논문 00-25-11B-12 한국통신학회논문지 '00-11 Vol.25 No.11B

클라이언트/서버 데이터베이스 시스템에서 역방향 로그 분석을 이용한 로그 관리

정회원 이 찬 섭*, 박 용 문**, 고 병 오***, 최 의 인*

Log Management Using Backward Log Analysis in Client-Server Database System

ChanSub Lee*, YongMun Park**, Byoung-oh Koh***, EuiIn Choi* *Regular Members*

요 약

기존 데이터베이스 시스템에서 사용되는 회복 기법들은 시스템 파손 시 빠른 회복을 지원하기 위해서 물리적 로깅(physical logging)을 사용한다. 그러나 이런 기법들을 클라이언트/서버 환경에 그대로 적용할 경우에는 여러 가지 문제점이 발생된다. 물리적 기법의 경우에는 로그 분석 시 before-image와 after-image의 중복이 발견된다는 문제점이 있으며, 기존의 대부분 회복 기법들은 시스템 파손 시 전방향(forward)으로 로그를 분석함으로써 불필요한 회복 동작이 존재할 수 있다. 또한 시스템 회복 시 로그 접근 횟수의 증가로 인해 회복 속도가 늦어지는 문제점이 있다. 이 논문에서는 이런 문제점을 해결하고 클라이언트/서버 환경에 적합한 회복 기법을 제안하기 위해 중복된 before-image를 제거하고 재수행 전용 로그 레코드(redo-only log record)만을 로그에 기록함으로써 로깅 오버헤드를 감소시키면서 로그 분석 시간을 감소시킨 역방향 로그 분석 기법을 제안하였다. 또한 로그 분석 시 유지해야 하는 자료구조의 오버헤드를 최소화했다. 마지막으로 제안된 기법과 기존의 기법을 비교 분석하였다.

ABSTRACT

The recovery methods of the existing database system use the physical logging technology for the fast recovery when the system is destroyed. However, there can be many problems occurred when these methods are deployed to the client/server system directly. In case of a physical logging technology, we can see an overlapping between before-image and after-image at the log analysis stage. Also, there can be an unnecessary recovery action generated because the log analysis performed by the most recovery methods is done by forwarding direction scheme. Besides that, there is a delay problem on the recovery process when the number of log access is increased. In this paper, we propose a backward log analysis method, which is suitable for the client/server environment. The method can minimize both the log analysis time and the logging overhead by recording a redo-only log record to the log. And it can remove an overlapped before-image at the same time. Also, the overhead of the data structure that must be maintained at the log analysis is minimized by this method. Finally, the proposed method is compared with an existing method.

Ⅰ. 서 론

최근 들어 고성능 워크스테이션의 등장, 통신장비의 발달과 더불어 다중 사용자들의 요구 증가로 인하여 대부분의 시스템을 클라이언트/서버 환경으로 확장하는 추세이고, 데이터베이스 시스템도 클라이언트/서버 데이터베이스 시스템에 대한 연구가 활발히 진행되고 있다. 클라이언트/서버 데이터베이스

* 한남대학교 공과대학 컴퓨터공학과 ** 한국전자통신연구원
*** 공주교육대학교 실과교육과
논문번호: 00229-0623, 접수일자: 2000년 6월 23일

시스템에서 트랜잭션의 빠른 처리와 응답을 지원하기 위해서 가능한 디스크 접근을 최소화한다. 이런 시스템에서 시스템 파손(system crash)이 발생하면 안정된 데이터베이스(stable database)에 반영하지 못한 많은 데이터들의 손실이 생길 수 있기 때문에 손실된 데이터베이스를 시스템 파손 직전의 일관된 상태로 복구하는 회복 기법이 절실히 요구된다[1,6,7].

디스크 접근을 최소화하기 위해서 steal과 no force를 지원하는 데이터베이스 시스템에서 빠른 회복을 지원하기 위해서는 트랜잭션 갱신 연산의 undo/redo 정보(함수 이름, 함수 인자 등)를 기록하는 논리적 로깅(logical logging)보다 트랜잭션의 갱신 연산을 적용하기 이전의 사전 이미지(before-image)와 이후의 사후 이미지(after-image)를 기록하는 물리적 로깅을 사용한다. 하지만, 물리적 로깅을 사용하면 로그 레코드의 크기가 커서 로깅 시 오버헤드 크다는 단점이 있다[2,3].

본 논문은 클라이언트/서버 데이터베이스 시스템의 회복 기법에 대한 연구로써, 시스템 파손 시 역방향 로그 분석(backward analysis log)을 수행하면서 손실된 각 페이지에 대해 한번의 재수행만 수행하는 빠른 회복 기법을 제안했다. 그리고 물리적 로깅의 오버헤드를 최소화하기 위해서 중복된 before-image를 제거했다. 또한 별도의 버퍼를 준비하지 않고 중복된 사전 이미지를 제거하여 재수행 전용 로그 레코드만을 로깅 함으로써 로깅 오버헤드를 감소시켰으며 로그 분석 시 소요 시간을 감소시켰다. 시스템 파손 시 재수행 전용 로그를 가지고 한 번의 역방향 로그 분석을 통하여 일관된 데이터베이스 상태로 복구하는 회복 기법을 제안함으로써 불필요한 재수행 동작을 제거했고, 로그 분석 동안에 동적인 완료 트랜잭션 리스트를 구성하여 오버헤드를 최소화했다.

제 2장에서 기존 회복 기법들의 문제점들을 제기하고, 제 3장에서는 기존 회복 기법의 문제점을 해결하기 위해 제안한 회복 기법으로 트랜잭션 처리 및 로깅 과정, 검사점, 로그 절단, 시스템 회복 동작을 제안한다. 제 4장에서는 트랜잭션 수행 예제와 알고리즘 분석을 통하여 제안한 회복 기법과 기존 회복 기법을 비교 분석한다. 제 5장에서는 결론 및 추후 연구 방향에 대하여 기술한다.

Ⅱ. 기존 회복 기법들의 문제점

이 장에서는 기존 회복 기법과 로깅 기법을 적용했을 때 문제점인 중복된 사전 이미지, 기존 전방향 로그 분석 시 문제점, 기존 역방향 로그 분석 시 추가된 자료구조의 오버헤드에 대해 설명한다.

1. 기존 클라이언트/서버 환경에서의 회복 기법

1.1 ARIES

ARIES(Algorithm for Recovery and Isolation Exploiting Semantics)는 단일 시스템에서 가장 널리 알려진 데이터베이스 회복 기법들 중에 하나이다[12,13]. 시스템 정상 운영 시 이 기법은 트랜잭션의 부분 철회(partial rollback)와 전체 철회(total rollback)를 제공하고, 로그 레코드의 LSN(Log Sequence Number)과 페이지 헤더 내의 page_LSN을 이용하여 WAL 규약에 따라서 논리적 로깅을 한다. 그리고 페이지 갱신 상태 정보(DPT; Dirty Page Table)와 트랜잭션 상태 정보를 유지 및 관리한다[8,9,10].

시스템 파손 시는 분석 단계, 재수행 단계, 철회 단계를 거쳐 시스템을 회복한다. 재수행 단계에서는 repeating history 패러다임을 사용하여 무조건 재수행을 하고, 트랜잭션 철회 시 다음 철회할 NxtLSN 필드 정보를 가지고 있는 보상(compensation) 로그 레코드를 로그에 기록하기 때문에 회복 동작 중 반복적인 시스템 파손으로 발생할 수 있는 compensating/duplicate compensation 문제를 해결한다.

대부분 클라이언트/서버 회복 기법들은 ARIES 기법을 확장하여 사용하고 있다. 다음절에서 설명하고 있는 ESM/CS, ARIES/CSA, EOS를 이용한 회복 기법도 이에 속한다.

1.2 ESM/CS

ESM/CS는 Winsconsin 대학에서 ARIES를 클라이언트/서버 환경으로 확장한 클라이언트/서버 EXODUS 저장 관리기(Client/Server EXODUS Storage Manager)에서 사용되는 회복 기법이다. 다음은 ESM/CS에서 기존의 ARIES를 확장한 부분에 대해서 기술한다[5,12].

ARIES는 단일 시스템에서의 회복 기법이므로 순차적으로 트랜잭션의 로그 레코드를 생성하여 WAL 규약에 따라서 로깅할 수 있지만, ESM/CS는 여러 클라이언트들에서 트랜잭션들을 수행함에 따라서 생성된 로그 레코드를 서버로 전송하여 로깅해야 하기 때문에 ARIES의 LSN을 이용하여 WAL 규약을 보장할 수 없다. 이를 해결하기 위해서 ESM/CS에서는 LSN이 지니는 성질 중 WAL 규약을 보장하

는데 필요한 성질만을 뽑아서 클라이언트에서 사용할 수 있는 LRC(Log Record Counter)와 페이지 헤더 내의 pageLRC를 이용한다.

클라이언트/서버 환경에서는 로그 레코드가 서버에 존재한다는 사실이 그 로그 레코드로 갱신된 데이터가 서버에 반영되었다는 사실을 의미하지는 않는다. 그러므로 트랜잭션 철회 시 기존의 ARIES에서 사용되었던 무조건 철회는 데이터베이스의 일관성을 위배할 수 있기 때문에 조건부 철회를 제공한다.

1.3 ARIES/CSA

ARIES/CSA는 ARIES를 클라이언트/서버 환경에서의 데이터베이스 회복 기법으로 확장했다[15]. 여기서도 ARIES의 LSN만으로는 WAL 규약을 보장하기가 어렵다. ARIES/CSA에서는 각 클라이언트에 의해서 지역적으로 부여되는 LSN을 사용한다. 이 로그 레코드의 LSN은 한 페이지에 해당하는 모든 레코드에 대해 유일하고 단조 증가하는 성질을 유지해야 하기 때문에 페이지를 갱신할 때 다음의 처리 과정이 필요하다.

① 갱신할 페이지의 pageLSN 조사
② 갱신 로그 레코드 작성하는 동안에 pageLSN을 로그 관리기에게 전달
③ 로그 관리기는 새로운 로그 레코드에 {1+page LSN, 1+Local_Max_LSN}값들 중에서 큰 값을 할당

위 처리 과정에서 Local_Max_LSN은 각 클라이언트들에서 로그 관리기에 의해서 부여되는 LSN 값들 중에 최고 값을 유지한다. 따라서 Local_Max_LSN을 유지하기 위해서 주기적으로 서버의 로그 관리기는 모든 클라이언트의 로그 관리기들과 통신을 통해 최고의 LSN 값을 유지한다.

1.4 EOS

EOS는 클라이언트/서버 환경에서 물리적 로깅을 이용한 재수행 회복 기법을 제공한다[11,14]. 앞서 살펴본 로깅 회복 기법과는 달리 EOS의 로그는 전역 로그 화일(global log file)과 개인 로그 화일(private log file)로 구성되어 있다. 또한, 각 트랜잭션마다 개인 로그 화일을 가지고 있다가 해당 트랜잭션이 완료할 때 해당 트랜잭션의 개인 로그 화일을 전역 로그에 추가한다. 즉, 로그에는 완료한 트랜잭션의 로그와 검사점 로그만을 기록함으로써 재수행 회복 기법을 가능하게 하지만 각 트랜잭션 완료까지 자신의 로그 레코드들을 개인 로그 화일에 유지해야 하는 오버헤드가 크다.

2. 로깅 기법에서의 문제점

2.1 중복된 사전 이미지

steal과 no force를 지원하는 시스템에서 물리적 로깅을 사용하면 트랜잭션의 갱신 연산 수행 시 갱신 연산에 대한 사전 이미지와 사후 이미지 모두 로그에 기록한다. 다음 예와 같이 한 페이지를 트랜잭션 T1, T2에 의해서 순차적으로 갱신된 경우에 표 1과 같은 로그 레코드가 생성된다.

- 트랜잭션 T1이 수행되고 완료된 뒤, T2가 수행되고 철회된다.
- 디스크에 반영된 페이지 P1의 값은 P1(0)이다.
- 트랜잭션 T1, T2가 순서대로 P1을 P1(1), P1(2)로 갱신한다.
- 로그 레코드의 상태는 표 1과 같다.

표 1. 로그 레코드 상태

LSN	trID	type	pgID	after-image	before-image
100	T1	'update'	P1	P1(1)	P1(0)
101	T1	'commit'	P1		
102	T2	'update'	P1	P1(2)	P1(1)

위 예제에서 트랜잭션 T1이 페이지 P1(0)을 P1(1)로 갱신한 뒤, T1이 완료되면 T1의 갱신 연산에 대한 before-image(P1(0))는 더 이상 필요 없다. 그리고 이어서 트랜잭션 T2가 페이지 P1(1)을 P1(2)로 갱신한 뒤, 트랜잭션 오류에 의해서 철회될 때 기존 회복 기법들은 T2에 대한 갱신 로그 레코드(LSN(102))의 before-image(P1(1))를 사용하여 철회한다. 하지만, 이 before-image(P1(1))는 이전에 수행한 T1의 after-image(P1(1))와 중복되어 있다. 이와 같이 중복된 사전 이미지를 기록하는 것은 로깅 오버헤드와 로그 유지비용 측면에서 낭비다.

제안한 회복 기법에서는 재수행 전용 로그 레코드(중복된 사전 이미지를 제거하고 사후 이미지만 로그에 기록)만을 로깅하고, 트랜잭션 철회 시 최근 완료한 트랜잭션에 대한 갱신 로그 레코드의 사후 이미지를 사용하여 해당 페이지를 회복하기 위해서는 다음 세 가지 문제들을 해결해야 한다. 첫째, 시

스템이 정상 운영 시 트랜잭션을 철회하기 위해서는 바로 최근에 해당하는 페이지를 갱신하고 완료한 트랜잭션에 대한 갱신 로그 레코드의 LSN을 기록해야 한다. 둘째, 위 예제에서 만일 T1이 철회될 경우에는 P1을 최근에 갱신하고 완료한 트랜잭션에 대한 갱신 로그 레코드가 반드시 존재해야 한다. 예를 들어, 시스템이 시작된 이후에 단 한번도 갱신되지 않은 페이지의 사후 이미지는 로그에 존재하지 않기 때문에 이런 페이지를 갱신하기 전에 별도의 사후 이미지를 로깅해야한다. 이 두 문제에 대한 해결은 3.1절에서 보여준다. 셋째, 제안한 로깅 기법은 페이지의 사후 이미지만을 로깅하기 때문에 시스템 회복 시 트랜잭션을 재수행하거나 철회할 때 완료한 트랜잭션에 대한 갱신 로그 레코드의 사후 이미지만을 이용해야 한다. 따라서 로그 분석 시 완료한 트랜잭션에 대한 갱신 로그 레코드의 사후 이미지만을 이용하여 손실된 페이지를 복구해야 한다. 이것은 2.2.2에서 기술하고 있는 역방향 로그 분석을 이용하여 해결 할 수 있다.

2.2 기존 전방향 로그 분석 시 문제점

전방향 로그 분석은 첫 번째 로그 레코드나 최근 수행한 검사점 로그 레코드부터 시작하여 마지막 로그 레코드까지 로그를 분석하는 것을 말하고, 역방향 로그 분석은 반대 방향으로 마지막 로그 레코드부터 시작하여 첫 번째 로그 레코드나 최근 수행한 검사점 로그 레코드까지 로그를 분석하는 것을 말한다.

전방향 로그 분석을 이용한 기존 회복 기법에서는 회복을 위해 로그 파일을 세 번(로그 분석을 위한 전방향 탐색, REDO를 위한 전방향 탐색, 그리고 UNDO를 위한 역방향 탐색) 접근한다. 이런 기법은 로그 분석 이후에 REDO나 UNDO를 위해 다시 로그에 접근해야하기 때문에 많은 로그의 입출력으로 회복 속도를 떨어뜨린다.

역방향 로그 분석을 이용한 기존 회복 기법에서는 로그를 역방향으로 분석하기 때문에 트랜잭션의 완료(commit) 로그 레코드가 로그에 존재하는지를 판단하여 트랜잭션의 완료 또는 철회를 바로 판단할 수 있다. 즉, 단 한번의 역방향 로그 분석을 이용하면, 트랜잭션이 완료되었는지 철회되었는지를 바로 판단할 수 있다. 따라서 한 번의 역방향 로그 분석을 통하여 REDO나 UNDO를 같이 수행할 수 있다.

그리고 다음 예제에서 알 수 있듯이 전방향 로그 분석을 하며 트랜잭션을 재수행하는 회복 기법들은 불필요한 재수행 동작이 존재한다.

- 트랜잭션 T1, T2, T3이 순서대로 수행하고 완료한 이후에 시스템 파손이 발생한다.
- 디스크에 반영된 페이지 P1의 값은 P1(0)이다.
- 트랜잭션 T1, T2, T3이 순서대로 P1을 P1(1), P1(2), P1(3)로 갱신한다.

- 로그 레코드는 표 2와 같다.

표 2. 로그 레코드 상태

LSN	tID	type	pgID	after-image	before-image
100	T1	'update'	P1	P1(1)	P1(0)
101	T1	'commit'	P1		
102	T2	'update'	P1	P1(2)	P1(1)
103	T2	'commit'	P1		
104	T3	'update'	P1	P1(3)	P1(2)
105	T3	'commit'	P1		

시스템 파손 시 전방향 로그 분석을 하며 REDO를 할 경우에는 LSN(100) 로그 레코드의 after-image(P1(1))를 이용하여 재수행하고, LSN(102) 로그 레코드의 after-image(P1(2))를 이용하여 재수행한다. 그리고 마지막으로 LSN(104) 로그 레코드의 after-image(P1(3))를 이용하여 재수행함으로써 시스템 파손 직전의 페이지 P1(3) 상태로 회복한다.

반면 역방향 로그 분석을 하며 REDO를 할 경우에는 LSN(104) 로그 레코드의 after-image(P1(3))를 이용하여 재수행함으로써 바로 페이지 P1(3) 상태로 회복할 수 있다. 즉, LSN(100) 로그 레코드와 LSN(102) 로그 레코드의 after-image를 이용하여 불필요한 재수행 동작을 제거한다.

2.3 기존 역방향 로그 분석 시 추가된 자료구조의 오버헤드

역방향 로그 분석을 이용한 기존 회복 기법에서는 시스템 회복 시 완료한 트랜잭션 리스트(COMMIT_LIST)와 회복된 페이지 리스트(UPDATED_LIST) 정보를 회복 완료 시점까지 유지 및 관리하기 때문에 메모리 낭비가 있고, 갱신된 페이지는 무조건 한번의 재수행 또는 철회를 수행하기 때문에 불필요한 회복 동작이 존재한다. 완료한 트랜잭션 리스트 정보는 역방향 로그 분석 시

'COMMIT' 로그 레코드를 분석할 경우 해당 트랜잭션 식별자를 추가하여 생성하는 코드(code) 부분만 있기 때문에 회복 완료 시점까지 이 정보를 유지한다. 하지만 역방향 로그 분석을 수행하다보면 완료한 트랜잭션 리스트에 포함된 트랜잭션의 로그 레코드가 더 이상 나오지 않는 경우가 있다. 즉, 완료한 트랜잭션 리스트 정보에는 더 이상 필요 없는 엔트리가 존재하게 된다. 제안한 회복 기법에서는 이 문제를 3.1절에서 해결하고 있다. 그리고 회복된 페이지 리스트 정보는 기존 회복 기법에서 해당 갱신 로그 레코드의 반영 여부를 확인하기 위해서 페이지 헤더 내에 존재하는 페이지 LSN과 로그 레코드의 LSN을 비교하는 부분을 코딩함으로써 제거할 수 있고, 이것은 무조건 한번의 재수행 또는 철회를 수행함으로써 발생하는 불필요한 회복 동작을 제거할 수 있다.

Ⅲ. 제안한 회복 기법

이 장에서는 제안한 회복 기법에서 트랜잭션 처리 및 로깅 과정, 검사점 수행, 로그 절단, 그리고 시스템 파손 시 회복 알고리즘에 대해서 기술한다.

1. 트랜잭션 처리 및 로깅 과정

트랜잭션이 한 페이지를 갱신하면, 표 3과 같은 구조의 재수행 전용 로그 레코드를 로깅한다.

표 3. 재수행 전용 로그 레코드 구조

LSN	trID	type	pgID	afterImg
로그 레코드 식별자	트랜잭션 식별자	로그 레코드 종류	페이지 식별자	갱신 연산의 사후 값

표 3에서 트랜잭션 식별자 trID에는 상위 1비트가 예약되어 있어 트랜잭션의 갱신 연산들 중, 첫 번째 갱신 연산의 로그 레코드인 경우 비트 연산자로 트랜잭션 식별자의 상위 1비트를 '0'에서 '1'로 변경한다(이것이 완료한 트랜잭션 리스트의 동적 구성을 지원한다). 로그 레코드의 종류에는 갱신('update'), 완료('commit'), 검사점('ckp') 그리고 2.1절에서 언급한 두 번째 문제를 해결하기 위한 별도의 after-image('afterImg') 로그 레코드가 있다.

트랜잭션 수행 시 표 4의 버퍼&록 관리 테이블(B&L_Tbl)을 이용하여 2.1절에서 언급한 첫 번째, 두 번째 문제점들을 해결한다.

표 4. B&L_Tbl

pgID	dirty	bufIdx	diskDisk
페이지 식별자	갱신 상태	버 퍼 인덱스	디스크 주소
lock	trID	LSN	prevLSN
록 정보	트랜잭션 식별자	로그 레코드 식별자	afterImg의 LSN

B&L_Tbl은 버퍼에 적재된 페이지와 수행 중인 트랜잭션의 록 정보를 가지고 있기 때문에, 페이지 교체로 해당 페이지가 버퍼에 존재하지 않으면서 해당 페이지를 갱신한 트랜잭션이 완료 또는 철회된 경우에 해당 페이지의 엔트리를 B&L_Tbl에서 삭제 할 수 있다.

표 4에서 LSN은 최근 해당 페이지에 반영된 갱신 로그 레코드의 LSN이고, prevLSN은 해당 페이지를 최근에 갱신하고 완료한 트랜잭션에 대한 갱신 로그 레코드의 LSN을 가진다. B&L_Tbl에 엔트리가 추가 될 경우 페이지 헤더 내에 존재하는 페이지 LSN이 truncatedLSN(로그 절단 LSN 정보로 3.2에서 설명함)보다 작으면 prevLSN에 '0'을 기록하고, 크거나 같으면 prevLSN에 페이지 LSN을 기록한다.

트랜잭션이 페이지 Pi를 갱신할 경우 다음과 같이 로깅 및 B&L_Tbl[Pi].prevLSN의 수정을 처리한다(i는 페이지 식별자를 나타냄).

B&L_Tbl[Pi].prevLSN이 '0'이면 ①부터 처리하고, 그렇지 않으면 ②부터 처리한다.

① 로그 레코드의 type에 'afterImg'를 기록하고 로그 레코드의 afterImg에 현재 페이지 값을 기록하여 별도의 after-image 로그 레코드를 생성하고 로그에 기록한다. 그리고, B&L_Tbl[Pi]. prev LSN에 after-image 로그 레코드의 LSN을 기록한다(2.1에서 언급한 두 번째 문제를 해결한다).

② 페이지를 갱신하면서 갱신 로그 레코드를 생성하여 로그에 기록하고, B&L_Tbl[Pi].LSN에 갱신 로그 레코드의 LSN을 기록한다. 일반적으로 로그 절단은 매번 수행하는 것이 아니기 때문에 과정 ①은 자주 처리되지 않는다.

트랜잭션 완료 시 완료 로그 레코드를 생성하여 로그에 기록하고, 해당 트랜잭션에 의해서 'write' 록이 걸려있는 페이지 Pi의 B&L_Tbl[Pi].prevLSN

에 B&L_Tbl[Pi].LSN을 기록한 뒤 'write' 록을 해제한다(2.1에서 언급한 첫 번째 문제를 해결한다).

트랜잭션 철회 시 해당 트랜잭션에 의해서 'write' 록이 걸려 있는 페이지 Pi의 B&L_Tbl[Pi].prevLSN를 이용하여 해당 로그 레코드의 afterImg로 트랜잭션을 철회한 뒤, 'write' 록을 해제한다.

2. 검사점 및 로그 절단

B&L_Tbl의 엔트리를 순차적으로 탐색하는 동안 다음 처리 과정과 같이 퍼지 검사점을 수행하여 갱신된 페이지를 디스크에 반영하고, 시스템 회복 시 필요한 redoLSN(검사점 시작 시 redoLSN은 현재의 로그 레코드의 LSN 값으로 설정함) 정보를 구성한다.

① pg := NxtB&L_Tbl();
② redoLSN := Min(redoLSN, pg.prevLSN);
③ if (pg.dirty = '1') then {Flush(pg.index); pg.dirty := '0'; }

①은 B&L_Tbl에서 페이지 엔트리를 읽고, ②는 redoLSN과 페이지 엔트리의 prevLSN 중에서 최소 값을 redoLSN에 기록한다. 그리고 ③은 페이지 엔트리가 갱신되었다면, 디스크에 해당 페이지를 반영(B&L_Tbl의 LSN도 페이지 헤더에 기록됨)하고 페이지 엔트리의 dirty에 '0'을 기록한다. 이와 같은 퍼지 검사점 수행이 끝나면, redoLSN을 검사점 로그 레코드에 기록하여 로깅한다.

시스템 파손 시 redoLSN보다 작은 로그 레코드들은 분석할 필요가 없기 때문에 로그 절단 정책에 의해서 보존 로그(archival log)에 보관될 수 있다. 따라서, 로그 절단은 검사점 수행 과정에서 ②만 수행하여 truncatedLSN에 redoLSN 값을 기록하고, 보존 로그를 생성한다.

3. 시스템 파손 시 회복 동작

그림 1의 회복 알고리즘과 같이 한번의 역방향 로그 분석을 하면서 재수행해야 하는 트랜잭션을

```
1: truncatedLSN := 0;
2: firstLSN := 0;
3: logRec := LAST_LOG();
4: DO {
5:     SWITCH (logRec.type) {
6:       CASE 'commit' : ADD(commitLst, logRec.trID);
7:         BREAK;
8:       CASE 'afterImg' :
9:         IF (B&L_Tbl[logRec.pgID].LSN < logRec.LSN) {
10:          REDO(B&L_Tbl[logRec.pgID].index, logRec.afterImg);
11:          MODIFY(B&L_Tbl[logRec.pgID], *, '1', *, *, *, *, logRec.LSN, logRec.LSN);
           }   // IF (B&L_Tbl[logRec.pgID].LSN < logRec.LSN)
12:        BREAK;
13:      CASE 'update' :
14:        IF (EXIST(commitLst, logRec.trID) {
15:          IF (B&L_Tbl[logRec.pgID].LSN < logRec.LSN) {
16:            REDO(B&L_Tbl[logRec.pgID].index, logRec.afterImg);
17:            MODIFY(B&L_Tbl[logRec.pgID], *, '1', *, *, *, *, logRec.LSN, logRec.LSN);
             }   // IF (B&L_Tbl[logRec.pgID].LSN < logRec.LSN)
18:          IF (logRec.trID(1) = '1') { DEL(commitLst, logRec.trID); }
           }   // IF (EXIST(commitLst, logRec.trID)
19:        ELSE {
20:          IF (B&L_Tbl[logRec.pgID].LSN = logRec.LSN) {
21:            MODIFY(B&L_Tbl[logRec.pgID], *, '1', *, *, *, *, '0', *);
             }   // IF (B&L_Tbl[logRec.pgID].LSN = logRec.LSN)
           }
22:        BREAK;
23:      CASE 'ckp' : firstLSN := logRec.redoLSN;
24:        BREAK;
       }   // SWITCH
25:    logRec := PREV_LOG();
26: } WHILE (logRec IS NOT BOF AND logRec.LSN ≠ firstLSN)
```

그림 1. 역방향 로그 분석을 이용한 회복 알고리즘

식별하고 해당 로그 레코드의 after-image를 이용하여 바로 재수행함으로써 시스템 파손 직전의 일관된 데이터베이스 상태로 복구시킨다.

라인 4~26은 마지막 로그 레코드부터 첫 번째 로그 레코드 또는 최근 검사점 로그 레코드에 기록된 redoLSN 로그 레코드까지 역방향으로 로그를 분석하는 반복 구문이다.

로그 분석 시 완료 로그 레코드를 분석할 경우, 해당 트랜잭션의 식별자를 완료한 트랜잭션 리스트(commitLst)에 추가한다(라인 6).

별도의 after-image 로그 레코드를 분석할 경우, 로그 레코드가 해당 페이지에 반영되지 않았으면 로그 레코드의 after-image를 이용하여 재수행한 뒤, B&L_Tbl에서 해당 페이지의 dirty('1'), LSN (logRec.LSN), prevLSN (logRec.LSN)을 수정하여 더 이상 재수행하지 않도록 한다(라인 9~11).

갱신 로그 레코드를 분석할 경우, 해당 로그 레코드의 트랜잭션 식별자가 commitLst에 존재하면 ① (라인 14~18)을, 그렇지 않으면 ②(라인 19~21)를 처리한다.

① 갱신 로그 레코드가 해당 페이지에 반영되지 않은 경우에만 로그 레코드의 after-image를 이용하여 재수행하고, B&L_Tbl에서 해당 페이지의 dirty('1'), LSN(logRec.LSN), prevLSN(logRec.LSN)을 수정한다(라인 15~17). 그리고 로그 레코드의 trID에서 예약된 상위 1비트가 '1'이면 해당 트랜잭션의 첫 번째 갱신 연산을 뜻함으로 commitLst에서 해당 트랜잭션 식별자를 삭제한다(라인 18).

② 해당 페이지에 갱신 로그 레코드가 최근에 반영된 것이면, B&L_Tbl에서 해당 페이지의 dirty('1'), LSN('0')을 수정하여 이전에 완료한 트랜잭션의 갱신 로그 레코드에 의해서 재수행할 수 있도록 한다(라인 20~21).

검사점 로그 레코드를 분석할 경우, 마지막으로 분석할 로그 레코드의 LSN을 의미하는 redoLSN을 firstLSN에 기록한다.

제안한 회복 알고리즘은 역방향 로그 분석을 이용한 기존 회복 기법과는 달리 라인 18에 의해서 완료한 트랜잭션 리스트에서 불필요한 엔트리는 삭제되고, 페이지의 LSN과 로그 레코드의 LSN을 비교해야 재수행 여부를 판별하기 때문에(라인 9, 15, 20) 회복된 페이지 리스트가 필요 없으며, 갱신된 페이지를 무조건 재수행하지 않는다. 그리고 로그에는 불필요한 before-image가 존재하지 않기 때문에 로그 분석 시 소요 시간을 감소시킨다.

Ⅳ. 제안한 회복 기법과 기존 회복 기법의 비교 분석

이 장에서는 제안한 회복 기법, 역방향 로그 분석을 이용한 기존 회복 기법, 전방향 로그 분석을 이용한 기존 회복 기법의 비교 분석으로 두 가지 트랜잭션 수행 예제를 적용한 비교 분석과 알고리즘 분석을 통한 비교 분석결과를 제시한다.

1. 트랜잭션 수행 예제를 통한 비교 분석 A

역방향 로그 분석을 이용한 기존 회복 기법[8]에서 기술된 트랜잭션 수행 예제를 적용하기 전에 몇 가지 가정을 세운다.

- 트랜잭션이 페이지 Pi(n)를 갱신하면, Pi의 after-image는 Pi(n+1)이다(n은 양의 정수).
- 현재의 버퍼와 데이터베이스 상태는 그림 2와 같다.

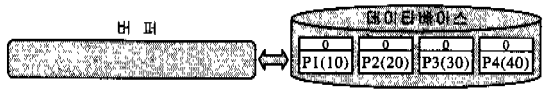


그림 2. 현재의 버퍼와 데이터베이스 상태

- 첫 번째 로그 레코드의 LSN은 100이다.
- 적용할 트랜잭션 수행 예제는 표 5와 같다.

표 5. 트랜잭션 수행 예제

TIME	EVENT
1	Update(T1, P1)
2	Update(T2, P3)
3	Update(T1, P2)
4	Update(T3, P4)
5	Commit(T1)
6	Update(T2, P1)
7	Update(T3, P2)
8	Commit(T3)
9	시스템 파손

- TIME 9의 시스템 파손 직전에 디스크에 반영된 페이지 상태는 P1(10), P2(20), P3(30), P4(40) 이다.
- TIME 9의 시스템 파손 직전에 버퍼 상의 페이지 상태는 P1(12), P2(22), P3(31), P4(41) 이다.

· 회복할 페이지의 상태는 P1(11), P2(22), P3(30), P4(41)이다.
· TIME 9의 시스템 파손 직전에 기존 회복 기법과 제안한 회복 기법의 로그 상태는 표 6, 7과 같다.

표 6. 전방향 로그 분석과 역방향 로그 분석을 이용한 기존 회복 기법의 로그 상태

LSN	trID	type	pgID	afterImg	beforeImg
100	T1	'update'	P1	P1(11)	P1(10)
101	T2	'update'	P3	P3(31)	P3(30)
102	T1	'update'	P2	P2(21)	P2(20)
103	T3	'update'	P4	P4(41)	P4(40)
104	T1	'commit'			
105	T2	'update'	P1	P1(12)	P1(11)
106	T3	'update'	P2	P2(22)	P2(21)
107	T3	'commit'			

표 7. 제안한 회복 기법의 로그 상태

LSN	trID	type	pgID	afterImg
100		'afterImg'	P1	P1(10)
101	T1	'update'	P1	P1(11)
102		'afterImg'	P3	P3(30)
103	T2	'update'	P3	P3(31)
104		'afterImg'	P2	P2(20)
105	T1	'update'	P2	P2(21)
106		'afterImg'	P4	P4(40)
107	T3	'update'	P4	P4(41)
108	T1	'commit'		
109	T2	'update'	P1	P1(12)
110	T3	'update'	P2	P2(22)
111	T3	'commit'		

표 8에서는 시스템 회복 시 제안한 회복 기법과 기존 회복기법의 차이점을 기술했다.

표 8에서 알 수 있듯이, 전방향 로그 분석을 이용한 기존 회복 기법의 재수행/철회 동작이 많음을 알 수 있다. 전방향 로그 분석과 역방향 로그 분석을 이용한 기존 회복 기법에는 중복된 before-image가 존재한다. 제안한 회복 기법은 별도의 after-image(LSN 100, 102, 104, 106) 로그 레코드를 로깅으로 4개의 로그 레코드가 더 많지만, 로그의 전체 크기를 비교해 보면 제안한 회복 기법의 로그가 작음을 알 수 있다. 그리고 계속해서 페이지 P1, P2, P3, P4에 대한 더 많은 트랜잭션의 갱신 연산이 존재하면 할수록 로그 크기의 차이는 더 커진다. 역방향 로그 분석을 이용한 기존 회복 기법에서는 시스템 회복 시 점차 증가하는 COMMIT_LIST, UPDATED_LIST의 정보를 계속 유지하는데 반해서 제안한 회복 기법에서는 동적으로 엔트리가 추가 삭제되는 commitLst 정보만을 유지한다.

2. 트랜잭션 수행 예제를 통한 비교 분석 B

트랜잭션 수행 예제를 적용하기 전에 몇 가지 가정을 세운다.

· 트랜잭션이 페이지 Pi(n)를 갱신하면, Pi의 after-image는 Pi(n+1)이다(n은 양의 정수).
· 현재의 버퍼와 데이터베이스의 상태는 그림 3과 같다.
· 첫 번째 로그 레코드의 LSN은 100이다.
· 적용할 트랜잭션 수행 예제는 표 9와 같다.

표 8. 회복 동작의 차이점

	전방향 로그 분석을 이용한 기존 회복 기법	역방향 로그 분석을 이용한 기존 회복 기법	제안한 회복 기법
로그 분석 동작	LSN(100) ~ LSN(107)	LSN(107) ~ LSN(100)	LSN(111) ~ LSN(100)
시스템 회복 동작	1. LSN(100) 재수행 2. LSN(101) 재수행 3. LSN(102) 재수행 4. LSN(103) 재수행 5. LSN(105) 재수행 6. LSN(106) 재수행 7. LSN(105) 철회 8. LSN(101) 철회	1. LSN(106) 재수행 2. LSN(105) 철회 3. LSN(103) 재수행 4. LSN(101) 철회	1. LSN(110) 재수행 2. LSN(107) 재수행 3. LSN(102) 재수행 4. LSN(101) 재수행
기타 차이점	· 중복된 before-image 존재	· 중복된 before-image 존재 · COMMIT_LIST{T1, T3}을 계속 유지 · UPDATED_LIST{P1, P2, P3, P4} 계속 유지	· B&L_Tbl에서 prevLSN 유지 및 관리, 별도의 after- image 로그 레코드(LSN 100, 102, 104, 106)를 로깅함 · commitLst{필요 없는 엔트리는 삭제됨}

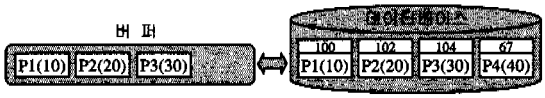


그림 3. 현재의 버퍼와 데이터베이스 상태

표 9. 트랜잭션 수행 예제

TIME	EVENT
1	Update(T11, P1)
2	Update(T12, P4)
3	Update(T11, P3)
4	Update(T11, P2)
5	Commit(T11)
6	Abort(T12)
7	Update(T13, P4)
8	Commit(T13)
9	퍼지 검사점 시작
10	Flush(P1, P2) by 검사점
11	Update(T14, P3)
12	Update(T15, P4)
13	Update(T15, P1)
14	Flush(P3, P4) by 검사점
15	퍼지 검사점 끝
16	Abort(T14)
17	Commit(T15)
18	Update(T16, P1)
19	Update(T16, P4)
20	Commit(T16)
21	Update(T17, P1)
22	시스템 파손

- TIME 22의 시스템 파손 직전에 디스크에 반영된 페이지 상태는 P1(11), P2(21), P3(32), P4(42) 이다.
- TIME 22의 시스템 파손 직전에 버퍼 상의 페이지 상태는 P1(14), P2(21), P3(31), P4(43) 이다.
- 회복할 페이지의 상태는 P1(13), P2(21), P3(31), P4(43) 이다.
- TIME 22의 시스템 파손 직전에 기존 회복 기법과 제안한 회복 기법의 로그 상태는 표 10, 11과 같다.

표 12에서는 시스템 회복 시 제안한 회복 기법과 기존 회복기법의 차이점을 기술했다.

표 12에서도 알 수 있듯이, 전방향 로그 분석을 이용한 기존 회복 기법의 재수행/철회 동작이 많음을 알 수 있다. 전방향 로그 분석과 역방향 로그 분석을 이용한 기존 회복 기법에는 중복된 before-

표 10. 전방향 로그 분석과 역방향 로그 분석을 이용한 기존 회복 기법의 로그 상태

LSN	TID	type	PID	beforeImg	afterImg
100	T10	'update'	P1	P1(10)	P1(9)
101	T10	'update'	P2	P2(20)	P2(19)
102	T10	'update'	P3	P3(30)	P3(29)
103	T10	'commit'			
104	T11	'update'	P1	P1(11)	P1(10)
105	T12	'update'	P4	P4(41)	P4(40)
106	T11	'update'	P3	P3(31)	P3(30)
107	T11	'update'	P2	P2(21)	P2(20)
108	T11	'commit'			
109	T13	'update'	P4	P4(41)	P4(40)
110	T13	'commit'			
111	T14	'update'	P3	P3(32)	P3(31)
112	T15	'update'	P4	P4(42)	P4(41)
113	T15	'update'	P1	P1(12)	P1(11)
114		'ckp'	redoLSN := 104		
115	T15	'commit'			
116	T16	'update'	P1	P1(13)	P1(12)
117	T16	'update'	P4	P4(43)	P4(42)
118	T16	'commit'			
119	T17	'update'	P1	P1(14)	P1(13)

표 11. 제안한 회복 기법의 재수행 전용 로그 상태

LSN	TID	type	PID	afterImg
100	(1)T10	'update'	P1	P1(10)
101	T10	'update'	P2	P2(20)
102	T10	'update'	P3	P3(30)
103	T10	'commit'		
104	(1)T11	'update'	P1	P1(11)
105		'afterImg'	P4	P4(40)
106	(1)T12	'update'	P4	P4(41)
107	T11	'update'	P3	P3(31)
108	T11	'update'	P2	P2(21)
109	T11	'commit'		
110	(1)T13	'update'	P4	P4(41)
111	T13	'commit'		
112	(1)T14	'update'	P3	P3(32)
113	(1)T15	'update'	P4	P4(42)
114	T15	'update'	P1	P1(12)
115		'ckp'	redoLSN := 104	
116	T15	'commit'		
117	(1)T16	'update'	P1	P1(13)
118	T16	'update'	P4	P4(43)
119	T16	'commit'		
120	T17	'update'	P1	P1(14)

표 12. 회복 동작의 차이점

	전방향 로그 분석을 이용한 기존 회복 기법	역방향 로그 분석을 이용한 기존 회복 기법	제안한 회복 기법
로그 분석 범위	LSN(104) ~ LSN(119)	LSN(119) ~ LSN(104)	LSN(120) ~ LSN(104)
재수행 회복 동작	1. LSN(113) 재수행 2. LSN(116) 재수행 3. LSN(117) 재수행 4. LSN(119) 재수행 5. LSN(119) 철회 6. LSN(111) 철회	1. LSN(119) 철회 2. LSN(117) 재수행 3. LSN(111) 철회 4. LSN(107) 재수행	1. LSN(118) 재수행 2. LSN(117) 재수행 3. LSN(107) 재수행
기타 자료들	· 중복된 before-image 존재	· 중복된 before-image 존재 · COMMIT_LIST{T11, T13, T15, T16}을 계속 유지 · UPDATED_LIST{P1, P2, P3, P4} 계속 유지	· B&L_Tbl에서 prevLSN 유지 및 관리, 별도의 after- image 로그 레코드(LSN 105)를 로깅 함 · commitLst{필요 없는 엔트리는 삭제됨}

image가 존재한다. 제안한 회복 기법은 별도의 after-image(LSN 105) 로그 레코드를 로깅 했기 때문에 1개의 로그 레코드가 더 있지만, 로그 전체 크기를 비교해 보면 제안한 회복 기법의 로그가 작음을 알 수 있다. 역방향 로그 분석을 이용한 기존 회복 기법에서는 시스템 회복 시 점차 증가하는 COMMIT_LIST, UPDATED_LIST의 정보를 계속 유지하는데 반해서 제안한 회복 기법에서는 동적으로 엔트리가 추가 삭제되는 commitLst 정보만을 유지정보만을 유지한다. 결과적으로 트랜잭션의 수행 예제를 통한 비교분석 A,B에서 ESM/CS모델보다 로그의 분석범위가 30%감소되고 재수행 동작 횟수도 40%정도 감소 하였으나 Client-based Logging모델에 비해서 서버의 재수행 동작 횟수가 클라이언트에서는 감소한 반면 약간 증가됨을 확인할 수 있다. 이는 클라이언트에서 트랜잭션이 많이 발생할수록 그 차이는 더욱 커진다.

V. 결 론

제안한 회복 기법은 STEAL과 NO FORCE를 지원하는 시스템에서 별도의 버퍼를 예비하지 않고, 불필요한 before-image를 제거하여 재수행 전용 로그 레코드만을 로깅 함으로써 로깅 오버헤드를 감소시켰으며, 로그 분석 시 소요 시간의 감소를 기대할 수 있다. 시스템 회복 시 재수행 전용 로그를 역방향으로 분석하면서 손실된 페이지를 바로 재수행하여 일관된 데이터베이스 상태로 복구하는 회복 기법을 제안했다. 그리고 역방향 로그 분석을 이용한 기존 회복 기법에서, 회복 시에 유지 및 관리해야 하는 자료구조의 오버헤드를 최소화하기 위해 동적으로 추가 삭제되는 commitLst를 사용하여 해결했다.

추후 연구 과제로는 제안한 로깅 및 회복 알고리즘에 대한 정확한 성능 평가와 클라이언트/서버 데이터베이스 시스템에 제안한 회복 기법을 적용하는 연구가 필요하다.

참 고 문 헌

[1] P. A. Bernstein, et al., "Concurrency Control and Recovery in Database Systems," Addison-Wesley, 1987.

[2] A. Delis and N. Roussopouls, "Modern Client-Server DBMS Architectures," Proc. ACM SIGMOD RECORD, pp.52-61, 1991.

[3] A. Delis, N. Roussopouls, "Performance and Scalability of Client-Server Database Architectures," Proc. VLDB, pp.610-623, 1992.

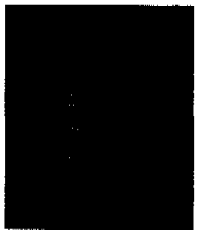
[4] D. Dewiff et al., "A Study of three Alternative Workstation-Server Architectures for Object Oriented Database Systems," Proc. VLDB, pp.107-121, 1990.

[5] M. Franklin, M. Carey, "Crash Recovery in Client-Server EXODUS," Proc. ACM SIGMOD, pp.165-174, 1992.

[6] Jim Gray, Andreas Reuter, "Transaction Processing: Concepts and Techniques," Morgan

Kaufmann, 1993.
[7] Theo Haerder, Andreas Reuter, "Principles of Transaction-Oriented Database Recovery," Computing Surveys, pp.287-317, 1983.
[8] Tobin J. Lehman, Michael J. Carey, "A Recovery Algorithm for A High-Performance Memory-Resident Database System", ACM, pp.104-117, 1987.
[9] David Lomet, Mark Tuttle, "Redo Recovery after System Crashes," Proc. VLDB, pp.457-468, 1995.
[10] David Lomet, "Persistent Applications Using Generalized Redo Recovery," IEEE ICDE, pp.154-163, 1998.
[11] David Lomet, Gerhard Weikum, "Efficient Transparent Application Recovery in Client-Server Information Systes," Proc. ACM SIGMOD, pp.460-471, 1998.
[12] C. Mohan & Don Haderle, et al, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," Proc. ACM TODS, pp.94-162, 1992.
[13] C. Mohan & I Narang, "ARIES/CSA: A Method for Database Recovery in Client-Server Architectures," Proc. ACM SIGMOD, pp.55-66, 1994.
[14] E. Panagos, A. Biliris, "Synchronization and recovery in a client-server storage system," The VLDB Journal, pp.209-223, 1997.
[15] E. Panagos, A. Biliris, H. V. Jagadish, R. Restogi, "Client-Based Logging for High Performance Distributed Architectures," pp.344-351, 1996.
[16] 유정준, 김유성, 배해영, "최소화된 로그 입출력 비용을 갖는 빠른 회복 기법," 한국정보과학회 추계학술발표논문집(A), 제 22권, 제 2호, pp.31-34, 1995.

이 찬 섭(Chan-Seob Lee) 정회원



1990년 : 한남대학교 컴퓨터공학과 졸업(학사)
2000년 : 한남대학교 대학원 컴퓨터공학과(공학석사)
2000년 : 한남대학교 대학원 컴퓨터공학과(박사과정)

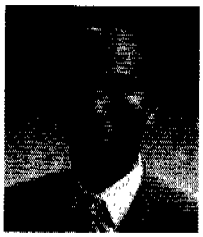
<주관심 분야> Data mining, Web DB, XML

박 용 문(Yong-Mun Park) 정회원

1983년 : 숭전대학교 계산통계학과 졸업(학사)
1985년 : 중앙대학교 대학원 전자계산학과(이학석사)
2000년 : 한남대학교 대학원 컴퓨터공학과(박사과정)

1985년~2000년 : 한국전자통신연구원 선임연구원
<주관심 분야> 이동컴퓨팅, 실시간 데이터베이스

고 병 오(Byung-Oh Koh) 정회원



1986년 : 충남대학교 계산통계학과 졸업(이학사)
1989년 : 홍익대학교 전자계산학과 졸업(이학석사)
1996년 : 홍익대학교 전자계산학과 졸업(이학박사)

1994년~1996년 : 세명대학교 정보처리학과 조교수
1997년~현재 : 공주교육대학교 실과 조교수
<주관심 분야> 실시간데이타베이스, 주기억 데이타베이스, 웹 데이터베이스, 컴퓨터 교육

최 의 인(Eui-In choi) 정회원

1982년 : 숭전대학교 계산통계학과 졸업(학사)
1984년 : 홍익대학교 전자계산학과 졸업(이학석사)
1995년 : 홍익대학교 전자계산학과 졸업(이학박사)
1985년~1988년 : 공군 교육사 전산실장
1992년~1996년 : 명지전문대학 전자계산과 조교수
1996년~현재 : 한남대학교 컴퓨터공학과 부교수
<주관심 분야> 실시간데이타베이스, 주기억 데이타베이스, 클라이언트/서버 데이타베이스