클라이언트/서버 환경에서 클라이언트 기반 로깅을 이용한 회복 기법

박 용 문[†] · 이 찬 섭[††] · 김 희 수[†††] · 최 의 인[††††]

요 약

클라이언트/서버 데이터베이스 시스템에서 로깅 기법을 사용하는 기존 회복 기법은 서버에서만 전체 로그를 관리한다. 이는 잠재적으로 각 클라이언트에서 수행되는 트랜잭션에 대한 로그 레코드의 전송 비용을 내포하고 있고, 네트워크 트래픽을 증가시킨다. 본 논문에서는 로그 레코드의 전송 비용을 제거하기 위해서 클라이언트 기반 로깅(client-based logging)을 지원하고, 중복된 before-image를 제거하고 재수행 전용 로그(redo-only log)만을 로깅하는 방법을 제안한다. 그리고 클라이언트 파손 시 클라이언트에서 자치적으로 한번의 후방향 클라이언트 로그 분석을 이용한 재수행 회복을 하고, 서버 파손 시 각 클라이언트에서는 병행적으로 후방향 클라이언트 로그 분석을 이용하여 회복해야 하는 페이지의 after-image만을 서버에 전송하며, 서버에서는 수신된 after-image와 후방향 서버 로그 분석을 이용하여 재수행 회복을 수행한다.

A Recovery Technique Using Client-based Logging in Client/Server Environment

Yong-Mun Park[†] · Chan-Seob Lee[††] · Hee-Soo Kim[†††] · Eui-In Choi[††††]

ABSTRACT

The existing recovery methods that use the logging technology in client/server systems manage all log only at the server. Potentially, it involves a transmission cost for log recording performed for certain transaction at each client and it increases network traffic. In this paper, we propose a redo-only log record method which can remove an overlap before-image and can support client base on logging method so as to get rid of the transmission cost required for log recording. This method also perform redo recovery process by using a backward client log analysis by itself when the client system is destroyed. When the server system is destroyed, each client send only after-image to server by using a backward client log analysis. After then, the server perform redo recovery process by using both the received after-image and backward log analysis.

1. 서 론

고성능 워크스테이션의 등장과 통신 장비의 발전으로 클라이언트/서버 데이터베이스 시스템에 대한 연구가 활발히 진행되고 있다. 이런 시스템의 확장은 높은 트랜잭션 처리율을 요구하고, 시스템 파손이 발생하면 그 만큼 디스크에 기록되지 못한 많은 데이터베이스의 내용이 손실됨으로 파손 직전의 일관된 데이터베이스 상태로 복구할 수 있는 회복 기법이 필요하다[1-5].

클라이언트/서버 데이터베이스 시스템에서 기존 회복 기법은 각 클라이언트에서 트랜잭션 수행 시 생성되는 모든 로그 레코드를 서버로 전송한다. 즉, 서버에서 전체 로그를 관리한다. 이는 로그 레코드의 전송으

※ 이 논문은 1999년도 한남대학교 학술 연구 조성비 지원에 의하여 지원되었음.
† 정 회 원 : 한국전자통신연구원 선임연구원
†† 준 회 원 : 한남대학교 대학원 컴퓨터공학과
††† 정 회 원 : it4web(주) 연구원
†††† 종신회원 : 한남대학교 컴퓨터공학과 교수
논문접수 : 2000년 6월 27일, 심사완료 : 2000년 7월 25일

로 네트워크 트래픽(network traffic)을 증가시킨다. 그리고, 트랜잭션의 완료(commit) 시점은 해당 트랜잭션에 대한 모든 로그 레코드를 기록한 뒤에 가능하기 때문에 로그 레코드의 전송은 트랜잭션 수행을 지연시킬 수도 있다. 시스템 파손 시 서버 로그를 이용해야 하기 때문에 시스템 파손 시 회복 동작에 대한 서버의 작업부하(workload)가 크고, 클라이언트 파손 시에는 불필요한 로그 분석이 많아진다[6, 7].

제안한 회복 기법에서 각 클라이언트는 트랜잭션 수행 시 생성된 로그 레코드를 자신의 로그 파일에 기록하는 클라이언트 기반 로깅을 지원하여 로그 레코드의 전송 비용을 제거하였다. 그리고 클라이언트 파손 시 자치적으로 한번의 후방향 클라이언트 로그 분석을 이용하여 손실한 페이지의 after-image를 식별하고 바로 재수행 하도록 했으며, 서버 파손 시 각 클라이언트에서는 병행적으로 후방향 클라이언트 로그 분석을 이용하여 회복해야 하는 페이지의 after-image만을 서버로 전송하고 서버에서는 수신된 after-image와 후방향 서버 로그 분석을 이용하여 손실된 페이지를 재수행 하도록 했다.

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 트랜잭션의 수행 예제를 통하여 제안한 회복 기법과 기존 회복 기법을 비교 분석한다. 제 5장에서는 결론 및 추후 연구 방향에 대하여 기술한다.

2. 기존 회복 기법들의 문제점

이 장에서는 클라이언트/서버 데이터베이스 시스템에서 기존 회복 기법들의 문제점 세 가지로서, 로그 레코드 전송 비용, 회복에 대한 서버의 작업부하, 그리고 논리적(logical) 로깅과 클라이언트 기반 로깅 지원 시 문제점을 제시한다.

2.1 로그 레코드 전송 비용

트랜잭션 T1이 차례로 페이지 P1(0), P2(0), P3(0)을 갱신하고 완료하기 위해서는 <표 1>의 로그 레코드들을 서버 로그에 기록해야 한다.

〈표 1〉 서버 로그에 기록해야 하는 로그 레코드

LSN	trID	type	pgID	afterImg	beforeImg
100	T1	'update'	P1	P1(1)	P1(0)
101	T1	'update'	P2	P2(1)	P2(0)
102	T1	'update'	P3	P3(1)	P3(0)
103	T1	'commit'			

트랜잭션 T1이 완료하기까지의 소요 시간은 'T1의 연산 소요 시간 + 해당 로그 레코드들(LSN(100~103))의 로깅 소요 시간'이다. 로그 레코드의 로깅 소요 시간은 '로그 레코드의 생성 및 기록 시간 + 로그 레코드의 전송 시간'이다. 즉, 클라이언트/서버 데이터베이스 시스템에서 트랜잭션 수행 시 생성된 로그 레코드를 서버로 전송하는 것은 로깅 비용을 증가시키는 것은 물론이고 트랜잭션 수행을 지연시킨다. 그리고 전체 로그가 서버에 존재하기 때문에 2.2절에 설명된 것처럼 시스템 파손 시 회복에 대한 서버의 작업부하가 크다는 단점이 있다.

2.2 회복에 대한 서버의 작업부하

서버에서 전체 로그를 유지 및 관리하고 있는 기존 시스템에서는 클라이언트 또는 서버 파손 시 서버 로그를 이용하여 회복한다. 즉, 시스템 파손에 대한 모든 회복 동작이 서버에서 수행하기 때문에 회복에 대한 서버의 작업부하가 크다. 또한 서버 로그에는 모든 클라이언트에서 수행한 트랜잭션의 로그 레코드가 기록되어있기 때문에 클라이언트 파손 시, 파손된 클라이언트와 상관없는 로그 레코드를 분석함으로써 로그 분석 시간을 증가시킬 수 있다[8, 9].

제안한 회복 기법에서는 클라이언트 기반 로깅을 지원하기 때문에 2.1절에서 문제를 제기한 로그 레코드의 전송 비용을 제거했다. 클라이언트 기반 로깅은 각 클라이언트에 자신의 로그를 가지고 있어서 트랜잭션 수행 시 생성된 로그 레코드를 자신의 로그에 기록하는 것을 말한다[8]. 그리고 클라이언트 파손 시 자신의 로그만을 이용하여 자치적으로 회복 할 수 있다. 서버 파손 시 각 클라이언트에서 병행적으로 후방향 클라이언트 로그를 분석하여 회복에 필요한 after-image만을 전송하고, 서버에서는 수신된 after-image와 후방향 서버 로그 분석 결과에 따라 손실된 페이지를 재수행하여 일관된 데이터베이스 상태로 복구한다. 따라서 기존 클라이언트/서버 데이터베이스 시스템에서 회복에 대한 서버의 작업부하를 감소시킴으로써 서버의 가용

성을 높일 수 있다.

2.3 논리적 로깅과 클라이언트 기반 로깅 지원 시 문제점

클라이언트 기반 로깅을 이용하면 위의 두 문제를 해결할 수 있다. 하지만, 시스템 파손 시 모든 로그가 각 시스템에 분산되어 있기 때문에 시스템 회복을 위해서 분산된 로그를 합병하거나 각 시스템과 통신을 해야하는 문제점이 있다[10, 11]. 제안한 회복 기법에서는 각 시스템의 로그를 합병하지 않고 회복하는 기법을 제안했다.

논리적 로깅을 사용하는 시스템에서는 트랜잭션 연산에 대한 함수 이름이나 함수 전달 인자 등을 로그 레코드에 기록하기 때문에 반드시 트랜잭션 연산이 적용된 순서대로 해당 로그 레코드를 이용하여 페이지에 적용해야 한다. 따라서 서버 파손 시 회복해야 하는 페이지들에 대해서 적용할 로그 레코드의 순서를 결정해야 하기 때문에 많은 통신이 필요하다[12, 13].

반면에, 물리적(physical) 로깅을 사용하는 시스템에서는 트랜잭션 연산을 해당 페이지에 적용하기 이전의 before-image와 적용 이후의 after-image를 로그 레코드에 기록하기 때문에 시스템 파손 이전에 마지막으로 수행한 갱신 연산의 before-image나 after-image만을 이용하여 손실된 페이지를 회복할 수 있다. 서버 파손 시 회복해야 하는 페이지의 after-image만을 서버로 전송하고, 서버는 수신된 after-image를 가지고 손실된 페이지를 회복한다. 클라이언트 기반 로깅을 지원하는 데이터베이스 시스템에서 논리적 로깅과 물리적 로깅의 사용에 따른 구체적인 차이점은 4장에서 설명한다.

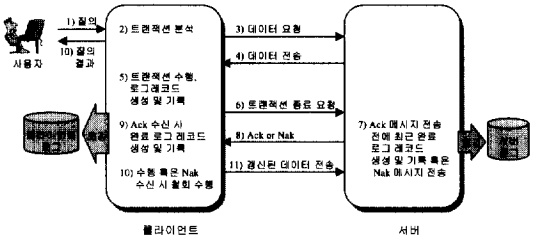
3. 제안한 회복 기법

이 장에서는 제안한 회복 기법에서 트랜잭션 처리 및 로깅 과정, 그리고 클라이언트 파손 시 회복 동작과 서버 시스템 파손 시 회복 동작에 대해서 기술한다.

3.1 트랜잭션 처리 및 로깅 과정

클라이언트에서 트랜잭션 처리 및 로깅 과정은 (그림 1)과 같다.

(그림 1)에서와 같이 클라이언트의 부회복관리기가 <표 2>와 같은 구조의 재수행 전용 로그 레코드를 클라이언트 로그에 기록한다.



(그림 1) 트랜잭션 처리 및 로깅 과정

〈표 2〉 클라이언트 로그

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

<표 2>에서 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(완료한 트랜잭션 리스트의 동적 구성을 지원한다). 로그 레코드의 종류에는 트랜잭션 연산에 의해 이전 페이지를 새로운 페이지로 교체를 나타내는 갱신('update'), 트랜잭션 연산의 종료를 나타내는 완료('commit'), 불필요한 로그 레코드를 제거하기 위한 검사점('ckp'), 별도의 after-image('afterImg') 로그 레코드가 있다. 이 after-image 로그 레코드는 트랜잭션 철회 시 이용될 after-image가 없는 경우를 대비해서 해당 페이지의 현재 값을 별도로 로그에 기록한 것이다.

(그림 1)과 같이 클라이언트에서 트랜잭션 완료 요청 메시지를 서버로 전송하면, 서버에서는 해당 트랜잭션의 완료를 허가하기 전에 <표 3>의 서버 로그에 최근 완료(recent commit) 로그 레코드를 기록한다. 그리고 트랜잭션 완료 메시지(Ack)에 해당 최근 완료 로그 레코드의 LSN(서버 B&L_Tbl에서 prevLSN)을 첨부하여 전송한다(최근 완료 로그 레코드는 트랜잭션들의 완료한 순서와 해당 트랜잭션에 의해서 갱신한 페이지의 식별자 정보를 갖는다). 만일, 트랜잭션 철회 메시지(Nak)를 전송하는 경우에는 최근 완료 로그 레코드를 기록하지 않는다. 그리고, 클라이언트에서 트랜잭션 완료 메시지를 수신하면 트랜잭션 완료 메시지에 첨부되어온 최근 완료 로그 레코드의 LSN을 해당 트랜잭션에 의해 write 록이 걸려있는 페이지의 LSN에 기록한 뒤, 해당 트랜잭션의 완료 로그 레코드를 기록하여 해당 트랜잭션을 완료한다.

〈표 3〉 서버 로그

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

<표 3>의 서버 로그는 클라이언트 로그와 같은 구조를 가지고 있으나, 최근 완료 로그 레코드가 추가되었다. 이 로그 레코드는 서버가 클라이언트에서 수행중인 트랜잭션(nodeID.trID)의 완료를 허락하는 시점에서 트랜잭션 완료 메시지를 전송하기 전에 로그 레코드의 type에 최근 완료('recCommit')를 기록하고, afterImg에 해당 트랜잭션에 의해서 갱신된 페이지의 식별자들(pgIDs)을 분리 기호('#')로 구분하여 모두 기록하고 최근 완료 로그 레코드를 생성한 다음 로그에 기록한다. 이와 같이, 서버의 회복 관리기가 최근 완료 로그 레코드를 생성하여 기록함으로써 서버 파손 시 각 클라이언트에 회복해야 할 페이지의 after-image만을 요청할 수 있다. 최근 완료 로그 레코드의 afterImg (pgIDs)는 <표 4>와 같은 서버의 버퍼&록 관리 테이블(B&L_Tbl)을 이용한다(클라이언트의 버퍼&록 관리 테이블 구조는 <표 4>에서의 nodeID.trID가 trID라는 점만 다르다).

〈표 4〉 서버의 B&L_Tbl

pgID	dirty	index	addrDisk	lock	nodeID.trID	LSN	prevLSN
페이지 식별자	갱신 상태	버퍼 인덱스	디스크 주소	록 정보	노드 식별자. 트랜잭션 식별자	로그 레코드 식별자	afterImg의 LSN

B&L_Tbl은 버퍼에 적재된 페이지의 상태와 록 정보를 가지고 있기 때문에, 갱신된 내용이 디스크에 반영된 페이지가 버퍼에 존재하지 않으면서 해당 페이지를 갱신한 트랜잭션이 완료 또는 철회된 경우에 해당 페이지의 엔트리를 B&L_Tbl에서 삭제 할 수 있다.

B&L_Tbl에서 LSN은 최근 해당 페이지에 반영된 갱신 로그 레코드의 LSN이고, prevLSN은 해당 페이지를 최근에 갱신하고 완료한 트랜잭션에 대한 갱신 로그 레코드의 LSN을 가진다. 서버의 B&L_Tbl의 LSN과 prevLSN은 서버 로그 레코드의 LSN으로 페이지 헤더 내에 있는 페이지 LSN에 적용된다. 클라이언트의 B&L_Tbl의 LSN과 prevLSN은 각 클라이언트 로그 레코드의 LSN으로 클라이언트의 B&L_Tbl에서 유지되는 정보이다. 이 정보는 페이지 헤더 내에 있는 페이지 LSN에는 적용되지 않는다. 그리고 서버의 B&L_Tbl의 LSN과 prevLSN은 갱신 로그 레코드의 LSN뿐만 아니라 최근 완료 로그 레코드의 LSN을 나타내는데, 이를 구별하기 위해서 페이지 LSN과 B&L_Tbl의 LSN, prevLSN에 상위 1비트를 예약하여 상위 1비트를 '0'에서 '1'로 변경한다.

서버에서 최근 완료 로그 레코드 생성 시, 서버의 B&L_Tbl의 페이지 중 해당 트랜잭션에 의해서 write 록이 걸려있는 페이지의 식별자를 최근 완료 로그 레코드의 afterImg(pgIDs)에 기록하고, 최근 완료 로그 레코드의 LSN을 비트 연산자를 이용하여 상위 1비트를 '1'로 변경하여 <표 4>의 prevLSN에 기록한다.

B&L_Tbl에 엔트리가 추가 될 경우 서버에서는 페이지 헤더에 존재하는 페이지 LSN이 서버의 truncatedLSN보다 작은 경우에 prevLSN에 '0'을 기록하고, 그렇지 않은 경우에는 prevLSN에 페이지 LSN을 기록한다. 클라이언트에서는 해당 페이지가 없어서 페이지를 캐쉬함으로써 클라이언트의 B&L_Tbl에 엔트리가 추가 될 경우에 무조건 B&L_Tbl의 prevLSN에 '0'을 기록한다.

각 시스템에서 트랜잭션이 페이지 Pi를 갱신할 경우 다음과 같이 로깅 및 B&L_Tbl[Pi].prevLSN을 수정한다(i는 페이지 식별자를 나타냄). B&L_Tbl[Pi].prevLSN이 '0'이면 ①부터 처리하고, 그렇지 않으면 ②부터 처리한다. 서버에서는 갱신할 페이지 LSN의 상위 1비트가 '1'인 경우에도 ①을 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]. prevLSN에 after-image 로그 레코드의 LSN을 기록한다.

② 페이지를 갱신하면서 갱신 로그 레코드를 생성하고 로그에 기록하고, B&L_Tbl[Pi].LSN에 갱신 로그 레코드의 LSN을 기록한다. 일반적으로 prevLSN이 '0'인 경우는 거의 발생하지 않기 때문에 과정 ①은 무시할 수 있다.

각 시스템에서 트랜잭션 완료 시 완료 로그 레코드를 생성하여 로그에 기록하고, 해당 트랜잭션에 의해서 'write' 록이 걸려있는 페이지 Pi의 B&L_Tbl[Pi]. prevLSN에 B&L_Tbl[Pi].LSN을 기록한 뒤 'write' 록을 해제한다. 그리고, 트랜잭션 철회 시 해당 트랜잭션에 의해서 'write' 록이 걸려 있는 페이지 Pi의 B&L_Tbl[Pi].prevLSN을 이용하여 해당 로그 레코드의 afterImg

로 트랜잭션을 철회한 뒤, 'write' 록을 해제한다.

각 클라이언트에서 완료한 트랜잭션에 의해서 갱신된 페이지가 서버로 전송될 때는 해당 페이지의 LSN을 B&L_Tbl의 LSN에 기록한다.

3.2 클라이언트 파손 시 회복 동작

클라이언트 파손 시 회복해야 하는 페이지들은 완료한 트랜잭션에 의해서 갱신된 페이지가 서버로 전송되지 않은 것들이다. 따라서, 서버가 클라이언트 파손을 인지한 다음, 자신의 B&L_Tbl에서 파손된 클라이언트에 의해서 write 록이 걸려 있는 페이지를 재시작한 클라이언트에 모두 전송하고, 재시작한 클라이언트의 부회복관리기는 자치적으로 (그림 2)와 같이 후방향 로그 분석을 이용한 재수행 회복 과정을 수행한다.

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

(그림 2) 후방향 클라이언트 로그 분석을 이용한 재수행 회복 알고리즘

라인 3~22는 클라이언트 로그의 마지막 로그 레코드부터 첫 번째 로그 레코드 또는 최근 검사점 로그 레코드에 기록된 redoLSN 로그 레코드까지 후방향으로 로그를 분석하는 반복 구문이다.

로그 분석 시 완료 로그 레코드를 분석할 경우에는 해당 트랜잭션의 식별자를 완료한 트랜잭션 리스트(commitLst)에 추가한다(라인 5).

별도의 after-image 로그 레코드를 분석할 경우에는 다음을 처리한다.

- 로그 레코드의 페이지 식별자가 자신의 B&L_Tbl에 존재하면서 해당 페이지가 갱신되지 않았다면 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 8~10).

갱신 로그 레코드를 분석할 경우에, 해당 로그 레코드의 트랜잭션 식별자가 commitLst에 존재하는 경우에만 다음의 ①, ②를 차례대로 처리한다.

① 별도의 after-image 로그 레코드를 분석하는 경우와 같이 라인(14~16)을 수행한다.

② 로그 레코드의 trID에서 예약된 상위 1비트가 '1'이면 해당 트랜잭션의 첫 번째 갱신 연산을 뜻함으로 commitLst에서 해당 트랜잭션 식별자를 삭제한다(라인 17).

검사점 로그 레코드를 분석할 경우, 마지막으로 분석할 로그 레코드의 LSN을 의미하는 redoLSN을 firstLSN에 기록한다(라인 19).

클라이언트의 부회복관리기는 (그림 2)와 같이 자치적으로 후방향 클라이언트 로그 분석을 이용하여 손실된 페이지를 한번의 재수행으로 회복시킬 수 있고, 자신의 로그만을 사용하기 때문에 로그 분석의 소요시간을 단축시킬 수 있다.

3.3 서버 파손 시 회복 동작

서버 파손 시, 제안한 회복 기법은 각 클라이언트의 로그를 합병하지 않고 회복하기 위해서 서버에서 최근 완료 로그 레코드를 분석하여 회복해야 하는 페이지의 after-image를 해당 클라이언트에 요청하고 있다(그림 3).

재시작한 서버는 회복 관리기에 의해서 (그림 3)과 같이 후방향 서버 로그 분석을 이용하여 재수행 및 회

복해야 할 페이지의 after-image를 해당 클라이언트에 요청한다.

```
1 : firstLSN := 0;
2 : logRec := LAST_LOG();
3 : DO {
4:      SWITCH (logRec.type) {
5 :         CASE 'recCommit' : CALL REQUEST_AFTERIMG
               (logRec);
6:          BREAK;
7:         CASE 'commit' : ADD(commitLst, logRec.trID);
8:          BREAK;
9:         CASE 'afterImg' :
10:         IF ((B&L_Tbl[logRec.pgID].LSN & '0111...1')
               < logRec.LSN) {
11:          REDO(B&L_Tbl[logRec.pgID].index,
             logRec.afterImg);
12:          MODIFY(B&L_Tbl[logRec.pgID], *, '1', *,
             *, *, *, logRec.LSN, logRec.LSN);
            } // IF ((B&L_Tbl[logRec.pgID].LSN &
                  '0111...1') < logRec.LSN)
13:         BREAK;
14:        CASE 'update' :
15:         IF (EXIST(commitLst, logRec.trID) {
16:          IF ((B&L_Tbl[logRec.pgID].LSN & '0111...1')
               < logRec.LSN) {
17:           REDO(B&L_Tbl[logRec.pgID].index,
              logRec.afterImg);
18:           MODIFY(B&L_Tbl[logRec.pgID], *,
              '1', *, *, *, *, logRec.LSN,
              logRec.LSN);
             } // IF ((B&L_Tbl[logRec.pgID].LSN &
                  '0111...1') < logRec.LSN)
19:          IF (logRec.trID(1) = '1') { DEL(commitLst,
              logRec.trID); }
            } // IF (EXIST(commitLst, logRec.trID)
20:         ELSE {
21:          IF ((B&L_Tbl[logRec.pgID].LSN & '0111...1')
               = logRec.LSN) {
22:           MODIFY(B&L_Tbl[logRec.pgID], *,
              '1', *, *, *, '0', *);
             } // IF ((B&L_Tbl[logRec.pgID]. LSN
                  & '0111...1') = logRec.LSN)
            }
23:         BREAK;
24:        CASE 'ckp' : firstLSN := logRec.redoLSN;
25:         BREAK;
        } // SWITCH
26:    logRec := PREV_LOG();
27 : } WHILE (logRec IS NOT BOF AND logRec.LSN =
          firstLSN)
```

(그림 3) 후방향 서버 로그 분석을 이용한 재수행 및 afterImg 요청 알고리즘

라인 3~27은 서버 로그의 마지막 로그 레코드부터 첫 번째 로그 레코드 또는 최근 검사점 로그 레코드에 기록된 redoLSN 로그 레코드까지 후방향으로 로그를 분석하는 반복 구문이다.

로그 분석 시, 완료 로그 레코드와 검사점 로그 레코드를 분석할 경우는 (그림 2)의 후방향 클라이언트 로그 분석을 이용한 재수행 회복 알고리즘과 같다.

별도의 after-image 로그 레코드를 분석할 경우, 로그 레코드가 해당 페이지에 반영되지 않았으면 로그 레코드의 after-image를 이용하여 재수행한 후, B&L_Tbl에서 해당 페이지의 dirty('1'), LSN(logRec.LSN), prevLSN (logRec.LSN)을 수정하여 더 이상 재수행 하지 않도록 한다(라인 10~12).

갱신 로그 레코드를 분석할 경우, 해당 로그 레코드의 트랜잭션 식별자가 commitLst에 존재하면 ①(라인 16~19)을, 그렇지 않으면 ②(라인 21~22)를 처리한다.

① 별도의 after-image 로그 레코드 분석 시 라인 10~12와 같이 수행한 다음(라인 16~17), 로그 레코드의 trID에서 예약된 상위 1비트가 '1'이면 해당 트랜잭션의 첫 번째 갱신 연산을 뜻함으로 commitLst에서 해당 트랜잭션 식별자를 삭제한다(라인 19).

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).

LSN이 로그 레코드의 LSN보다 작은 경우(즉, 해당 페이지에 갱신 연산이 반영되지 않은 경우)는 ③을 처리하고, 그렇지 않은 경우는 ④를 처리한다.

③ 페이지의 after-image 요청 여부를 나타내는 플래그 변수(flagReq)가 false인 경우 true로 변경한다. B&L_Tbl에서 해당 페이지의 lock('write'), LSN(logRec.LSN), prevLSN(logRec.LSN)을 수정하여, 클라이언트로부터 해당 페이지의 after-image 수신 시 재수행할 수 있게 한다.

④ 해당 페이지는 회복할 필요가 없기 때문에 로그 레코드의 afterImg에서 해당 페이지 식별자를 논리적으로 삭제한다.

서버에서는 손실된 페이지의 재수행 및 after-image를 요청하는 중에, 다음 (그림 4)에 의해서 각 클라이언트로부터 요청한 손실된 페이지의 after-image를 수신하면 바로 재수행 한다.

서버 파손을 인지한 각 클라이언트에서는 우선 부회복관리기에 의해서 (그림 4)와 같이 후방향 클라이언

트 로그 분석을 이용하여 완료한 트랜잭션에 대한 갱신 로그 레코드의 afterImg를 클라이언트 버퍼에 미리 적재함으로써 서버에서 회복할 페이지의 after-image 요청 시 바로 전송할 수 있게 한다.

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

(그림 4) 후방향 클라이언트 로그 분석을 이용한 after-image 적재 알고리즘

(그림 4)는 라인 8~10, 14~16을 제외하고 클라이언트 파손 시 자치적으로 수행하는 (그림 2)의 후방향 클라이언트 로그 분석을 이용한 재수행 회복 알고리즘과 같다. (그림 4)에서는 완료한 트랜잭션에 대한 갱신 로그 레코드의 afterImg를 버퍼에 적재하기 때문에 라인 8~10, 14~16에서는 다음을 처리한다.

- 로그 레코드의 페이지 식별자가 자신의 B&L_Tbl에 존재하지 않으면 B&L_Tbl에 로그 레코드의 페이지 식별자를 추가하고(라인 9, 15), 로그 레코드의 afterImg를 버퍼에 적재한다(라인 10, 16).

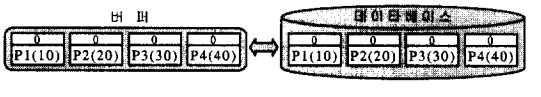
(그림 4)에 의해서 완료한 트랜잭션에 대한 갱신 로그 레코드의 afterImg를 버퍼에 적재하는 도중 서버로부터 회복할 페이지의 식별자들을 수신하면 버퍼 상에서 해당 페이지의 after-image를 전송한다. 그리고, (그림 4) 수행 중에 버퍼가 가득 차서 더 이상 갱신 로그 레코드의 afterImg를 적재할 수 없는 경우에는 서버로부터 수신된 페이지 식별자들을 버퍼 상에서 해당 페이지의 after-image를 검색하고, after-image가 없을 경우 B&L_Tbl의 모든 엔트리를 삭제하여 계속해서 갱신 로그 레코드의 afterImg를 버퍼에 적재한다 (단, 서버로부터 수신된 페이지 식별자가 없는 경우는 페이지 식별자를 수신할 때까지 기다린다).

서버 파손 시 제안한 회복 기법은 병행적으로 각 클라이언트에서는 후방향 클라이언트 로그 분석을 이용하여 회복해야 하는 페이지의 after-image만을 서버로 전송하고, 동시에 서버에서는 후방향 서버 로그 분석을 이용하여 손실된 페이지를 재수행하고 회복해야 하는 페이지의 after-image를 해당 클라이언트에 요청하여 수신된 페이지의 after-image를 이용하여 재수행한다. 따라서 각 시스템에서의 병행적 로그 분석, 후방향 로그 분석을 이용한 재수행 회복으로 회복에 대한 서버의 작업부하와 로그 분석의 소요 시간을 감소 시켰다. 그리고 각 클라이언트에서는 회복해야할 페이지의 after-image만을 전송함으로써 회복 시 필요한 네트워크 트래픽을 최소화했다.

4. 제안한 회복 기법과 기존 회복 기법의 비교 분석

여기서는 기존의 ESM/CS[2], EOS[7], 그리고 Client-Based Logging[8] 기법들과 제안한 회복 기법에 <표 5>의 트랜잭션 수행 예제를 적용하여 비교 분석한다. 비교를 위해서 다음과 같이 가정한다.

- 트랜잭션이 페이지 Pi(n)를 갱신하면, Pi의 after-image는 Pi(n+1)이다(n은 양의 정수).
- 현재의 버퍼와 데이터베이스 상태는 (그림 5)와 같다.



(그림 5) 현재의 버퍼와 데이터베이스 상태

- 서버 로그에서 첫 번째 로그 레코드의 LSN은 100이다(클라이언트1에서 첫 번째 로그 레코드의 LSN은 200, 클라이언트2에서 첫 번째 로그 레코드의 LSN은 300이다).

- 적용할 트랜잭션 수행 예제는 <표 5>와 같다.
- TIME 11의 시스템 파손 직전에 디스크에 반영된 페이지 상태는 P1(11), P2(22), P3(31), P4(42) 이다.
- TIME 11의 시스템 파손 직전에 버퍼 상의 페이지 상태는 P1(13), P2(24), P3(32), P4(42) 이다.
- 시스템 파손 직전에 일관된 페이지의 상태는 P1(13), P2(23), P3(32), P4(42) 이다.
- TIME 11의 시스템 파손 직전에 기존 회복 기법[2, 7, 8]의 로그 상태는 <표 6, 7, 8>이고, 제안한 회복 기법의 로그 상태는 <표 9>와 같다.

〈표 5〉 트랜잭션 수행 예제

TIME	SERVER		CLIENT1		CLIENT2	
	EVENT	BUFFER	EVENT	BUFFER	EVENT	BUFFER
1		P1(10) P2(20) P3(30) P4(40)	Update(T201, P1)	P1(11)	Update(T301, P3)	P3(31)
2	Update(T101, P4)	P1(10) P2(20) P3(30) P4(41)	Update(T201, P2)	P1(11) P2(21)	Abort(T301)	P3(30)
3	Commit(T101)	P1(11) P2(21) P3(30) P4(41)	Commit(T201)	P1(11) P2(21)	Update(T302, P3)	P3(31)
4	검사점 수행 시작	P1(11) P2(21) P3(30) P4(41)	Update(T202, P4)	P1(11) P4(42)	Update(T302, P2)	P2(22) P3(31)
5	Flush(P1, P2) by 검사점	P1(11) P2(22) P3(31) P4(41)	Update(T202, P1)	P1(12) P4(42)	Commit(T302)	P2(22) P3(31)
6	Flush(P3, P4) by 검사점	P1(12) P2(22) P3(31) P4(42)	Commit(T202)	P1(12) P4(42)	Update(T303, P2)	P2(23) P3(32)
7	검사점 수행 완료 redoLSN 기록	P1(12) P2(23) P3(31) P4(42)	Update(T203, P1)	P1(13) P4(42)	Commit(T303)	P2(23) P3(32)
8		P1(12) P2(23) P3(31) P4(42)	Update(T203, P3)	P1(13) P3(32) P4(42)	Update(T304, P2)	P2(24)
9		P1(13) P2(23) P3(32) P4(42)	Commit(T203)	P1(13) P3(32) P4(42)	Abort(T304)	P2(23)
10	Update(T102, P2)	P1(13) P2(24) P3(32) P4(42)	Update(T204, P1)	P1(14) P3(32) P4(42)		
11	시스템 파손					

〈표 6〉 ESM/CS의 로그 상태

LSN	type	nodeID.trID	pgID	LRC	prevLSN	UndoNxtLSN	Redo_Op	Undo_Op
100	'update'	C1.T201	P1	101	0		Redo_Op(P1(11))	Undo_Op(P1(10))
101	'update'	C2.T301	P3	301	0		Redo_Op(P3(31))	Undo_Op(P3(30))
102	'update'	S.T101	P4	401	0		Redo_Op(P4(41))	Undo_Op(P4(40))
103	'update'	C1.T201	P2	201	100		Redo_Op(P2(21))	Undo_Op(P2(20))
104	'compensation'	C2.T301	P3	302	101	0		Undo_Op(P3(30))
105	'commit'	S.T101			102			
106	'commit'	C1.T201			103			
107	'update'	C2.T302	P3	303	0		Redo_Op(P3(31))	Undo_Op(P3(30))
108	'ckp_begin'							
109	'update'	C1.T202	P4	402	0		Redo_Op(P4(42))	Undo_Op(P4(41))
110	'update'	C2.T302	P2	202	107		Redo_Op(P2(22))	Undo_Op(P2(21))
111	'update'	C1.T202	P1	102	109		Redo_Op(P1(12))	Undo_Op(P1(11))
112	'commit'	C2.T302			110			
113	'commit'	C1.T202			111			
114	'update'	C2.T303	P2	203	0		Redo_Op(P2(23))	Undo_Op(P2(22))
115	'ckp_end'	Flush(P1(11), P2(22), P3(31), P4(42)), redoLSN := 111						
116	'update'	C1.T203	P1	103	0		Redo_Op(P1(13))	Undo_Op(P1(12))
117	'commit'	C2.T303			114			
118	'update'	C1.T203	P3	304	116		Redo_Op(P3(32))	Undo_Op(P3(31))
119	'update'	C2.T304	P2	204	0		Redo_Op(P2(24))	Undo_Op(P2(23))
120	'commit'	C1.T203			118			
121	'compensation'	C2.T304	P2	205	119	0		Undo_Op(P2(23))
122	'update'	S.T102	P2	206	0		Redo_Op(P2(24))	Undo_Op(P2(23))
123	'update'	C1.T204	P1	104	0		Redo_Op(P1(14))	Undo_Op(P1(13))

〈표 7〉 EOS의 전역(global) 로그 상태

LSN	type	nodeID.trID	pgID	afterImg
100	'update'	S.T101	P4	P4(41)
101	'commit'	S.T101		
102	'update'	C1.T201	P1	P1(11)
103	'update'	C1.T201	P2	P2(21)
104	'commit'	C1.T201		
105	'ckp_begin'			
106	'update'	C2.T302	P3	P3(31)
107	'update'	C2.T302	P2	P2(22)
108	'commit'	C2.T302		
109	'update'	C1.T202	P4	P4(42)
110	'update'	C1.T202	P1	P1(12)
111	'commit'	C1.T202		
112	'update'	C2.T303	P2	P2(23)
113	'commit'	C1.T202		
114	'ckp_end'	Flush(P1(11), P2(22), P3(31), P4(42)), redoLSN := 110		
115	'update'	C1.T203	P1	P1(13)
116	'update'	C1.T203	P3	P3(32)
117	'commit'	C1.T203		

〈표 8〉 Client-Based Logging의 로그 상태

서버 로그 상태								
LSN	type	trID	pgID	PSN	prevLSN	UndoNxtLSN	Redo_Op	Undo_Op
100	'update'	T101	P4	401	0		Redo_Op(P4(41))	Undo_Op(P4(40))
101	'commit'	T101			100			
102	'ckp_begin'							
103	'ckp_end'	Flush(P1(11), P2(22), P3(31), P4(42)), redoLSN := 0(DPT 엔트리가 없음)						
104	'update'	T102	P2	206	0		Redo_Op(P2(24))	Undo_Op(P2(23))
클라이언트1 로그 상태								
LSN	type	trID	pgID	PSN	prevLSN	UndoNxtLSN	Redo_Op	Undo_Op
200	'update'	T201	P1	101	0		Redo_Op(P1(11))	Undo_Op(P1(10))
201	'update'	T201	P2	201	200		Redo_Op(P2(21))	Undo_Op(P2(20))
202	'commit'	T201			201			
203	'update'	T202	P4	402	0		Redo_Op(P4(42))	Undo_Op(P4(41))
204	'update'	T202	P1	102	203		Redo_Op(P1(12))	Undo_Op(P1(11))
205	'commit'	T202			204			
206	'update'	T203	P1	103	0		Redo_Op(P1(13))	Undo_Op(P1(12))
207	'update'	T203	P3	304	206		Redo_Op(P3(32))	Undo_Op(P3(31))
208	'commit'	T203			207			
209	'update'	T204	P1	104	0		Redo_Op(P1(14))	Undo_Op(P1(13))
클라이언트2 로그 상태								
LSN	type	trID	pgID	PSN	prevLSN	UndoNxtLSN	Redo_Op	Undo_Op
300	'update'	T301	P3	301	0		Redo_Op(P3(31))	Undo_Op(P3(30))
301	'compensation'	T301	P3	302	300	0		Undo_Op(P3(30))
302	'update'	T302	P3	303	0		Redo_Op(P3(31))	Undo_Op(P3(30))
303	'update'	T302	P2	202	302		Redo_Op(P2(22))	Undo_Op(P2(21))
304	'commit'	T302			303			
305	'update'	T303	P2	203	0		Redo_Op(P2(23))	Undo_Op(P2(22))
306	'commit'	T303			305			
307	'update'	T304	P2	204	0		Redo_Op(P2(24))	Undo_Op(P2(23))
308	'compensation'	T304	P2	205	307	0		Undo_Op(P2(23))

〈표 9〉 제안한 회복 기법의 로그 상태

서버 로그 상태				
LSN	type	nodeID.trID	pgID	afterImg
100	'afterImg'	S.T101	P4	P4(40)
101	'update'	S.T101	P4	P4(41)
102	'commit'	S.T101		
103	'recCommit'	C1.T201		P1, P2
104	'recCommit'	C2.T302		P3, P2
105	'recCommit'	C1.T202		P4, P1
106	'recCommit'	C2.T303		P2
107	'ckp'	Flush(P1(11), P2(22), P3(31), P4(42)), redoLSN := 103		
108	'recCommit'	C1.T203		P1, P3
109	'afterImg'	S.T102	P2	P2(23)
110	'update'	S.T102	P2	P2(24)
클라이언트1 로그 상태				
LSN	type	trID	pgID	afterImg
200	'afterImg'	T201	P1	P1(10)
201	'update'	T201	P1	P1(11)
202	'afterImg'	T201	P2	P2(20)
203	'update'	T201	P2	P2(21)
204	'commit'	T201		
205	'afterImg'	T202	P4	P4(41)
206	'update'	T202	P4	P4(42)
207	'update'	T202	P1	P1(12)
208	'commit'	T202		
209	'update'	T203	P1	P1(13)
210	'afterImg'	T203	P3	P3(31)
211	'update'	T203	P3	P3(32)
212	'commit'	T203		
213	'update'	T204	P1	P1(14)
클라이언트2 로그 상태				
LSN	type	trID	pgID	afterImg
300	'afterImg'	T301	P3	P3(30)
301	'update'	T301	P3	P3(31)
302	'update'	T302	P3	P3(31)
303	'afterImg'	T302	P2	P2(21)
304	'update'	T302	P2	P2(22)
305	'commit'	T302		
306	'update'	T303	P2	P2(23)
307	'commit'	T303		
308	'update'	T304	P2	P2(24)

<표 10>에서는 시스템 회복 시 제안한 회복 기법과 기존 회복기법의 차이점을 기술하였다.

<표 10>에 기술한 비교 분석 내용을 로그 분석 범위, 재수행 동작 회수, 그리고 철회 동작 회수 측면에서 요약하면 다음 <표 11>과 같이 나타낼 수 있다 (<표 11>에서 '없음'은 각 동작이 발생하지 않음을 의미한다).

<표 11>에서 나타난 바와 같이, ESM/CS와 EOS에서는 서버가 전체 로그를 분석하기 때문에 로그 분석 범위가 크다는 것을 알 수 있다. 물론, 제안한 회복 기법에서는 클라이언트1의 경우 14개의 로그 레코드를 분석해야 하지만 서버와 병행적으로 로그를 분석하고, 실제로 5개의 로그 레코드 분석(LSN(213)~LSN (209)) 만으로 클라이언트 1에서의 회복 동작이 끝난다. 또한, 클라이언트2는 자치적으로 9개의 로그 레코드를 분석할 뿐 시스템의 회복에는 참여하지 않는다. 여기서는 하나의 서버와 두 개의 클라이언트들로 제한하고 있지만 더 많은 클라이언트들에서 트랜잭션

〈표 10〉 회복 동작의 비교

	로그 분석 동작 및 범위		시스템 회복 동작	기타 차이점
ESM/CS[2]	LSN(111)~LSN(123)		1. LSN(111) 재수행 2. LSN(116) 재수행 3. LSN(118) 재수행 4. LSN(119) 재수행 5. LSN(121) 재수행 6. LSN(122) 재수행 7. LSN(123) 재수행 8. LSN(123) 철회 9. LSN(122) 철회	· 논리적 로깅으로 로깅 오버헤드 감소 · 로그 레코드 전송 비용 존재 · 재수행 및 철회 동작이 많음 · 회복 시 3번의 로그 접근
EOS[7]	LSN(110)~LSN(117)		1. LSN(110) 재수행 2. LSN(112) 재수행 3. LSN(115) 재수행 4. LSN(116) 재수행	· 재수행 전용 로그를 구성하기 위해 각 트랜잭션마다 별도의 전용 로그 버퍼 존재 · 로그 레코드 전송 비용 존재 · 불필요한 재수행 존재
Client-Base Logging[8]	Server	LSN(100)~LSN(104)	1. 서버가 소유하는 페이지 식별자(P1, P2,P3,P4)를 각 클라이언트에 전송 2. 각 클라이언트로부터 DPT 정보 수신 시, 회복해야 하는 페이지(P2)와 회복에 참여하는 클라이언트(Client2) 결정 3. 회복에 참여하는 클라이언트(Clie nt2)에 회복해야 하는 페이지(P2)의 NodePSNList 요청 4. 각 클라이언트로부터 NodePSN List 수신 시, 회복해야 하는 페이지(P2)별로 PSN의 오름 순서로 정렬함 5. (PSN의 오름 순서에 따라서) P2(22)를 Client2에 전송함 6. (PSN의 오름 순서에 따라서) P2(23) 수신시, LSN(104) 재수행 7. LSN(105) 철회 /* compensation 로그 레코드 생성 및 기록 */	· 논리적 로깅으로 로깅 오버헤드 감소 · 로그 레코드 전송 비용 제거 · 재수행 및 철회 동작이 많음 · 회복 시 3번의 로그 접근 · 각 클라이언트 버퍼에 존재하는 일관된 페이지 이용 · 회복 시 네트워크의 트래픽 증가(서버가 소유하는 모든 페이지 식별자, 각 클라이언트의 DPT 정보, 각 클라이언트의 NodePSNList 전송)
	Client1	없음	1. LSN(209) 철회 /*compensation 로그 레코드 생성 및 기록 */ 2. 서버가 소유하는 페이지 식별자 수신 시, 버퍼 페이지 P1(13), P3(32) 전송	
	Client2	LSN(307)~LSN(308)	1. DPT[P2] 정보를 서버로 전송 2. 서버로부터 NodePSNList 요청시, 로그를 분석하여 NodePSN List [PSN of P2(203,204)]를 생성하고 서버로 전송 3. LSN(305) 재수행 4. LSN(307) 재수행 5. LSN(308) 재수행 6. P2(23)을 서버로 전송	
제안한 회복 기법	Server	LSN(110)~LSN(103)	1. LSN(109) 재수행 2. LSN(108) P1, P3의 after-image 요청 3. 수신된 P3의 after-image 재수행 4. 수신된 P1의 after-image 재수행	· 로그 레코드 전송 비용 제거 · 중복된 before-image 제거 · B&L_Tbl의 prevLSN 유지 및 관리, 별도의 after-image 로그 레코드의 로깅 · 회복 시 commitLst 자료구조 유지 및 관리, 네트워크 트래픽 존재 · 클라이언트 파손 시 자신의 로그만 이용 · 서버 파손 시 병행적인 로그 분석
	Client1	LSN(213)~LSN(200)	1. LSN(211) P3의 after-image 전송 2. LSN(209) P1의 after-image 전송	
	Client2	LSN(308)~LSN(300)	없음	

이 발생하면 로그 분석 범위에 대한 차이는 더욱 커진다.

〈표 11〉 회복 동작의 비교 분석

	로그 분석 범위		재수행 동작 회수	철회 동작 회수
ESM/CS[2]	17 개		8 회	2 회
EOS[7]	8 개		4 회	없음
Client-Base Logging[8]	Server	5 개	1 회	1 회
	Client1	0 개	0 회	1 회
	Client2	4 개	3 회	0 회
제안한 회복 기법	Server	6 개	3 회	없음
	Client1	14 개	없음	없음
	Client2	9 개	없음	없음

서버 파손 시에도 제안한 회복 기법은 후방향 로그 분석을 이용해서 한번의 로그 접근으로 갱신된 페이지에 대해 단지 한번만 재수행 하기 때문에 재수행 동작 회수가 작음을 알 수 있다. 반면에 전방향 로그 분석을 이용한 회복 기법들은 많은 재수행 및 철회 동작이 있으며, 특히 EOS의 경우, 불필요한 재수행(LSN(110) 재수행, <표 10>)이 존재하고, ESM/CS와 Client-base Logging의 경우에는 로그 분석 이후에 두 번 더 로그에 접근해야 한다.

<표 6, 7, 8, 9>에서 각 회복 기법의 로그 상태를 살펴보면, EOS가 가장 작은 수의 로그 레코드를 가지고 있는데, 이것을 위해서 서버에서는 각 트랜잭션마다 별도의 전용 로그 버퍼를 할당해 주고 있다[13]. 이에 반해, 제안한 회복 기법에서는 각 트랜잭션마다 별도의 전용 로그 버퍼를 두지 않는다. 또, 불필요한 before-image를 제거하고 재수행 전용 로그만을 구성한다. 로그 관리 측면에서도 서버에서 전체 로그를 관리하는 기존 회복 기법들은 로그 레코드 전송 비용이 존재한다[5, 9, 14]. 하지만, 제안한 회복 기법은 클라이언트 로깅을 이용하여 로그 레코드 전송 비용을 제거했다. 클라이언트 로깅을 지원하는 기존 기법은 서버 파손 시 각 클라이언트의 로그를 합병하지 않고 회복하기 위해서 각 클라이언트와 많은 통신을 필요로 한다[9]. 제안한 회복 기법에서는 회복해야 하는 페이지의 after-image만을 서버로 전송하기 때문에 회복 시 필요한 통신 양을 최소화했다.

제안한 회복 기법은 후방향 로그 분석을 수행하기 때문에 시스템 회복 시 commitLst 자료구조를 유지 및 관리하는 오버헤드가 있지만, commitLst의 엔트리를 동적으로 추가 삭제함으로써 오버헤드를 최소화했다. 로깅 시 중복된 before-image를 제거하기 위해서 B&L_Tbl에서 prevLSN을 유지 및 관리하고, 경우에 따라서 별도의 after-image를 기록해야하는 단점을 가지고 있다. 그리고 서버 회복 시 네트워크 트래픽을 최소화하기 위해 서버에서는 최근 완료 로그 레코드를 생성하여 기록하는데 오버헤드가 있지만, 이는 각 클라이언트에서 생성된 로그 레코드를 전송하는 비용보다는 적다.

5. 결 론

제안한 회복 기법에서는 불필요한 before-image를 제거하여 재수행 전용 레코드만을 로그에 기록 관리함으로써 로깅 오버헤드를 감소시켰으며, 클라이언트 기반 로깅을 지원하여 로그 레코드의 전송 비용을 제거하였다. 클라이언트 파손 시 자치적으로 한번의 후방향 클라이언트 로그 분석을 이용하여 손실된 페이지의 after-image를 식별하고 바로 재수행 하도록 했다. 그리고 서버 파손 시 각 클라이언트의 로그를 합병하지 않고 회복하기 위해서, 서버에서는 후방향 서버 로그 분석을 이용하여 최근 완료 로그 레코드에서 회복해야 하는 페이지의 after-image를 식별하여 요청하고, 수신된 페이지의 after-image를 이용하여 재수행 한다. 동시에, 각 클라이언트에서는 병행적으로 후방향 클라이언트 로그 분석을 이용하여 서버로부터 요청된 페이지의 after-image만을 전송한다. 따라서, 제안한 회복 기법은 클라이언트의 자치적 회복을 지원하며, 서버 회복 시 각 클라이언트에서 병행적 로그 분석으로 로그 분석의 소요 시간을 단축시킴으로써 빠른 회복을 기대할 수 있다. 또한, 각 클라이언트에서는 회복해야 하는 페이지의 after-image만을 서버로 전송하기 때문에 회복 시 네트워크 트래픽을 최소화했다. 그리고, 후방향 로그 분석을 이용한 기존 회복 기법에서, 회복 시에 유지 관리해야 하는 자료구조의 접근 오버헤드를 동적으로 추가 및 삭제되는 commitLst에 의해서 최소화했다.

추후 연구 과제로는 본 논문에서 제안한 로깅 및 회복 알고리즘에 대한 정확한 성능 평가가 이루어져야 하며, 현재 구현 중에 있다.

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] David Lomet, Gerhard Weikum, "Efficient Transparent Application Recovery in Client-Server Information Systems," Proc. ACM SIGMOD, pp.460-471, 1998.

[4] 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.

[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.

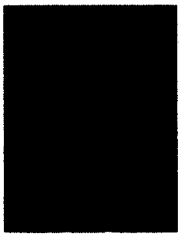
[9] E. Panagos, A. Biliris, H. V. Jagadish, R. Restogi, "Client-Based Logging for High Performance Distributed Architectures," pp.344-351, 1996.

[10] David Lomet, Mark Tuttle, "Redo Recovery after System Crashes," Proc. VLDB, pp.457-468, 1995.

[11] David Lomet, "Persistent Applications Using Generalized Redo Recovery," IEEE ICDE, pp.154-163, 1998.

[12] C. Mohan & I Narang, "ARIES/CSA : A Method for Database Recovery in Client-Server Architectures," Proc. ACM SIGMOD, pp.55-66, 1994.

[13] E. Panagos, A. Biliris, "Synchronization and recovery in a client-server storage system," The VLDB Journal, pp.209-223, 1997.



박 용 문

e-mail : ympark@etri.re.kr

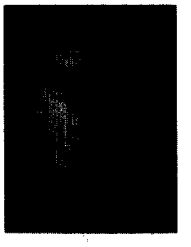
1983년 숭전대학교 계산통계학과 졸업(학사)

1985년 중앙대학교 대학원 전자계산학과(이학석사)

2000년 한남대학교 대학원 컴퓨터공학과(박사과정)

1985년~현재 한국전자통신연구원 선임연구원

관심분야 : 이동컴퓨팅, 실시간 데이타베이스



이 찬 섭

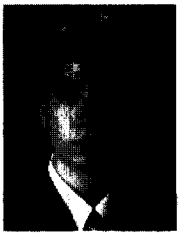
e-mail : dblab@hannam.ac.kr

1990년 한남대학교 컴퓨터공학과 졸업(학사)

2000년 한남대학교 대학원 컴퓨터공학과(공학석사)

2000년 한남대학교 대학원 컴퓨터공학과(박사과정)

관심분야 : Data mining, Web DB, XML



김 희 수

e-mail : first@it4web.com

1998년 한남대학교 컴퓨터공학과 졸업(학사)

2000년 한남대학교 대학원 전자계산학과(공학석사)

2000년~현재 it4web(주) 연구원

관심분야 : 클라이언트/서버 데이터베이스, 이동컴퓨팅

최 의 인

e-mail : eichoi@.hannam.ac.kr

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

관심분야 : 실시간데이타베이스, 주기억데이타베이스, 클라이언트/서버 데이타베이스