주기억 장치 데이터베이스시스템에서 효율적인 POLUG 회복 기법

김 명 근*, 조 숙 경*, 정 광 철*, 이 순 조**, 배 해 영*
* 인하대학교 전자계산공학과
** 서원대학교 전자계산공학과
g1991247@inhavision.inha.ac.kr

Efficient POLUG Recovery Method in Main Memory Database System

Myung-Keun Kim*, Sook-Kyoung Cho*, Kwang-Chul Jung*, Soon-Jo Lee**, Hae-Young Bae*
* Dept. of Computer Science, Inha University
** Dept. of Computer Science, Seowon University

요약

MMDBMS(Main Memory Database Management System)에서 사용되는 갱신기법 중 퍼지 검사점을 지원하는 직접갱신 기법은 많은 양의 로그를 필요로 하며 회복 과정이 복잡하다는 단점이 있다. 본 논문에서는 퍼지 검사점을 지원하면서 저장해야 할 로그의 양을 줄이고 회복 과정이 단순한 회복 기법을 제안한다. 제안하는 회복 기법은 로그의 양을 줄이기 위해 철회된 트랜잭션의 로그를 비휘발성 메모리에 저장하지 않는 시스템 구조와 많은 양의 메모리를 사용하지 않는 POLUG(Page Oriented Logical Undo loG)를 사용하여 퍼지 검사점을 지원할 수 있는 장점을 가진다.

1. 서론

주기억 장치 상주 데이타베이스 시스템은 전체 데이타베이스가 주기억 장치에서 관리 되어지는 시스템을 말한다[4]. 주기억 상주 데이타베이스 시스템은 항상 데이타베이스가 주기억 장치에 상주하고 있기 때문에 DRDBS(Disk Resident DataBase System)보다 빠른 응답시간과 효율적인 트랜잭션 처리능력을 가진다[4]. 그러나 시스템이 붕괴되면 주기억 장치의 모든 데이터를 잃어 버리게 된다. 따라서 데이터베이스의 일관성을 유지하기 위한 회복기법은 매우 중요하다[2,3,4].

기존의 MMDBS 에서의 데이타베이스 갱신 기법으로는 섀도우 갱신 기법[2]과 직접 갱신 기법[3]으로 크게 구분할 수 있다. 섀도우 갱신 기법은 섀도우 메모리로의 복사가 이루어져야 하기 때문에 연산 속도가 느리며, 섀도우 메모리를 유지하기 위하여 많은 양의 메모리를 필요로 하는 단점이 있다. 직접 갱신 기법은 재수행 로그와 취소 로그를 필요로 하므로 로그의 양이 많아지고, 회복 과정이 섀도우 갱신 기법에 비해 복잡한 단점이 있다. 이러한 단점을 극복하고자 재수행 로그만을 기록하여 로그의 양을 줄이는 회복 기법이 연구되어 졌다[3,6]. 그러나 이러한 시스템은 검사점 수행 시 트랜잭션이 완료나 철회하기 전에 변경된 페이지를 디스크로 반영 못하는 단점이 있다.

본 논문에서는 퍼지 검사점을 지원하면서 저장해야 할 로그의 양을 줄이기 위한 POLUG를 이용하여 회복 과정이 단순한 회복 기법을 제안한다. POLUG 는 페이지의 논리적 구조에 대한 회복을 지원하기 위해 저장해야 할 로그를 말한다. 이러한 로그정보는 페이지의 물리적인 위치를 가지고 회복을 수행하는 물리적인 로그와는 달리 페이지 식별자와 연산 종류를 가지고 회복을 지원하는 논리적인 로그이다.

본 논문의 회복 구조는 트랜잭션 철회 시 POLUG 에 대한 역 연산만으로 데이타베이스를 회복 할 수 있기 때문에 철회 작업이 단순해 진다. 또한 전역로그와 트랜잭션별 로그를 나누어 관리함으로써 비 활성 메모리에 완료된 트랜잭션의 재수행 로그만을 기록하여 로그의 양을 줄이고자 한다.

본 논문의 구성 다음과 같다. 제 2 장에서는 관련 시스템에 대하여 설명하고, 제3 장에서는 로그 레코드의 구조와 POLUG 회복기법을 사용하기 위한 트랜잭션의 로그 기록과정과 POLUG 회복 처리 과정을 설명한다. 마지막 제4 장에서 결론 및 향후 연구 방향을 기술한다.

2. 관련 연구

현재까지 주기억 상주 데이타베이스 시스템의 회복 문제에 대한 연구는 상당히 많이 이루어져 왔다.

IBM 에 의해 구현된 최초의 주기억 상주 데이타베이스 시스템인 IMS FastPath 에서는 디스크로의 출력을 줄이기 위해 완료 그룹(commit group)의 개념을 이용하였다. 검사점 수행 기법은 TCC(Transaction Consistency Checkpoint) 방법이 이용되며 주기억 장치 데이타베이스

* 본 연구는 정보통신부의 대학 S/W 연구센터 지원사업의 연구 결과임.

처리와 동시에 수행되는 비동기 IMS 태스크에 의해 처리된다[1].

프린스턴 대학의 MMM(Massive Memory Machine) 프로젝트에서 제안한 회복 기법으로 HALO(HArdware LOgging device)가 있다. HALO 는 사전 이미지와 사후 이미지를 가진 로그 데이터를 만들어 비휘발성의 주기억장치에 먼저 기록한 후, 시간이 허용할 때 디스크 상의 로그 파일에 기록한다. 그리고, 검사점 수행 방법으로는 ACC(Action Consistency Checkpoint)방법이 이용된다[2].

MARS 는 주기억장치 섀도우 페이지, 사전 완료(pre-commit), 트랜잭션, 그리고 임의 접근 로그와 자동 검사점 수행 및 회복 작업을 전담하는 프로세서를 이용하는 회복 관리 기법을 제안하였다. 검사점 수행 방법은 TCC 형태의 검사점 생성을 위하여 회복 전담 프로세서 상에서 수행되는 자동 검사점 수행 기법이다[5].

System M 은 프린스턴 대학에서 개발한 DBMS testbed 이다. System M 은 로그 처리과정을 효과적으로 하기 위해 사전 완료 및 그룹 완료를 지원하며 퍼지 검사점, black/white 검사점, copy-on-update 검사점 방식 등의 다양한 검사점 방식과 로깅 기법을 사용한다[4].

Dali 는 AT&T 에서 개발한 주기억 상주형 데이터베이스 시스템이다. Dali 의 회복 기법은 트랜잭션별로 로그를 관리하며 철회된 트랜잭션에 대한 로그를 기록하지 않음으로써 디스크 접근 시간을 줄이고자 했다. 검사점 방식은 ping-pong 검사점 방식을 사용하였으며 WAL 에 의한 위반 문제를 제거 하였다[3].

3. POLUG 를 이용한 회복 기법

3.1 로그 레코드의 구조

본 논문에서 제안하는 회복 기법은 전역 로그 버퍼와 트랜잭션별 로그 버퍼를 사용한다. [표 1]은 트랜잭션별 로그 버퍼에서 사용하는 로그 레코드의 형식을 나타낸다.

[표 1] 전역 로그 레코드의 형식

Log Type	PYSICAL/BT/ET/BC/EC/POLUG
TxID	Transaction Identifier
LSN	Log Sequence Number
RID/FID/PID	Record ID/Field ID/Page ID
Image	After image/commit or abort/ INSERT/DELETE/UPDATE

LSN 은 순차적인 로그 식별자로써 로그 버퍼마다 고유한 값을 유지한다. PYSICAL 은 물리적인 로그로써 갱신한 이미지를 저장하며, 트랜잭션 식별자(TxID)와 레코드 식별자(RID) 및 필드 식별자(FID)를 갖는다. BT 는 단지 트랜잭션이 시작 되었음을 로그에 기록하는 것이다. ET 는 트랜잭션의 끝을 의미하며, 완료(commit)와 철회(abort)의 2 가지 종류가 있다. BC 는 퍼지 검사점 방식에서 시작을 의미하며, EC 는 검사점의 완료를 의미한다. POLUG 는 페이지 헤더의 일치성을 보장하기 위한 일종의 논리적인 로그이며 트랜잭션 식별자와 POLUG 종류 및 페이지 식별자가 저장되어 진다. [표 2]는 전역 로그 레코드의 형식을 나타낸다.

[표 2] 트랜잭션별 로그 레코드의 형식

Log Type	UPDATE/INSERT
TxLSN	Transaction Log Sequence Number
RID/FID	Record ID/Field ID
Image	After image
NRLSN	Next Record TxLSN

트랜잭션별 로그버퍼는 재수행 로그 레코드 만을 저장한다. UPDATE 는 갱신연산에 대한 재수행 로그 레코드를 나타내며 트랜잭션 내에서 유일한 로그 식별자인 TxLSN 을 갖으며 레코드 식별자/필드 식별자 및 같은 레코드에 갱신한 다음 로그 레코드 식별자를 나타내는 NRLSN 을 갖는다. NRLSN 은 삽입이나 갱신이 이루어진 레코드에 대한 검색 시 임시 공간에서의 재수행 과정을 통해 검색을 지원하기 위한 것이다.

3.2 트랜잭션의 로그 기록 과정

본 논문의 회복 기법을 지원하기 위한 트랜잭션의 로그 기록 과정은 [알고리즘 1]과 같다.

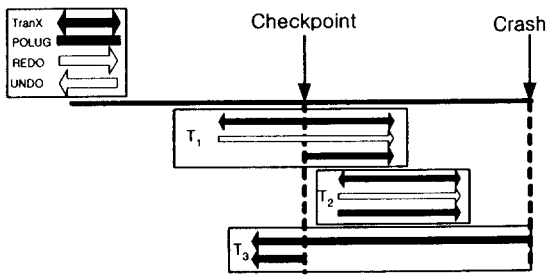
[알고리즘 1] 로그 기록 알고리즘

```
Update(Ti, Record) // Ti: 트랜잭션 식별자,Record: 연산할 레코드
{
  Do_Lock(relation name)// 릴레이션에 락을 요청
  LSN = Write_Log(Ti LB, Record.TxLSN)
                    // Ti 로그 버퍼에 재수행 로그를 기록
  Write_Log(POLUG,GLB)  // POLUG 를 전역로그로 복사
  Record.TxLSN = LSN    // LSN 을 레코드 구조체에 저장
  Record.TxID = Ti      // Ti 를 레코드 구조체에 저장
}
TranX_End(Ti, Type) // Ti: 트랜잭션 식별자,Type: 완료 or 철회
{
  if( Type == commit) {
    Copy_Log(Ti LB,GLB) // Ti 로그버퍼에서 전역로그 버퍼로 복사
    Redo(Page,Ti LB)    // 로그를 페이지에 재수행
    Write_Log(Eti(commit),GLB) // ETi 를 기록
    Release_Lock(Ti)    // 락 해제
  }
  else  { // Type == abort
    Delete_Log(Ti LB)   // Ti 로그 버퍼를 지운다
    Undo(POLUG)         // POLUG 의 취소 연산
    Write_Log(ETi(abort),GLB) // ETi 를 기록
    Release_Lock(Ti)    // 락 해제
  }
}
```

갱신 연산(삽입 연산)이 일어나면 릴레이션에 락을 요청하고, 레코드에 기록해야 할 이미지를 로그로써 휘발성 메모리에 있는 트랜잭션별 로그 버퍼에 기록한다. 기록된 로그 레코드의 식별자를 레코드의 TxLSN 에 저장한다. 이는 삽입한 트랜잭션이 검색연산을 했을 경우 갱신된 이미지는 로그 레코드가 가지고 있기 때문에 이를 찾기 위해서는 해당 레코드에 갱신한 LSN 을 기록하여야 한다. 트랜잭션이 성공적으로 완료하면 트랜잭션별 로그 버퍼에서 전역로그로 기록한다. 그리고 재수행 로그를 해당 페이지에 실제적으로 반영한 후, 전역 로그에 트랜잭션이 완료를 나타내는 ET 를 기록한다. 마지막으로 걸었던 락을 해제한다. 트랜잭션이 철회했을 경우 트랜잭션별 로그 버퍼영역을 메모리에서 삭제하고, 전역 로그 버퍼의 POLUG 에 대해 취소 연산을 수행한다. 이는 물리적인 회복이 아니기 때문에 다른 로그와는 달리 연산이 빨리 수행되어지며, 또한 많은 양의 메모리를 필요로 하지 않는다. 다음으로 전역로그에 트랜잭션의 철회를 나타내는 ET 를 기록하고 난후 락을 해제한다. 본 회복 기법은 락의 단위를 릴레이션으로 한정짓는다. 릴레이션 단위의 락은 레코드의 TxLSN 필드에 대한 회복을 하지 않고 트랜잭션 식별자의 비교만으로 TxLSN 이 유효한 지를 판단할 수 있기 때문에 시스템 회복의 성능을 높일 수 있다.

3.3 POLUG 를 이용한 회복 과정

제안된 회복 관리 기법은 트랜잭션의 완료 시에만 페이지에 반영함으로 고장이 발생했을 때 트랜잭션 철회작업이 단순해 진다. 트랜잭션 철회 시 POLUG 에 대해서만 철회연산을 수행하고, 재수행 로그는 버퍼에 지우기만 하면 된다. 시스템 재시작 시 디스크에 저장된 데이타베이스를 메모리에 적재한다. 그리고 전역로그와 로그 파일에서 가장 최근에 완료된 검사점의 시작 로그를 찾기 위해서 역 방향으로 탐색하면서 재수행 할 트랜잭션들과 검사점 이후에 활성 상태에 있던 트랜잭션의 리스트를 구성한다. [그림 1]은 고장 후 트랜잭션의 상태에 따른 회복 과정을 보여준다.



[그림 1] 고장 후 트랜잭션의 상태에 따른 회복 과정

검사점 이후에 완료한 트랜잭션[T_1,T_2]은 전역로그를 바탕으로 재수행 작업을 한다. 이러한 경우에는 2 가지 종류가 있다. 첫번째는 검사점이 트랜잭션 사이에 존재하는 경우[T_1]이고, 또 하나는 트랜잭션 전 과정이 검사점 이후에 존재하는 경우[T_2]이다. 전자의 경우에는 검사점 이전의 POLUG 가 반영한 페이지는 디스크의 데이타베이스에 기록되었기 때문에 재수행을 하지 않는다. 하지만 검사점 이후의 POLUG 에 대해서는 재수행 연산을 수행한다. 후자의 경우는 POLUG 와 물리적인 재수행 로그에 대하여 재수행을 한다. 검사점 이후에 완료하지 않은 트랜잭션[T_3]은 트랜잭션 철회를 해야 한다. 검사점 이후의 작업은 디스크에 반영이 안되었기 때문에 회복은 하지 않아도 된다. 하지만 검사점 이전의 POLUG 가 반영한 페이지가 디스크에 기록 되었기 때문에 POLUG 에 대한 취소연산을 해야 한다.

4. 결론

본 논문에서는 주기억 상주 데이타베이스 시스템을 구성하는 회복 관리자를 구현하기 위한 기법을 제안하였다. 제안된 기법은 재수행 로그만을 기록하는 기존의 회복 관리 기법이 퍼지 검사점 방식을 사용할 수 없는 단점을 개선하기 위하여 POLUG 의 개념을 도입하였고, 또한 로그의 양을 줄이기 위해 로그 버퍼를 두 가지 종류의 메모리에 나누어서 사용하였다. POLUG 는 논리적인 로그이므로 많은 양의 로그 버퍼를 사용하지 않기 때문에 로그 버퍼가 가득 참으로써 발생할 수 있는 디스크 입출력을 줄일 수 있으며 로그 버퍼를 나누어서 관리함으로써 철회된 트랜잭션들의 재수행 로그를 디스크에 저장할 필요가 없기 때문에 로그의 양을 줄일 수가 있다는 장점을 가진다.

참고문헌

[1] D. Gawlick, D. Kinkade, " Varieties of Concurrency Control in IMS/VS Fast Path" , Data Engineering Bullitin, 8, 2, June, 1985

[2] H. Garcia-Molina, K. Salem, " High Performance Transaction Processing with Memory Resident Data ", International Workshop on High Performance Transaction Systems, Paris, December, 1987.

[3] H. V. Jagadish, D. Lieuwen, R. Rastogi, and A. Silberschatz, " Dal : A High Performance Main Memory Storage Manager, " Proceedings of the 20th VLDB Conference, 1994.

[4] K. Salem, and H. Garcia -Molina, " System M: A Transaction Processing Testbed for Memory Resident Data " IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, 1990

[5] M. H. Eich, " MARS: The Design of a Main Memory Database Machine" , Proceedings of the International Workshop on Database Machines, October, 1987.

[6] 조숙경, 김경배, 이순조, 임기욱, 배해영 " 실시간 데이터베이스 시스템에서 고장 허용 한계 시간 제약을 갖는 회복 관리 기법의 설계" , Proceedings of The 20th KISS Spring Conference, 1993