데이타베이스 공유 환경에서 빠른 회복을 위한 버전 관리

정용석[0] 조행래
영남대학교 컴퓨터공학과
yongs@yumail.ac.kr[0], hrcho@yu.ac.kr

A Version Management for Fast Recovery in a Database Sharing System

Yongseok Jeong[0] Haengrae Cho
Dept. of Computer Engineering, Yeungnam University

요 약

데이타베이스 공유 시스템(Database Sharing System : DSS)은 고성능 트랜잭션 처리를 위해 다수 개의 노드들을 연동하며, 각 노드는 디스크 계층에서 데이타베이스를 공유한다. DSS를 구성하는 노드들이 고장 날 경우, 데이타베이스를 정확한 상태로 복구하기 위한 회복 기법이 필요하다. DSS에서의 데이타베이스 회복 과정은 여러 노드에 분산된 로그 레코드의 병합 작업을 포함하며, 병합된 로그 레코드를 이용한 *REDO* 작업을 수행하여야 하므로 일반적인 단일 데이타베이스 시스템에 비해 많은 시간이 소요된다. 본 논문에서는 Oracle 9*i*에서 개발된 캐쉬 연합(cache fusion) 기법을 개선한 버전 관리 기법을 제안한다. 제안한 기법은 DSS를 구성하는 단일 노드의 고장 시 로그 병합 과정이 필요 없으므로 빠른 회복을 지원할 수 있으며, Oracle 9*i*에서 발생하는 빈번한 디스크 저장 오버헤드를 줄일 수 있다는 장점을 갖는다.

1. 서론

데이타베이스 공유 시스템(Database Sharing System : DSS)에서는 데이타베이스를 저장하고 있는 모든 디스크들이 상이한 프로세서와 운영체제, 그리고 DBMS를 갖는 여러 노드들에게 공유된다[4]. 각 노드에서 실행되는 트랜잭션들은 공유 디스크에 저장된 데이터베이스의 모든 부분을 액세스하고 수정할 수 있다. DSS의 장점은 DBMS들이 별개의 노드에서 상호 독립적으로 동작하기 때문에 트랜잭션 실행시 부하 균형이 가능하고, 새로운 노드의 추가가 용이하며, 기존 노드의 고장이 전체 시스템에 큰 영향을 주지 않는다는 점들을 들 수 있다[1,2].

DSS를 구성하는 각각의 노드는 자신의 버퍼에 최근에 액세스한 페이지들을 캐싱한다. 페이지에 대한 효율적인 캐싱은 각 노드에서 발생하는 디스크 액세스 수나 노드들 간의 페이지 전송량을 줄임으로써 DSS의 성능을 크게 향상시킬 수 있다. 그러나, 노드들이 최신의 데이타를 항상 사용할 수 있기 위해서는 버퍼에 캐싱된 페이지의 일관성이 유지되어야 한다. 이를 위해 각 노드에 존재하는 버퍼 관리자는 캐쉬 일관성 기법을 지원하여야 하며, 이에 관한 많은 연구들이 기존에 수행된 바 있다[4,8].

DSS를 구성하는 노드들이 고장이 날 경우, 데이타베이스를 정확한 상태로 복구하기 위한 회복 기법이 필요하다. DSS에서 제안된 가장 대표적인 데이타베이스 회복 기법으로 ARIES/SD[4]를 들 수 있다. ARIES/SD의 경우, 회복을 위해서 페이지가 갱신될 때마다 로그 레코드가 생성되며, 각 로그 레코드에 대해 일련번호가 할당된다[4]. 이러한 로그 레코드들은 그 페이지가 디스크에 기록될 때, 혹은 로그 레코드를 생성한 트랜잭션이 완료할 경우나 페이지에 대한 소유자 노드가 변경 될 때 로그 디스크에 기록된다. 로그 디스크는 노드마다 별도로 유지되며 고장 발생 후 시스템 회복을 위하여 각 노드의 로그 레코드들을 통합하는 과정이 필요하다.

DSS에서의 회복 기법은 캐쉬 일관성을 위해 노드들 간에 발생하는 페이지 전송 방식과 동시성 제어를 위한 로킹 방식에 크게 의존한다. 노드들 간의 페이지 전송 방식은 공유 디스크를 이용하는 방법과 노드의 메모리 버퍼 간에 네트워크를 통해 직접 전송하는 방법이 존재한다[4]. 로킹의 단위는 페이지 단위 로킹, 레코드 단위 로킹으로 나눌 수 있다. 레코드 로킹은 여러 트랜잭션들이 특정 페이지에 대한 로그를 동시에 생성할 수 있다. 따라서, 페이지에 대한 회복 작업은 여러 노드에 나누어 저장된 로그를 통합한 전역 로그를 생성한 후, 전역 로그를 순차적으로 검색하고, REDO, UNDO의 과정을 거치게 된다. 이 과정 중에서 로그의 병합은 특히 많은 시간이 소요된다[3].

본 논문에서는 레코드 단위 로킹을 지원하는 DSS에서 버전 개념을 이용하여 로그 병합에 소요되는 시간을 줄일 수 있는 기법을 제안한다. 제안한 기법은 단일 노드 고장의 경우에 로그 병합이 필요 없다는 장점을 갖는다.

본 논문의 구성은 다음과 같다. 2절에서는 기존에 제안된 회복 기법에 대해 살펴보고, 3절에서는 본 논문에서 제안한 버전 관리 기법을 설명한다. 마지막으로 4절에서 결론 및 앞으로의 연구 방향에 대해 논의하기로 한다.

2. 관련 연구

본 절에서는 로그의 병합이 필요 없는 기존 데이타베이스 회복 기법에 대해 설명한다.

2.1 클라이언트 기반 로깅

클라이언트 기반의 로깅 기법은 로그 디스크를 갖는

클라이언트/서버 DBMS를 가정하였으며, 각 클라이언트의 로컬 디스크에 자신의 로그 레코드를 저장하고, 이를 이용한 데이타베이스 회복 기법을 제안하고 있다[5]. 이 기법의 장점은 회복을 위한 로그 디스크의 병합 과정이 필요 없다는 것이다. 그러나, 페이지 단위의 로킹만을 지원하고, REDO 과정에서 트랜잭션 처리의 순서대로 노드간에 페이지 전송이 발생함으로써 레코드 단위의 로킹에는 적용이 힘들고, 적용을 한다해도 오버헤드가 너무 크다는 단점을 갖는다.

2.2 IBM DB2 7.0 데이타베이스 공유버전

이 기법은 모든 노드들간에 전역 버퍼 풀(Global Buffer Pool : GBP)을 공유하며, 트랜잭션 완료시 마다 갱신된 페이지를 GBP에 저장하는 기법이다.[6] 단일 노드 고장의 경우 고장난 노드에서 수행되었던 모든 완료 트랜잭션들의 결과는 GBP에 저장되어 있으므로, REDO 작업이 필요 없고, 로그의 병합 또한 필요 없다는 장점을 가진다. 그러나, GBP를 구현하기 위해서 IBM Parallel Sysplex라는 특수한 하드웨어가 필요하며, 이 하드웨어 상에서만 동작이 가능하다는 단점이 있다.

2.3 Oracle 9*i* Real Application Clusters (ORAC)

ORAC에서 구현한 캐쉬 연합 정책은 이전의 Oracle Parallel Server의 캐쉬 관리 정책을 개선하여 노드 N_1에서 갱신된 페이지 P_1이 디스크에 저장되지 않은 상태에서 다른 노드 N_2로, N_2에서 다시 다른 노드 N_3로의 전송이 가능하다[7]. 이때 N_1, N_2, N_3가 P_1을 모두 캐싱함으로써 회복에 있어서 성능 개선을 기대할 수 있다. 단일 노드의 회복과정을 구체적으로 살펴보면, 어떤 노드 N_k에서 고장이 발생했을 때, 회복을 위해서 그 이전 노드인 N_{k-1}의 버퍼에서 P_1을 전송 받은 후, N_k의 분실된 갱신에 대해서만 REDO를 적용하게 된다. 이 과정에서 ARIES/SD와는 달리 통합 로그 스캔 과정이 불필요하게 된다는 장점을 가진다.

그러나, $1 \le i \le k$인 임의의 노드 N_i에서 갱신된 P_1이 버퍼에서 교체될 경우, P_1의 소유자인 N_k의 버퍼에서 P_1을 디스크에 저장해야 한다. 그 결과, 트랜잭션 처리 중에 여러 노드 중에서 버퍼 교체가 일어나는 노드가 있을 경우 무조건 디스크에 저장하므로, 디스크 저장 빈도수가 ARIES/SD보다 많아진다는 단점을 갖는다.

3. 제안 기법

본 논문에서 제안한 기법의 개념은 노드 N_1에서 갱신된 페이지 P_1을 다른 노드 N_2로, N_2에서 다시 다른 노드 N_3로 전송 할 때, N_3와 N_2의 쌍만을 관리한다는 것이다. 그 결과, N_1의 버퍼에서 P_1이 교체될 경우 N_3에서 P_1이 디스크에 저장될 필요가 없으므로 ORAC의 단점인 빈번한 디스크 저장 오버헤드를 해결할 수 있다. 뿐만 아니라, N_3의 고장시 N_2에 캐싱된 P_1을 이용하여 회복을 수행함으로써, 단일 노드 고장의 경우 로그 병합을 피할 수 있다는 장점을 갖는다.

본 논문에서는 N_1에 캐싱된 P_1을 처리하는 방식에 따라 무효화 기법과 검사 기법을 제안한다. 제안한 기법들은 ARIES/SD의 "Fast Scheme"에서 사용한 페이지 전송 및 소유권 개념을 지원한다. 즉, 페이지 P_1의 현재 소유자 노드가 N_2이고, 다른 노드 N_3가 P_1을 갱신하고자 할 경우를 가정하자. N_3는 Global Cache Service (GCS)에 데이타 전송을 요청하고, GCS는 요청을 N_2에게 전송한다. 요청을 받은 N_2는 데이타를 전송하고 이전 소유자가 된다. 데이타를 전송받은 N_3는 현재 소유자가 되고, 전송된 데이타의 로크에 관련된 정보를 GCS에게 전송한다[4].

3.1 무효화 기법

무효화 기법에서는 GCS가 로크 관련 정보를 전송 받은 후, 현재의 소유자와 이전 소유자의 정보의 쌍([Owner, Prev Owner])을 관리하게 되고 이전 소유자의 이전 소유자에게 무효화 통보를 하게 된다. 이렇게 통보를 함으로써 그 노드는 더 이상 관리 대상이 아님을 알게되고, 버퍼내의 캐시 버전을 무효화 할 수 있게 된다. 따라서, 그 노드에서 버퍼교체가 일어나더라도 소유자 노드에서 디스크에 저장할 필요는 없다. 무효화 기법의 단계를 그림 1에 정리하였다.

무효화 기법은 현재 소유자와 이전 소유자를 제외한

데이타는 N_1, N_2, N_3순으로 전송된다고 가정한다

1. 요청자 N_3는 GCS로 페이지 P_1 전송 요청
2. GCS는 현재 소유자 N_2로 전송 요청을 전송
3. N_2는 N_3에게 데이타 전송
   N_2 : 현재 소유자에서 이전 소유자로 변경
   N_3 : 요청자에서 현재 소유자로 변경
4. N_3는 GCS로 로크 정보 전송
5. GCS는 [Owner=N_3, Prev Owner=N_2]정보를 관리
6. GCS는 N_2의 이전 소유자인 N_1으로 무효화 통보
7. N_1의 P_1은 무효화 상태가 됨

그림 1. 무효화 기법의 의사코드

노드의 경우는 이미 무효화 통지를 받았으므로, 버퍼교체가 일어나더라도 ORAC과 같은 디스크 저장 오버헤드가 발생하지 않는다는 장점을 가진다. 그러나, 트랜잭션 처리 과정에서 무효화를 위한 1회의 메시지가 추가적으로 더 필요하다. 일반적으로 노드들 간의 통신 오버헤드는 디스크 액세스 오버헤드에 비해 훨씬 작은 것으로 알려져 있으므로, 메시지의 추가로 인한 오버헤드는 빈번한 디스크 액세스 오버헤드에 비해 훨씬 작다고 볼 수 있다. 로크 정보의 크기와 관리 측면에서 살펴보면, 무효화 기법의 경우는 페이지 버전을 캐싱하고 있는 모든 노드들의 정보를 유지하는 ORAC에 비해서 유지해야 될 로크 정보의 크기가 줄어든다. 뿐만 아니라, 캐싱된 페이지에 대해 최대 2개의 버전만 버퍼에 유지되므로, 버퍼 사용 효율 측면에서도 ORAC보다 우수하다는 장점을 갖는다.

3.2 검사 기법

무효화 기법과 달리 검사 기법은 페이지의 소유권 변경시 기존의 이전 페이지에 대한 무효화 메시지를 전송하지 않는다. 그 대신 노드 N_i에서 버퍼 교체가 일어날 경우 N_i는 교체될 페이지 P_i가 자신에 의해 갱신된 페이지일 경우 GCS에게 교체 통보 메시지를 전송한다. GCS는 N_i가 교체 통보한 페이지 P_i의 현재 소유자나 이전 소유자가 N_i인지를 검사한다. N_i가 P_i의 현재 소유자일 경우 GCS는 N_i에게 교체허용 메시지를 전송한 후, Pi의 소유권 정보를 삭제한다. N_i가 P_i의 이전 소유자일 경우에는 P_i의 현재 소유자에게 P_i의 저장 요청 메시지를 전송한 후, GCS는 P_i의 이전 소유자 정보를 삭제한다. N_i가 P_i의 현재/이전 소유자로 등록되지 않은 경우에는 N_i에게 교체 허용 메시지를 즉시 전송한다. 검사기법의 단계를 그림 2에 정리하였다.

```
단계 1~5는 그림 1과 동일.
6. if (Ni에서 Pi의 버퍼교체 발생 & Ni가 Pi 갱신)
        Ni는 교체 통보 메시지를 GCS에게 전송
7. GCS는 소유자 정보를 검토
8. if (Ni = Pi의 현재 소유자)
        Ni에게 교체 허용 메시지 전송
        & 소유권 정보 삭제
   else if (Ni = Pi의 이전 소유자)
        Pi의 소유자에게 Pi 저장 요청 메시지 전송
        & Pi의 이전 소유자 = ∅
   else
        Ni로 버퍼 교체 허용 메시지 전송
```

그림2. 검사 기법의 의사코드

무효화 기법과 마찬가지로 검사 기법도 ORAC에서 발생하는 빈번한 디스크 저장 오버헤드를 줄일 수 있으며, 무효화 기법에서 발생하는 무효화 메시지 전송 오버헤드를 줄일 수 있다.

뿐만 아니라, 교체되기 전까지는 오래된 버전들이 버퍼에 계속 캐싱될 수 있으므로 ORAC에서 지원하는 'old version read' 기능을 지원할 수 있다는 장점을 갖는다.

3.3 데이타베이스 회복 과정

본 절에서는 데이타베이스 회복 과정에 대해 간단히 설명한다. 단일 노드 고장의 경우, 고장 후 재실행되는 노드 N_1은 GCS에 자신이 소유자로 등록된 페이지가 있는지 확인하여, 회복해야할 페이지를 결정한다. 회복할 페이지 P_1이 있을 경우 P_1의 이전 버전을 이전 소유자에게 전송 요청을 한다. 이전 소유자가 없을 경우에는 디스크에서 P_1을 액세스한 후, N_1의 로그에 대한 REDO 작업을 수행한다. N_1은 전송 받은 이전 버전과 자신의 로그를 이용하여 페이지를 최신의 상태로 회복한다. 복합 고장의 경우, 체크포인팅과 로그를 이용하여 회복 시점을 결정한다. 복합 고장의 경우에도 모든 페이지를 디스크로부터 읽어 올 필요가 없고, 이전 버전을 가지고 있는 노드에서 전송이 가능하며 이전 버전을 이용한 회복을 수행한다. 특히 검사 기법의 경우에는 오래된 버전들이 다른 노드의 버퍼에 계속 캐싱될 수 있으므로, [3]에서 제안한 이전 버전들을 이용한 빠른 데이타베이스 회복 알고리즘을 적용할 수 있다.

4. 결론 및 향후 과제

본 논문에서는 DSS에서 각 노드의 버퍼에 캐싱된 페이지들의 일관성을 유지하기 위하여 제안된 ORAC의 캐쉬 연합 기법을 개선한 버전 관리 기법을 제안하였다. 본 논문에서 제안한 버전 관리 기법의 장점은 다음과 같다. 첫째, 단일 노드의 회복에 있어서 로그의 병합이 필요하지 않으므로, 빠른 회복이 가능하며 ARIES/SD에 비해 성능이 향상된다. 일반적으로 단일 노드의 고장이 복합 노드의 고장보다 빈번하므로, 단일 노드의 회복에서 우수한 성능을 보이는 것은 매우 중요하다. 둘째, 트랜잭션 처리과정에서 ORAC에서 발생하는 빈번한 디스크 저장의 오버헤드가 없다. 따라서, 빠른 트랜잭션 처리를 기대할 수 있으며, 빠른 회복 또한 가능하다. 본 논문의 향후 과제는 제안한 알고리즘을 구현하여, 기존의 알고리즘들과의 성능을 비교하는 것이다.

5. 참고문헌

[1] D. Dewitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems," *Comm. ACM*, Vol. 35, No. 6, pp.85-98, 1992.

[2] P. Valduries, "Parallel Database Systems: Open Problems and New Issues," *Distributed and Parallel Databases*, Vol. 1, No. 2, pp.137-165, 1993.

[3] H. Cho, "Database Recovery using Incomplete Page Versions in a Multisystem Data Sharing Environment," *Information Processing Letters*, Vol. 83, No. 1, pp.49-55, 2002.

[4] C. Mohan and I. Narang, "Recovery and Coherency Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment," in: *Proc. 17th VLDB Conf*, pp.193-207, 1991.

[5] E. Panagos, A. Biliris, H. V. Jagadish, and R. Rastogi, "Client-Based Logging for high Performance Distributed Architectures," in: *Proc. ICDE*, pp.344-351, 1996.

[6] *IBM DB2 Data Sharing : Planning and Administration*, IBM, SC26-9935-01, 2001.

[7] *Oracle 9i Real Application Clusters*, Oracle Corp., part A89867-02, 2001.

[8] E. Rahm, "Recovery Concepts for Data Sharing Systems," in: *Proc. 21st Int. Conf. on Fault-Tolerant Computing*, pp. 109-123, 1991.