고, 비축 공간을 생성한 트랜잭션을 비축 트랜잭션이라 부른다. 비축 트랜잭션이 종료되면 트랜잭션이 비축한 공간이 모두 반환되어 다른 트랜잭션이 사용할 수 있게 된다. 공간 비축 문제는 로크의 단위가 페이지보다 작은 단위를 사용하는 DBMS에서만 발생하기 때문에, 본 논문에서는 레코드 단위의 로크를 사용하는 시스템을 가정한다.

예를 들어 데이타 페이지 P가 50 바이트의 가용 공간을 갖고 있고, 트랜잭션 T1이 페이지 P에 저장된 100 바이트의 레코드를 삭제하였다고 가정하자. 삭제 후 페이지 P는 전체 150 바이트의 가용 공간을 갖게 된다. 만일 다른 트랜잭션 T2가 페이지 P에 80 바이트의 레코드를 삽입하고 완료한 경우를 생각해보자. 이때 트랜잭션 T1에 문제가 발생하여 철회될 때, 페이지 P에서 삭제한 레코드를 다시 복원시키기 위해서는 100바이트의 공간이 필요하지만, 페이지 P에는 단지 30 바이트의 공간만이 남아 있어 트랜잭션 T1은 철회될 수 없게 된다.

이러한 문제를 막기 위해서는 공간 할당시 삽입 대상 페이지에 비축 공간을 제외한 충분한 가용 공간이 있는가를 검사할 필요가 있다. 본 논문에서는 이를 비축 공간 검사라 부른다. 위 예에서는 비축 공간 검사를 통해 트랜잭션 T1이 수행되는 동안은 트랜잭션 T2가 페이지 P에서 80 바이트의 공간을 할당하는 것을 막아야 한다. 공간 할당에 실패한 트랜잭션은 다른 페이지에 대해 공간 할당을 시도한다. 비축 공간 검사를 위해서는, 비축 공간을 소유한 페이지에 대한 정보를 유지하여 비축 공간 검사에 사용될 수 있도록 한다. 기존에 제안된 대부

2) 엄밀하게는, 레코드 갱신으로 레코드의 길이가 이전의 레코드의 길이보다 짧은 경우도 빈 공간이 생성되나, 본 논문에서는 설명의 편의를 위해 레코드가 삭제되는 경우만을 가정한다. 제안된 방법은 이러한 레코드 축소 연산에도 적용 가능하다.

Copyright (C) 2005 NuriMedia Co., Ltd.

분의 공간 비축 방법들은 비축 공간 정보 관리의 효율성을 위해 페이지의 자세한 비축 공간 정보를 유지하지 않고 대략의 정보만을 유지한다. 이로 인해 비축 공간 검사시 부정확한 비축 공간 정보로 인해 잘못된 판정을 내릴 수 있다. 즉, 페이지에 비축 공간 이외의 충분한 가용 공간이 존재하는 경우에도 비축 공간 검사가 실패하는 경우가 발생한다.

본 논문에서는 레코드 삽입시 대상 페이지를 찾는 방법으로 FSIP(Free Space Inventory Page)3)[6, 13, 14]를 참조하여 선택하는 것을 가정한다. FSIP는 일반 데이타 페이지와는 달리 주위 여러 페이지들의 요약된 가용 공간 정보를 저장한 페이지로, 트랜잭션이 공간 할당에 필요한 가용 공간을 갖고 있는 페이지를 찾을 때 사용된다. 일반적으로 FSIP에 저장된 정보는 페이지들의 정확한 가용 공간 양을 기록하지 않고 대략의 가용 공간 정보를 기록하고 있어[13], FSIP를 참조하여 선택한 페이지에 실제로 원하는 양의 가용 공간이 없는 경우도 발생한다. 또한 FSIP에는 페이지의 비축 공간 정보가 기록되지 않아, FSIP를 참조하여 선택한 페이지가 비축 공간 검사에 실패할 수도 있다.

3. 기존의 공간 비축 방법들

3.1 기본적 방법: BASIC[13]

BASIC은 페이지의 비축 공간 소유 여부를 페이지 단위 로크를 이용하여 기록하는 방법이다. 트랜잭션이 페이지 P내의 레코드 삭제를 수행한 경우, 해당 페이지에 IX 형식의 로크를 거는 방법으로 비축 공간이 존재함을 알리게 된다. 비축된 공간은 해당 트랜잭션이 종료될 때까지 유지되어야 하기 때문에, 이 로크는 트랜잭션이 종료할 때까지 유지된다. 레코드 삽입을 위해 공간 할당이 필요한 트랜잭션은 FSIP을 참조하여 선택된 페이지에 EX 형식의 로크를 걸어 비축 공간 존재 여부를 검사한다. 페이지에 비축 공간이 있는 경우에는 IX 형식의 로크가 걸려있기 때문에 EX 형식 로크가 허락되지 않는다. 로크가 허락되지 않는 경우 트랜잭션의 대기(block)를 막기 위해 로크를 요청할 때는 conditional 형식으로 시도한다. 또한 이 로크는 단순히 해당 페이지에 비축 공간 존재 여부 확인만을 위해 필요하기 때문에, 앞서의 IX 형식의 로크와는 달리 로크 허락 즉시 반환하는 instant 형식의 로크를 사용한다. 이와 같이 비축 공간 여부의 기록과 검사를 위한 로크를 비축 로크(reservation lock)라고 부르기로 한다. 비축 로크는 단순히, 페이지의 비축 공간 소유 여부 기록 및 검사를 위해 사용되는 페이지 단위의 로크로, 일반적인 레코드 단위의 데이타 로크와는 별개의 것이다. 비축 로크를 포함한 트랜잭션의 모든 로크는 트랜잭션 종료시 자동적으로 풀어지기 때문에 트랜잭션 종료시 비축 공간은 자동적으로 없어지게 된다.

BASIC은 기존의 로크 관리자의 기능을 사용하는 장점과 비교적 간단한 방법으로 공간 비축 문제를 해결하는 점, 비축 공간 정보의 설치/확인/제거를 위해 추가의 I/O가 필요없다는 점, 그리고 비축 공간 정보의 제거를 위한 추가의 작업이 불필요하다는 장점을 갖고 있다. 그러나 공간의 할당과 반환을 위해 별도의 로크 연산이 필요하다는 단점과, 단순히 페이지의 비축 공간의 존재 여부로 비축 공간 검사가 이루어지기 때문에 비축 공간 검사가 자주 실패될 수 있다는 단점을 갖는다.

3.2 Startburst에서의 방법: STARBURST[9]

STARBURST는 모든 데이타 페이지의 헤더에 총 가용 공간의 크기를 기록하는 FREE 필드, 페이지에 비축된 공간의 크기를 기록하는 RSVD 필드, 페이지에 공간을 비축한 트랜잭션들 중 가장 늦게 수행된 트랜잭션의 식별자를 기록하는 TRANS 필드, 그리고 그 트랜잭션이 해당 페이지에 비축한 공간의 크기를 기록하는 TRSVD 필드를 두어, 이 정보를 이용하는 방법이다.

레코드 삭제 등을 통해 빈 공간을 생성하는 트랜잭션은 페이지 헤더의 해당 필드를 적절히 갱신하여 해당 공간의 비축 사실을 기록하게 된다. 레코드 삽입 등을 위해 공간 할당을 요구하는 트랜잭션은 페이지의 헤더를 검사하여, 자신이 요구하는 공간의 크기가 FREE-RSVD보다 작은 경우만 해당 페이지에서 공간을 할당한다. 만일 공간 할당을 요구한 트랜잭션의 식별자가 TRANS와 같은 경우에는, 요구하는 공간의 크기가 FREE-RSVD+TRSVD 보다 작은 경우에 공간 할당을 허용하여 자신이 비축한 공간도 활용할 수 있도록 한다. STARBURST에서는 비축 트랜잭션 종료시 비축한 공간이 존재하는 모든 페이지를 접근하여 헤더의 필드 값을 갱신하는 작업이 매우 비용이 크기 때문에, 종료시에 비축 공간 정보를 갱신하지 않는다. 대신 다른 공간 요구 트랜잭션이 레코드 삽입을 위해 페이지에 비축 공간 검사를 시도할 때, 현재 수행 중인 모든 트랜잭션의 식별자가 삽입 대상 페이지 헤더의 TRANS 보다 크다면 (즉, TRANS 보다 나중에 시작되었다면), 해당 페이지에 공간을 비축한 트랜잭션들은 모두 종료되었음을 의미하므로 RSVD를 0으로 갱신한다

STARBURST는 BASIC과는 달리 공간 비축을 위해

3) 이것은 SMP(Space Map Page)라고 불리기도 한다.

Copyright (C) 2005 NuriMedia Co., Ltd.

비축 로크 연산을 요하지 않는다. 또한 페이지에 비축된 공간이 있어도, 이를 제외한 가용 공간(즉, FREE-RSVD)이 충분하다면 해당 공간을 사용할 수 있는 장점을 갖는다. 그리고 특별한 경우이지만, 공간을 요구하는 트랜잭션의 식별자가 페이지의 TRANS와 같은 경우에는 트랜잭션이 비축한 공간을 활용할 수 있는 장점도 갖고 있다. 그러나 STARBURST에서는 비축 트랜잭션 종료시 RSVD 값이 즉시 갱신되지 않고 지연되어, 페이지 내에 존재하는 비축 공간의 실제 크기보다 커질 수 있어 비축 공간 검사가 실패될 수 있다. 또한 할당 트랜잭션의 식별자가 대상 페이지의 TRANS와 같은 경우에만 자신이 비축한 공간을 활용할 수 있어, 다른 트랜잭션들의 경우에는 비축 공간 검사 실패가 보다 자주 발생할 수 있다.

3.3 C. Mohan의 방법: CMOHAN[14]

일반적으로 DBMS에서는 페이지 내에 가용 공간 조각들을 연결 리스트로 연결하여 관리한다. 이 가용 공간 조각들 중 마지막에 위치하는 공간 조각을 CFS(Contiguous Free Space)라 하고, 나머지 가용 공간 조각들을 NFS(Non-contiguous Free Space)라 한다. 전체 가용 공간 TFS(Total Free Space)는 CFS와 NFS의 합으로 정의된다.

CMOHAN에서는 모든 데이타 페이지의 헤더에 NFS 내에 비축 공간의 존재 여부를 기록하는 RSB1 비트와 CFS내에 비축 공간의 존재 여부를 기록하는 RSB2 비트를 둔다. 빈 공간이 생성되는 경우 빈 공간이 NFS와 CFS 중 어느 곳으로 편입되는가에 따라 RSB1 또는 RSB2를 1로 갱신시키고, 해당 페이지에 IX형식의 비축 로크를 걸어 해당 페이지에 비축 공간이 존재함을 기록한다.

공간 할당을 시도하는 트랜잭션은 삽입 대상 페이지의 헤더를 조사하여 RSB1과 RSB2 모두 0이면 해당 페이지에 비축된 공간이 없음을 의미하므로, 비축 로크 없이 바로 페이지에서 공간을 할당한다. RSB1은 1이지만 RSB2가 0인 경우에는 CFS에는 비축된 공간이 없음을 의미하므로, CFS의 크기가 할당할 공간보다 크다면 비축 로크 요구 없이 바로 CFS에서 필요한 공간을 할당하게 된다. RSB1과 RSB2가 모두 1이라면 해당 페이지가 비축 공간이 존재할 가능성이 있으므로, BASIC에서와 같이 비축 로크를 이용하여 페이지에 비축 공간의 존재 여부를 판단한다.

비축 트랜잭션 종료시 해당 트랜잭션이 비축한 공간이 풀리기 때문에, RSB1과 RSB2 값을 적절히 갱신해야 한다. 그러나 CMOHAN에서는 STARBURST처럼 비축 트랜잭션 종료시 RSB1과 RSB2를 갱신하지 않고, 나중에 다른 트랜잭션이 페이지의 commit_LSN[12, 13] 등을 이용하여 페이지가 완료 상태에 있는 경우에 0으로 갱신한다.

CMOHAN은 경우에 따라 비축 로크 요청 없이 공간 할당과 반환이 가능하다는 장점과, BASIC과는 달리 비축된 공간이 존재하는 페이지에서도 공간 할당이 가능하다는 장점을 갖는다. CMOHAN의 단점은 비축 공간 검사가 실패하는 경우에 있다. CMOHAN에서는 비축 공간 검사가 실패로 결정나기까지 한번의 비축 로크 연산과 한번의 버퍼 적재 연산을 필요로 하기 때문에, 다른 방법보다 많은 낭비를 초래한다. 더구나 이 방법은 RSB1과 RSB2 두 비트만을 이용하여 비축 공간 상태를 추적하기 때문에, 비축 공간 검사의 잘못된 판정으로 실패 횟수가 증가할 수 있다. 또한 RSB1과 RSB2의 갱신이 지연되기 때문에 부정확한 비축 공간 정보로 인해 비축 공간 검사의 잘못된 판정으로 추가의 검사 실패가 발생하기도 한다.

3.4 기존 공간 비축 방법들의 평가

기존의 공간 비축 방법들은 대부분의 데이타 페이지에 비축 공간이 없거나 한 페이지에 최대 한 트랜잭션만이 비축한 공간이 존재한다고 가정하였다. 그러므로 기존의 방법들은 비축 공간 정보 관리에 소요되는 부하를 줄이기 위해, 페이지의 자세한 비축 공간 정보를 기록하지 않는 방법과, 비축 트랜잭션의 종료시에도 비축 공간 정보의 갱신을 즉시하지 않고 나중에 다른 작업과 함께 하는 방식을 택하였다. 그러나 다수의 트랜잭션이 동일 테이블에 대해 동시에 삽입과 삭제 연산을 수행하는 환경에서는 공간을 할당하려는 트랜잭션이 FSIP를 참조하여 선택한 페이지에 다수의 다른 트랜잭션에 의해 비축된 공간이 존재할 경우가 높다. 이 경우 기존의 방법을 사용하면 비축 공간 검사가 부정확하여 충분한 가용 공간을 갖은 페이지에 대한 검사가 자주 실패하게 될 수 있다.

비축 공간 검사가 실패하면 공간 할당 트랜잭션은 FSIP를 다시 참조하여 다른 페이지를 선택해야 하기 때문에, FSIP를 다시 버퍼에 적재하는 연산이 필요하게 된다. 또한 CMOHAN과 STARBURST와 같이 비축 공간 검사 이전에 미리 대상 데이타 페이지를 버퍼에 적재하는 방법에서는, 검사가 실패되면 다른 데이타 페이지를 버퍼로 적재하기 때문에 추가의 버퍼 연산이 낭비된다. 특히 트랜잭션이 공간 할당을 위해 여러 번의 다른 데이타 페이지에 대한 버퍼 적재 연산을 수행하게 되면, 결국 트랜잭션와 working-set의 크기를 증가시킨

Copyright (C) 2005 NuriMedia Co., Ltd.

다. 이는 버퍼의 적중률(hit ratio)을 떨어뜨려 추가의 물리적 디스크 I/O를 유발하게 되어 급격히 성능이 저하된다.

그러므로 다수의 트랜잭션이 동시에 특정 테이블에 삽입과 삭제를 자주 수행하는 환경에서, 공간 비축 방법이 좋은 성능을 보이기 위해서는 각 페이지에 비축된 공간의 정보를 정확히 추적하고 이를 통한 정확한 비축 공간 검사가 필요하다. 정확한 비축 공간 검사는 잘못된 판정을 제거하여 검사의 실패 횟수를 최소화하고 검사 실패로 유발되는 낭비를 최소화한다. 또한 대상 페이지의 버퍼 적재 전에 비축 공간 검사를 실시하도록 하여 검사 실패시의 낭비를 최소화해야 한다.

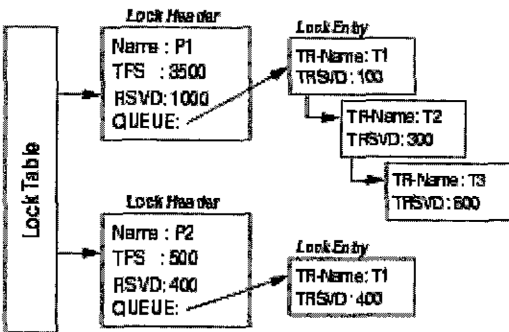

그림 1 제안된 방법을 위한 로크 테이블의 확장

4. 제안된 공간 비축 방법: NEW

대부분의 DBMS에서 데이타 객체가 로크가 되면, 로크 테이블에 로크 정보를 기록하기 위해 로크 헤더(lock header)와 로크 엔트리(lock entry)가 삽입된다(그림 1 참조)[4, 6]. 로크 헤더는 로크된 데이타 객체의 상태를 기록하고, 로크 엔트리는 로크를 소유한 트랜잭션 또는 해당 객체의 로크 허용을 기다리는 트랜잭션의 상태를 기록한다.

NEW는 BASIC과 유사하게, 한 트랜잭션이 공간을 할당/반환하는 경우 항상 대상 페이지에 대해 비축 로크를 시도한다. NEW는 로크 헤더에 TFS와 RSVD 필드를 추가하여, 비축 로크된 페이지의 총 가용 공간의 크기와 총 비축 공간의 크기를 기록한다. 또한 로크 엔트리에 TRSVD 필드를 추가하여, 비축 로크를 설치한 트랜잭션이 해당 페이지에 비축한 가용 공간의 크기를 기록하도록 한다. 그림 1에 의하면 페이지 P1과 P2 내에 각각 비축 공간이 존재함을 알 수 있다. 페이지 P1은 3500 바이트의 총 가용 공간을 갖고, 그 중 1000 바이트는 트랜잭션 T1, T2, T3에 의해 비축되었음을 알 수 있다. 트랜잭션 T1은 페이지 P2의 유일한 비축 트랜잭션이고, 500 바이트의 가용 공간 중 400 바이트를 비축하고 있는 것을 알 수 있다.

그림 2는 페이지 P에서 트랜잭션 T에 의해 반환되는 B바이트 공간을 비축하는 과정을 보여준다

NEW는 페이지 P에 IS 형식의 로크를 요청하여, 해당 페이지의 로크 헤더와 요청 트랜잭션에게 할당된 로크

```
페이지 P에서 B 바이트 길이의 레코드 삭제
페이지 P에 해당하는 로크 헤더 H 검색
if 검색되지 않으면 then
    페이지 P를 위한 새로운 로크 헤더 H를 할당
    H.RSVD ← 0
end if
H.TFS ← H.TFS + B
H.RSVD ← H.RSVD + B
H.QUEUE에서 트랜잭션 T에 해당하는 로크 엔트리 E 검색
if 검색되지 않으면 then
    H.QUEUE에 트랜잭션 T를 위한 로크 엔트리 E를 할당 및 삽입
    E.TRSVD ← 0
end if
E.TRSVD ← E.TRSVD + B
```

그림 2 트랜잭션 T에 의한 페이지 P내의 B 바이트 길이의 레코드의 삭제

엔트리를 검색한다. 검색된 로크 헤더의 TFS와 RSVD 필드 그리고 검색된 로크 엔트리의 TRSVD가 비축되는 공간의 크기(B) 에 따라 변경되는 것을 볼 수 있다.

그림 3은 페이지 P에서 B 만큼의 공간을 할당하는 과정을 보여준다. 트랜잭션 T는 페이지 P에 instant, conditional 형식의 IS 로크를 요구하여, 페이지에게 할당된 로크 헤더와 트랜잭션에게 할당된 로크 엔트리를 찾는다. 트랜잭션 T는 대상 페이지의 총 가용 공간(TFS)에서 총 비축된 공간(RSVD)을 제외한 부분을 사용할 수 있되, 비축 공간 중 자신이 비축한 공간(TRSVD)은 활용할 수 있기 때문에, TFS-RSVD+TRSVD 크기의 공간을 사용할 수 있다. 트랜잭션이 페이지 P에 비축 공간을 갖고 있지 않는 경우는 TRSVD가 0으로 간주되어 TFS-RSVD이 사용할 수 있는 공간이 된다. 공간 할당 대상 페이지에 비축된 공간이 없는 경우(즉, 페이지에 비축 로크가 걸려 있지 않는 경우)는

Copyright (C) 2005 NuriMedia Co., Ltd.

```
페이지 P와 트랜잭션 T에 해당하는 로크 헤더 H와 로크 엔트리 E를 검색한다.
if 만일 두 개다 검색되면 then
  ▶ 트랜잭션 T는 이미 페이지 P에 비축된 공간 (E.TRSVD)을 소유
  availsize ← H.TFS - H.RSVD + E.TRSVD
else if H만 검색되면 then
  ▶ 트랜잭션 T는 페이지 P에 비축된 공간을 비소유
    availsize ← H.TFS - H.RSVD
else
  ▶ 페이지 P에는 비축된 공간이 없음
  availsize ← PAGESIZE
end if
if B > availsize then
  공간 할당은 실패되고, 다른 페이지에 공간 할당을 시도
end if
페이지 P를 버퍼에 fix 시키고, B 바이트 공간 할당을 시도
if 할당이 실패되면 then
  페이지 P를 unfix시키고 다른 페이지에 공간 할당을 시도
end if
H.RSVD ← H.RSVD - min(E.TRSVD, B)
E.TRSVD ← E.TRSVD - min(E.TRSVD, B)
H.TFS ← H.TFS - B
페이지 P를 unfix 시킴
```

그림 3 트랜잭션 T가 페이지 P에 B 바이트 공간을 할당

페이지 P의 로크 헤더가 로크 테이블에 없게 된다. 이 때는 총 비축 공간 크기(RSVD)가 0이 되고 페이지의 총 가용 공간 크기(TFS)를 물리적 페이지 크기(PAGESIZE)로 간주하여, 할당 가능한 최대 공간은 PAGESIZE가 되기 때문에 비축 공간 검사에 통과하게 된다[4].

비축 트랜잭션 종료시 해당 트랜잭션에게 할당된 로크 엔트리들은 자동적으로 로크 테이블에서 삭제되는데, 이때 각 로크 엔트리 E의 해당 로크 헤더 H를 접근하여, H.RSVD를 H.RSVD-E.TRSVD로 수정하게 되면, 손쉽게 트랜잭션 종료에 따른 페이지의 비축 공간 정보 변경을 즉시 반영시킬 수 있다.

NEW는 페이지의 정확한 비축 공간 정보를 통해 페이지 내에 사용 가능한 최대의 가용 공간을 사용할 수 있는 공간 검사법을 제공하기 때문에, 기존의 방법들과는 달리 정확한 비축 공간 검사가 가능하여, 검사 실패의 횟수를 최대한 줄인다. NEW는 STARBURST와 CMOHAN과는 달리 비축 공간 검사가 대상 페이지의 버퍼 적재 이전에 수행되기 때문에, 검사가 실패되는 경우에 낭비되는 비용이 적다. 또한 비축 공간 정보의 기록을 위해 페이지 헤더에 추가되는 필드가 없어, NEW를 사용하기 위해 기존에 이미 구축된 데이타베이스의 페이지의 포맷 변환 작업이 필요없다.

2절에서도 언급하였듯이 공간 할당시 때때로 FSIP를 참조하여 선택한 페이지가 충분한 가용 공간을 갖지 않는 경우, 비록 비축 공간 검사를 통과하여도 해당 페이지에서 공간을 할당할 수 없어 버퍼 적재 연산을 낭비하게 될 수 있다. 그러나 NEW를 사용하면 대상 페이지의 로크 헤더를 통해 정확한 가용 공간 정보(TFS)를 비축 공간 검사 때 알 수 있기 때문에, 잘못 선택된 페이지들은 걸러지게 된다. 그러나 삽입 대상 페이지에 비축 공간이 없는 경우(즉, 페이지에 비축 로크가 걸려있지 않은 경우)는 페이지의 로크 헤더가 로크 테이블에 없기 때문에, 페이지의 총 가용 공간 크기를 알 수 없게 된다 이 경우는 NEW 에서도 FSIP의 잘못된 페이지 추천으로 인한 버퍼 연산의 낭비가 발생할 수 있다. NEW에서는 이러한 낭비를 최소로 줄이기 위해, 페이지에 설치된 모든 비축 로크가 풀려도 로크 헤더를 시스템에 반환하지 않고 유지되도록 한다. 만일 가용 로크 헤더 풀(pool)에 더 이상 남은 로크 헤더가 없는 상태에서 다른 데이타 객체의 로크를 위해 로크 헤더가 필요한 경우, 이렇게 방치된 로크 헤더 중에서 가장 오랫동안 방치된 로크 헤더를 새로운 객체의 로크 헤더로 사용한다. 이 방법을 사용하면 일부 페이지에 대해서 비축된 공간이 없더라도 페이지의 정확한 가용 공간의 크기가 유지되어, FSIP의 잘못된 페이지 추천으로 발생하는 낭비되는 버퍼 연산을 줄이는 효과를 갖는다.

제안된 방법은 CMOHAN과 STARBURST과는 달리, 비축 공간 검사를 위해 추가의 로크 연산을 필요로 하는 단점과, 로크 헤더와 레코드 엔트리에 필드 추가에 따른 메모리 사용이 증가한다는 단점과, 비축 로크 설정할 때와 트랜잭션 종료시 추가의 작업을 필요로 하는 단점을 갖는다. 그러나 일반적인 DBMS를 가정할 때, 로크 헤더의 TFS와 RSVD를 위해 각각 2바이트의 추가 공간과 로크 엔트리의 TRSVD를 위해 2바이트의 추가 공간만을 필요로 하기 때문에, 최근 시스템의 메모리 용량으로 볼 때 그리 큰 부담이라 볼 수 없다. 또한 비축 로크 설정시와 트랜잭션 종료시 필요한 추가의 작업은 단지 몇 번의 단순한 산술 연산의 추가만을 필요로 하기 때문에, 전체 로크 설정에 소요시간과 트랜잭션 종료에 소요되는 시간을 고려할 때 이것이 큰 부하가 된다고 볼 수 없다.

---

4) 본 논문에서는 LOB(Large Object)를 고려하지 않기 때문에 요구 공간 크기는 항상 물리적 페이지 크기 PAGESIZE보다 작다고 가정한다.

Copyright (C) 2005 NuriMedia Co., Ltd.

5. 시뮬레이션

5.1 시뮬레이션 모델

본 절에서는 시뮬레이션을 통해 기존의 비축 공간 검사 방법들과 NEW 사이의 성능을 비교한다. 시뮬레이션은 동일 테이블에 삽입과 삭제를 수행하는 트랜잭션의 수에 따른 삽입/삭제의 성능을 비교하였다. 측정 대상은 트랜잭션들이 삽입과 삭제를 수행할 경우 비축 공간 검사 실패 횟수, 데이타 페이지에 대한 버퍼 연산 횟수, 그리고 비축 로크 요구 횟수 등을 조사하였다.

표 1 시뮬레이션 인자 및 설정 값

시뮬레이션 인자	설정 값
페이지 크기	4K 바이트
레코드 크기	(30, 80), (100, 300), (400, 700) 바이트
버퍼 슬롯의 수	32 개
실험 시작시 레코드 수	200 개
페이지 할당 방법	first fit
버퍼 연산 소요시간	2 time unit
비축 로크 연산 소요시간	2 time unit
디스크 읽기/쓰기 소요시간	50 time unit
클라이언트 개수	1-40 개
클라이언트 당 트랜잭션 실행 수	100 개
트랜잭션 길이	5 번
삽입/삭제 후 think time	1 time unit
트랜잭션간 think time	5 time unit

표 1은 본 시뮬레이션 실험에서 사용된 인자 및 설정 값을 보여준다. 시뮬레이션의 목적은 비축 공간 검사 방법에 따른 삽입과 삭제에 소요되는 실제 시간의 측정에 있지 않고, 공간 비축 방법들 사이의 성능의 비교에 있기 때문에 시뮬레이션에서는 절대적인 시간 단위를 사용하지 않고 상대적인 시간 단위를 사용하였다.

트랜잭션은 클라이언트에서 실행되고 클라이언트의 수는 실험에 따라 1개에서 40개까지 변한다. 클라이언트는 한번에 1개의 트랜잭션만을 수행시키고 트랜잭션이 종료되면 다음 트랜잭션을 실행시킨다. 그러므로 동시에 수행되는 트랜잭션의 수는 클라이언트의 수에 의해 결정된다. 연속되는 트랜잭션 사이는 5 tu(time unit)의 think time을 둔다. 트랜잭션의 길이는 트랜잭션이 수행하는 삽입 또는 삭제 연산의 횟수로 결정되고 이번 실험에서는 5로 설정되었다. 즉, 트랜잭션은 삽입과 삭제를 임의로 선택하여 5번 수행하되, 각 연산 사이에는 1 tu의 think time을 둔다.

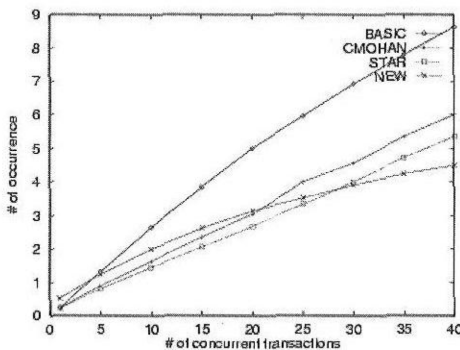

그림 4 레코드 삽입당 발생한 검사 실패 횟수

데이타 페이지의 크기는 4K 바이트로 설정하였고, 삽입되는 레코드의 크기는 세 가지로 분류되어 임의적으로 선택하되 작은 레코드는 최소 30 바이트에서 최대 80 바이트 범위에서, 중간 크기의 레코드는 100 바이트에서 300 바이트에서, 그리고 긴 레코드는 400 바이트에서 700 바이트에서 각각 임의로 결정되게 하였다. 버퍼 슬롯의 총 개수는 32개로 정하였다. 버퍼 연산에 소요되는 시간은 디스크 I/O를 요하지 않는 경우에 2 tu으로 정하였고, 비축 로크 연산에 소요되는 시간 역시 2 tu를 정하였고, 디스크 읽기/쓰기 작업은 50 tu으로 정하였다. 시뮬레이션은 200개의 레코드가 이미 테이블에 균등하게 분포된 상태에서 시작하도록 하였다. 레코드 삽입시 대상 페이지를 찾는 방법은 가장 일반적이고 쉬운 방법인 first-fit 방법[11]을 사용하였다[5].

본 절에서 보이는 모든 결과는 동일 실험을 10번 반복하여 평균을 구한 값이다.

5.2 시뮬레이션 결과 및 분석

그림 4는 한 개의 레코드 삽입에 발생한 비축 공간 검사의 실패 횟수를 보여주어, 비축 공간 검사들이 얼마나 정확한 결정을 내리는 가를 보여준다.

BASIC은 비축 공간 검사가 단순히 비축 공간 소유 여부로 결정되기 때문에 트랜잭션의 수가 증가함에 따라 다른 방법들보다 많은 검사 실패 횟수를 보여준다. NEW가 비교적 적은 수의 클라이언트가 수행되는 경우 CMOHAN과 STARBURST보다 검사 실패 횟수가 약간 많게 되는데, 그 이유는 CMOHAN과 STARBURST에서는 FSIP가 부적절한 페이지를 추천하는 경우, FSIP를 수정하여 해당 페이지에 추가 가용 공간이 발생할 때까지 공간 할당시 대상 페이지로 선택되지 않도록 한다. 반면 NEW는 이러한 상황에서 FSIP를 수정하지 않아 계속 이러한 페이지가 대상 페이지로 선택되기 때문이다. 그러나 검사 실패 횟수 차이는 트랜잭션

5) 대상 페이지를 찾는 방법도 비축 공간 검사 방법들의 성능의 영향을 미칠 수 있으나, 구현의 편의상 본 논문에는 first fit를 가정하였다.

Copyright (C) 2005 NuriMedia Co., Ltd.

수가 증가함에 따라 정확한 비축 공간 검사로 절약되는 실패 횟수에 상쇄되어, 트랜잭션의 수가 약 30이상부터는 NEW는 다른 방법들보다 적은 실패 횟수가 유발된다.

그림 5는 한번의 레코드 삽입에 소요되는 데이타 페이지에 대한 버퍼 적재 연산의 횟수를 보여준다. 레코드 삽입을 위해서는 데이타 페이지를 읽기 위한 최소한 한번의 버퍼 연산을 요하기 때문에, 이상적인 데이타 페이지 버퍼 연산 횟수는 1번이다. NEW는 비축 공간 검사를 통과하는 경우에만 해당 데이타 페이지를 버퍼에 적재하기 때문에, 클라이언트의 수가 증가하는 것과 무관하게 소요되는 버퍼 연산의 수가 이상적 값인 1에 매우 근접한 것을 볼 수 있다.

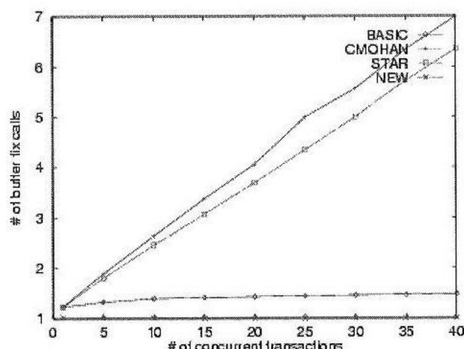

그림 5 레코드 삽입당 소요되는 버퍼 연산 횟수

BASIC도 비축 공간 검사가 통하는 경우에만 해당 데이타 페이지를 적재하기 때문에 클라이언트의 수가 증가함과 무관하게 버퍼 연산의 수가 변하지 않음을 알 수 있다. 그러나 버퍼 연산 횟수가 1보다 약간 큰 값(약 1.3-1.4)을 유지하게 되는데, 이는 FSIP가 부적절한 페이지를 추천하기 때문이다. 그러므로 NEW와 BASIC의 버퍼 연산 횟수 차이는 FSIP에 기록된 가용 공간 정보의 정확도를 보여준다.

STARBURST와 CMOHAN에서 모두 비축 공간 검사를 위해 해당 페이지를 접근하기 때문에, 트랜잭션의 수의 증가에 따른 실패 횟수 증가(그림 4 참조)만큼 버퍼 연산의 수가 증가되는 것을 볼 수 있다. CMOHAN은 단지 RSB1과 RSB2 두 비트만을 이용하여 비축 공간 정보를 추적하는 반면, STARBURST는 페이지 헤더의 상대적으로 자세한 가용 공간의 정보를 이용하기 때문에 STARBURST가 CMOHAN보다 다소 적은 버퍼 연산 횟수를 보인다.

그림 6은 클라이언트 수의 증가에 따른 한 개의 레코드 삽입에 요구되는 비축 로크의 요청 횟수를 보여준다.

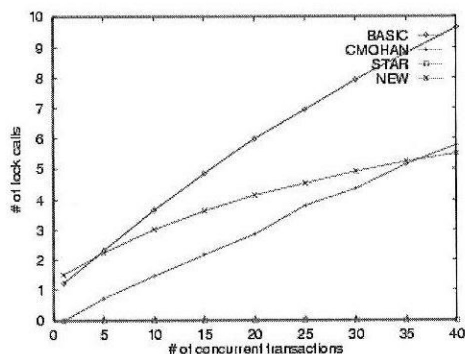

그림 6 레코드 삽입당 필요한 비축 로크 횟수

비축 공간 검사에 로크를 사용하지 않는 STARBURST는 0번의 로크 수를 보여준다. CMOHAN도 RSB1과 RSB2 값에 따라 비축 로크 없이 비축 공간 검사를 통과가 가능하여, BASIC과 NEW보다 비축 로크 연산 횟수가 적다. 그러나 클라이언트의 수가 증가함에 따라 부정확한 비축 공간 검사로 검사 실패 횟수가 증가(그림 4 참조)하여 클라이언트 수가 35 이상부터는 오히려 NEW보다 비축 로크 연산 횟수가 증가하게 된다. BASIC은 부정확한 비축 공간 검사로 인한 검사 실패 증가와 매 검사에 한번의 로크가 필요하기 때문에, 네 방법 중 가장 많은 비축 로크 연산이 필요하고 클라이언트 수가 증가함에 따라 그 차이는 더 벌어짐을 볼 수 있다.

그림 7은 클라이언트 수의 증가함에 따라 레코드 삽입에 유발되는 실제 물리적 디스크 페이지 읽기 횟수를 보여준다. CMOHAN과 STARBURST는 비축 공간 검

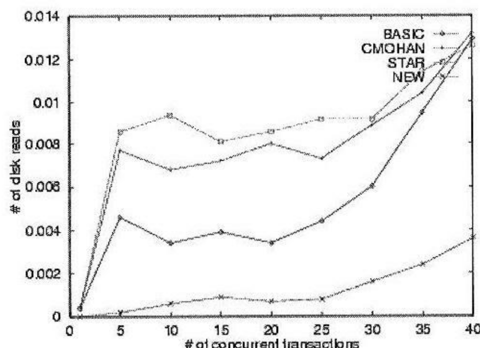

그림 7 레코드 삽입당 필요한 디스크 read 수

Copyright (C) 2005 NuriMedia Co., Ltd.

사를 위해 대상 페이지를 버퍼로 적재하기 때문에, 클라이언트 수가 증가함에 따라 레코드 삽입에 쓸모없이 버퍼에 올려지는 페이지 수가 증가하게 되어(그림 5 참조), 트랜잭션의 working-set의 점차 커지고 버퍼 적중률이 떨어져 점차 많은 수의 디스크 읽기가 발생하게 된다.

BASIC도 클라이언트 수의 증가와 무관하게 일정 수의 버퍼 연산만을 필요로 하지만, 비축된 공간을 포함한 페이지에 레코드 삽입이 불가능하여, 레코드 삽입시 비축된 공간이 전혀 없는 페이지만을 찾아 버퍼에 올리기 때문에, locality가 저하로 인한 버퍼의 적중률 저하로 점차 많은 수의 디스크 읽기를 유발한다.

그림 5에서 볼 수 있듯이, NEW는 클라이언트 수가 증가하여도 1번에 근접한 버퍼 연산만을 필요로 하고 비축 공간이 존재하는 페이지에서도 공간 할당이 가능하여, 다른 방법에서 유발하는 디스크 읽기 수의 약 1/4 정도만 유발되게 된다. 더구나 그림 7은 단순히 디스크 읽기 수만을 측정했으나, 실제로는 버퍼 교체시 때로는 victim으로 선택된 페이지의 디스크 쓰기 작업을 필요로 하기 때문에 실제 발생되는 디스크 I/O 연산 횟수의 차이는 더 벌어지게 된다.

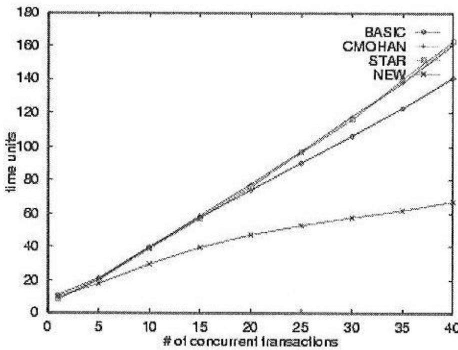

그림 8 레코드 삽입당 소요되는 경과시간.

그림 8은 클라이언트 수가 증가함에 따라, 레코드 삽입에 소요되는 경과시간을 보여준다. 그림에서 볼 수 있듯이 클라이언트의 개수가 10이상인 경우부터 NEW는 다른 방법과 경과 시간의 차이를 보이기 시작하여, 점차 그 격차가 벌어져 클라이언트의 개수가 40인 경우에는 NEW가 기타 방법보다 75tu 이상의 성능을 보이는 것을 볼 수 있다. 이러한 차이의 주요 원인은 NEW가 다른 방법들에 비해 월등히 적은 횟수의 디스크 I/O를 유발하기 때문이다. 본 실험에서는 디스크 I/O를 유발하지 않는 버퍼 연산의 오버헤드와 비축 로그 연산의 오버헤드를 같다는 가정 하에서 측정하여, 약간의 차이는 있을지는 모르나, 위 결과에 영향을 주지는 않는다

본 실험의 결과에서 알 수 있듯이, 다수의 트랜잭션이 동시에 특정 테이블에 레코드 삽입과 삭제를 자주 수행하는 환경에서는 NEW와 같이 정확한 비축 공간 검사와 검사 통과 후에 대상 페이지를 버퍼로 적재하는 방식이 다른 방식의 공간 비축 방법들보다 월등히 좋은 성능을 보인다. 이는 검사 실패로 유발되는 불필요한 버퍼 적재 연산을 최소화하고 버퍼의 적중률 저하를 방지하여 물리적 디스크 I/O를 최소화하기 때문이다.

6. 결론 및 향후 연구 계획

여러 새로운 DBMS 응용들 중에 주요 응용분야인 워크플로우, 그룹웨어, 주문 처리 응용에서는 다수의 트랜잭션들이 동일 테이블에 레코드의 삽입과 삭제 연산을 수행하는 접근 형태가 자주 일어난다. 공간 비축 문제는 레코드의 삽입/삭제시 고려할 중요 문제 중 하나이다. 그러나 기존의 공간 비축 방법들은 주로 페이지가 비축된 공간을 소유하지 않는다는 가정 하에 설계되어서, 비축 공간 검사가 정확하지 않거나, 검사 실패시 많은 부하가 초래되어, 레코드의 삽입과 삭제가 주요 연산인 응용에서 사용하기에는 성능 면에서 많은 문제점을 갖고 있다.

우리는 로그 테이블에 페이지의 비축 공간 상태를 정확히 추적할 수 있는 추가의 필드를 두어, 기존의 방법보다 보다 정확하고 적은 부하의 비축 공간 검사를 수행할 수 있는 새로운 공간 비축 방법을 제안하였다. 또한 우리는 시뮬레이션 실험을 통해, 제안된 방법이 동일 테이블을 동시에 접근하는 트랜잭션의 수가 많은 경우에 기존의 방법들보다 매우 우수한 성능을 보임을 보였다.

공간 비축 방법의 성능은 페이지 할당 방법에 따라 많은 영향을 받을 수 있다. 현재 우리는 [11]에서 언급된 여러 가지 페이지 할당 방법에 따른 여러 가지 공간 비축 방법의 성능을 비교를 할 계획이다. 또한 로그 테이블에 기록된 페이지 정보가 FSIP에 저장된 페이지 정보보다 정확한 정보라는 점에 착안하여, 레코드 삽입시 대상 페이지를 선택하는 방법으로 로그 테이블을 검색하는 방법에 대해 연구할 계획이다.

참 고 문 헌

[1] D. Chamberlin, "Using The New DB2: IBM's Object-Relational Database System," Morgan Kaufmann Publichers, Inc. 1996

Copyright (C) 2005 NuriMedia Co., Ltd.

[2] A. K. Elmagarmid, "Database Transaction Models for Advanced Applications, Morgan Kaufmann Publichers, Inc. 1992

[3] H. Garcia-Molina and K. Salem, "Services for a Workflow Management System," IEEE Database Engineering Bulletin, Vol.17, No.1, pp. 40-44, 1994

[4] V. Gottemukkala, T. J. Lehman,"Locking and Latching in a Memory-Resident Database System," Proc. of VLDB, pp. 533-544, Aug. 1992

[5] J. Gray, "The Benchmark Handbook: For Database and Transaction Processing Systems," 2nd Ed., Morgan Kaufmann Publichers, Inc. 1993

[6] J. Gray and A. Reuter, "Transaction Processing: Concepts and Techniques," Morgan Kaufmann Publichers, Inc. 1993

[7] J. Gray, "THESIS: Queues are Databases," HPTS'95, 1995

[8] T. J. Lehman and P. J. Gainer, "DB2 LOBs: The Teenage Years," Proc. of IEEE TCDE, pp. 192-199, Mar. 1996

[9] B. G. Lindsay, C. Mohan and M. H. Pirahesh, "Method for Reserving Space Needed for "Rollback" Actions," IBM Technical Disclosure Bulletin, Vol.29, No.6, pp. 2743-2746, Nov. 1986

[10] G. M. Lohman, B. G. Lindsay, H. Pirahesh and K. B. Schiefer, "Extensions to Starburst: Objects, Types, Functions, and Rules," Comm. ACM, Vol.34, No.10, pp. 94-109, Oct. 1991

[11] M. L. McAuliffe, M. J. Carey and M H. Solomon, "Towards Effective and Efficient Free Space Management," Proc. of ACM SIGMOD, pp. 389-400, Jun. 1996

[12] C. Mohan, "Commit_LSN: A Novel and Simple Method for Reducing Locking and Latching in Transaction Processing Systems," Proc. of VLDB, Aug. 1990

[13] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," ACM Trans. Database Syst., Vol.17, No.1, pp. 94-162, Mar. 1992

[14] C. Mohan and D. Haderle, "Algorithms for Flexible Space Management in Transaction Systems Supporting Fine-Granulairty Locking," Proc. of EDBT, Mar. 1994

[15] Oracle Corporation, "Oracle8TM Server Application Developer's Guide," Jun. 1997

이 강 우

1991년 2월 서울대학교 계산통계학과(이학사). 1993년 2월 서울대학교 계산통계학과 전산과학전공(이학석사). 1993년 3월부터 현재까지 서울대학교 전산과학과 박사과정. 관심분야는 트랜잭션 처리 시스템, 질의어 처리 시스템, 객체관계형 데이타베이스 시스템.

김 형 주

제 25 권 제 2 호(B) 참조

Copyright (C) 2005 NuriMedia Co., Ltd.