클라이언트/서버 데이타베이스 시스템에서 상태 정보 테이블과 퍼지 검사점을 이용한 로그 관리 및 회복 기법

최 의 인[†] · 고 병 오[††]

요 약

클라이언트/서버 환경에서는 서버의 구성이 디스크 데이타베이스 시스템으로 이루어지기 때문에 디스크 입출력의 증가로 서버의 병목 현상이나 오버헤드가 증가되며, 이로 인해 실시간 시스템의 경우에는 빠른 트랜잭션 처리 및 응답을 할 수 없는 문제가 발생하고 있다. 이 논문에서는 서버 시스템을 주 기억 데이타베이스 시스템으로 구성하여 디스크 입출력을 감소시켜 서버의 오버헤드를 줄이고, 빠른 응답을 제공하기 위한 로그 관리 및 회복 기법을 제안하였다. 특히 서버의 환경을 주 기억 데이타베이스 시스템으로 구성하였으므로 이에 대한 로깅 및 회복 기법을 중점적으로 다루었다. 1) 로그 관리 측면에서는 지연 로깅을 사용하면서 상태 정보 테이블을 제안하여 로그 관리나 트랜잭션 관리의 오버헤드를 감소시켰다. 2) 검사점 기법은 기존의 퍼지 검사점 기법이 가지고 있는 문제점인 데이타베이스의 불일치를 해결할 수 있도록 개선된 퍼지 검사점 기법을 제안하였다. 디스크 입출력 감소를 위해 디스크 로그를 사용하지 않고 배터리 백업 로그를 사용하였다. 이 환경을 토대로 하여 클라이언트/서버 간의 데이타 갱신 동작 및 파손시의 회복 동작에 대한 시나리오와 수행 과정을 제안하였다. 파손시의 회복 동작에 있어서도 빠른 회복을 위해 클라이언트도 자신의 상태 정보를 파악하여 주기적으로 서버에게 전송하도록 하였다. 마지막으로 ESM-CS와 ARIES/CSA와 비교 분석하였다.

Log Management and Recovery Technique Using Status Information Table and Fuzzy Checkpoint in Client/Server Database Systems

Eui In Choi[†] · Byung Oh Koh[††]

ABSTRACT

In this paper, we propose improved techniques about logging, checkpointing, and recovery in a Client-Server database environment.

※이 논문은 1997년도 한남대학교 교비 학술연구 조성비 지원에 의하여 연구되었음.

† 정 회 원:한남대학교 컴퓨터공학과
†† 정 회 원:공주교육대학교 컴퓨터교육학과

논문접수:1996년 9월 5일, 심사완료:1997년 12월 8일

In order to improve the performance of system, a) the log management and processing reduce by the Status Information Table(SIT), b) the dirty pages at clients are written in the status information table when client log records arrive at the server. Our checkpoint scheme is that the server and the clients take checkpoints periodically to ensure correct operations.

The key features of our scheme are: 1) the server only contains Main-memory databases and a battery backup device log; this reduce the number of input/output activities for backup and recovery. 2) the client perform checkpoint; this makes the server acknowledge the page dirtied by the client. 3) database consistency guarantee even during checkpoint using Improved Fuzzy checkpoint. Finally, our system was compared with ESM-CS and ARIES/CSA.

1. 서 론

최근 컴퓨터 하드웨어 및 통신 장비의 발전과 가격 하락은 데이타베이스 시스템에 많은 영향을 주었다. 또한, 워크스테이션과 같은 저렴한 컴퓨터의 개발과, LAN과 같은 대용량의 데이타를 고속으로 전송시킬 수 있는 통신 장비의 지속적인 발전과 가격 인하로 컴퓨팅 환경에 많은 변화가 일어나고, 이에 따라 클라이언트/서버 컴퓨팅이 여러 응용 분야에서 활용되고 있다. 또한, 데이타베이스 시스템도 이전의 중앙 집중 식이나 분산 데이타베이스 시스템 환경에서 운용되는 것 보다, 클라이언트/서버 환경에서 데이타베이스를 유지 및 관리해주는 기능으로 변화되고 있다. 클라이언트/서버 데이타베이스 시스템은 서버가 계산 처리, 화일 관리 및 서비스를 하고, 클라이언트는 서버에게 제공받은 서비스를 이용하여 응용을 처리하고 사용자 인터페이스를 통해 사용자와 접속하는 것이다. 그러나 이 시스템은 클라이언트와 서버의 역할이 분리되어 있는 상태에서 트랜잭션이 처리되기 때문에 여러 종류의 파손(시스템 파손, 미디어 파손)이나 클라이언트에서 처리된 데이타 페이지에 대한 갱신 연산 결과와 서버에서 유지되는 데이타 페이지 사이에 불일치 등의 문제가 발생될 수 있어 이를 해결하기 위한 연구가 절실하게 요구되고 있는 실정이다. 클라이언트/서버 환경은 데이타 페이지에 대한 갱신이 주로 클라이언트에서 이루어지고 진행된 연산에 대한 내용을 로그에 기록하여 갱신 결과와 함께 서버로 전송한다. 그리고 클라이언트와 클라이언트 간의 데이타 페이지의 전달은 클라이언트 간에 직접 전달을 허용하지 않고 서버를 통해서만 이루어진다. 그러므로 기존의 처리 기법과는 다른 클라이언트/서버 데이타베이스 시스템 환경에 맞는 처리 기법이 필요한 실정이다. 클라이언트/서버 데이타베이스 시스템을 운영할 경우에는 다음과 같은 문제점이 발생된다[7].

1) 클라이언트에서 데이타 페이지가 갱신(dirty)되어졌을 때, 서버의 데이타 페이지는 이전 상태로 존재 할 수도 있다.
2) 클라이언트가 로그 레코드를 서버에게 전송하는 시기를 결정하기 어렵다.
3) 클라이언트로부터 로그 레코드와 갱신된 데이타 페이지가 각각 서버에 따로 도착할 때, WAL (Write Ahead Logging) 프로토콜을 유지하기가 어렵다는 등의 문제점이 있다.

시스템 파손시의 회복 기법[1]은 로깅, 검사점, 백업 및 재적재 등으로 구분되어 진행되는데 본 논문에서는 로그 관리 측면에서는 상태 정보 테이블을 이용한 로그 관리 기법을 제안하였다. 이 제안은 기존의 기법이 트랜잭션 관리를 위해 트랜잭션 상태 리스트와 갱신 리스트 또는 완료된 갱신 리스트들을 이용하는데 비하여, 본 제안은 클라이언트/서버 환경에 맞도록 구성하므로써 다른 기법에 비해 효율적인 로그 관리가 이루어지도록 하였다. 검사점(checkpoint)은 퍼지(fuzzy) 검사점 기법이 트랜잭션(transaction)과 액션 일치(action consistent) 검사점 기법[8,9]에 비해 검사점 수행시 동기화 비용이 적고 트랜잭션의 처리가 중단되지 않는 장점이 있지만 데이타베이스의 불일치가 발생할 수 있다는 단점이 있기 때문에 데이타베이스의 일치성을 보장할 수 있도록 본 연구에서는 퍼지 검사점 기법을 개선하였다. 또한, 클라이언트도 주기적으로 자신의 상태 정보를 파악하여 그 내용을 서버에게 전송하도록 하였다. 그 이유는 클라이언트가 주

기적으로 자신이 보유하고 있는 데이타 페이지에 대한 정보와 현재 갱신 연산이 진행중인 트랜잭션에 대한 기록을 서버에게 주기적으로 전송하므로 써 클라이언트 파손시 서버가 이들 정보를 활용하여 빠른 회복을 이루는데 목적이 있기 때문이다.

2. 관련 연구

2.1 클라이언트/서버 시스템의 분류

클라이언트/서버간의 시스템 분류는 크게 다음과 같이 세 가지로 나뉘어진다. 첫째, 클라이언트/서버 간의 데이타 전송 단위 둘째, 클라이언트/서버 간의 질의 처리 방식 셋째, 클라이언트의 메모리 캐쉬 보유 여부 등이다. 첫 번째의 경우는 전송의 단위를 DBMS의 관점에서 서버가 클라이언트로 전송하는 데이타의 단위를 화일, 페이지 또는 세그먼트 같은 물리적 단위, 그리고 객체나 튜플과 같이 논리적 단위의 데이타를 주고받는 시스템으로 구분한다. 이들 중에서 페이지 단위의 전송이 가장 우수한 방법으로 평가되고 있다[3, 4]. 두 번째의 경우는 클라이언트가 질의를 서버에게 전송하여 그 결과를 돌려 받는 질의 전송(query-shipping) 시스템과 클라이언트가 질의 처리에 필요한 특정 데이타 페이지를 서버에게 요청하여 클라이언트가 질의를 처리하는 데이타 전송(data-shipping) 시스템으로 구분 될 수 있다[11]. 세 번째의 경우는 클라이언트에 메모리 캐쉬가 존재하는 가인데 이 경우가 클라이언트/서버 DBMS에서 제일 중요한 관점이다. 클라이언트가 메모리 캐쉬를 가지지 않은 시스템은 일반적인 상용 DBMS을 말하며 데이타베이스의 관리 및 처리를 서버가 모두 전담하는 시스템이다. 그러나 클라이언트가 메모리 캐쉬나 디스크 캐쉬를 가진 시스템은 클라이언트가 메모리 캐쉬를 가지지 않은 시스템과는 달리 트랜잭션의 요청에 의해 클라이언트로 전송되어온 데이타를 나중에 다른 트랜잭션이 재사용이 가능하도록 메모리 캐쉬나 디스크 캐쉬에 저장하는 방법이다. 이 경우는 서버의 오버헤드를 줄이고, 클라이언트와 서버 사이의 데이타 전송에 따른 비용을 절감하며, 클라이언트의 처리 능력 및 저장 능력을 활용한다는 장점이 있다[2, 3].

본 연구에서는 클라이언트와 서버 사이의 데이타 전송은 페이지 단위로 전송하고, 질의의 처리 방식은 클라이언트가 질의에 필요한 특정 데이타를 서버에게 요구하면 서버가 그 데이타를 클라이언트에게 전송하여 클라이언트가 처리하는 데이타 전송 방식으로 운영하며, 클라이언트가 메모리 캐쉬 기능을 가지고 있다고 가정하였다.

2.2 회복 기법

본 연구의 기본 모델인 ESM-CS(Client-Server Exodus Storage Manager system)는 [7]에서 개발한 시스템으로 클라이언트/서버 환경에서의 데이타베이스 시스템으로는 대표적인 시스템이며 다른 많은 연구에서도 인용되었다. ESM-CS는 사용자의 응용 프로그램과 서버에 연결된 클라이언트들로 구성된다. 각각의 클라이언트들은 독립적으로 수행되는 하나의 프로세서로서 자신의 메모리 캐쉬와 록 캐쉬를 가지고 있으며, 한번에 하나의 트랜잭션을 처리한다. 서버는 전체 데이타베이스와 로그를 관리하며, 록 관리 및 시스템 파손 시 회복 동작을 제공한다.

ESM-CS는 ARIES[10]라는 WAL 프로토콜을 보장하는 디스크 데이타베이스 회복 기법을 클라이언트/서버 환경에 맞게 개선하였다. WAL 프로토콜이란 갱신된 데이타 페이지가 디스크 데이타베이스에 기록되기 전에 갱신 결과에 대한 로그가 안정된 로그에 먼저 기록되는 것을 보장하는 기법을 말한다. WAL 프로토콜을 구현하기 위해 ESM-CS는 클라언트가 갱신 연산에 따른 로그를 갱신된 데이타 페이지보다 먼저 서버에 전송하여 기록하도록 하였으나 ARIES를 클라이언트/서버 환경으로 확장시 문제가 발생되어 LRC(Log Record Counter, 한 페이지에 해당하는 모든 레코드에 대해 유일한 값을 갖고 단조 증가하는 성질을 갖는다)를 사용하여 서버에게 데이타 페이지를 전송할 때 로그 레코드의 LRC와 데이타 페이지에 대한 pageLRC를 비교하여 전송하므로 써 문제점을 해결하였다. 그러나 갱신 페이지 리스트(Dirty Page List; DPL)에 대한 기록이 갱신된 데이타 페이지가 서버에 도착해야만 기록되기 때문에 재수행상의 문제가 발생되어 데이타베이스의 일치성을 보장하지 못하는 문제점이 발생한다. ESM-CS는 CDPL(commit dirty page list)를 제안하여 사용하므로 써 해결하였다.

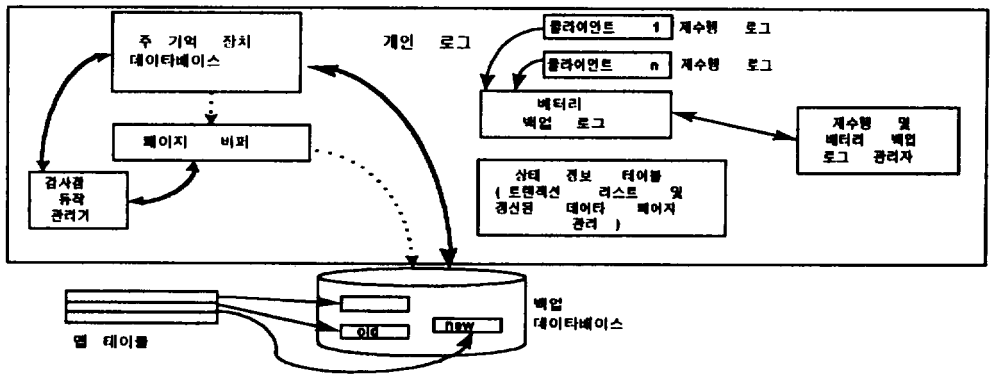
ARIES/CSA는 클라이언트/서버 구조에서 회복 기법을 위해 ARIES를 확장하여 제안하였는데 ARIES

를 기본으로 하였기 때문에 부분적으로는 ESM-CS와 매우 유사하다. 이 기법도 ESM-CS와 같이 WAL을 지원한다. 트랜잭션은 임의의 클라이언트나 서버에서 실행된다. 또한 전역 록 관리 기법을 지원한다. ARIES/CSA는 ESM-CS와 달리 클라이언트와 서버 사이에서 정상완료 전에는 갱신된 내용을 서버에 기록하지 않는다. 즉, 트랜잭션이 종료될때(정상완료 또는 철회) 트랜잭션에 의해 갱신된 페이지는 트랜잭션의 종료가 허용되기전에는 서버에게 보낼 필요가 없다. 물론 트랜잭션은 모든 로그 레코드가 서버에게 보내지고 서버는 안정된 저장 장치에 저장이 된후에만 정상완료라고 선언되어질 수 있다. ARIS/CSA는 ESM-CS가 트랜잭션이 정상완료시는 트랜잭션에 의해 갱신된 모든 페이지는 서버에게(force to server at commit) 전달할 것을 요구하는데 비하여 바로 전달할 것을 요구하지 않는다. 또한 ESM-CS에서는 트랜잭션 갱신을 위해 로그 레코드를 기록하는 것을 제외하고는 임의의 회복 동작을 실행하지 않지만 ARIES/CSA는 임의의 회복 동작을 실시한다[11].

3. 시스템 환경의 제안

이 논문에서 제안한 시스템 환경의 특징은 디스크 입출력으로 인한 서버의 오버헤드를 줄이고 빠른 트랜잭션 처리와 처리율 향상을 위해 ESM-CS, ARIES/CSA와는 달리 서버의 구성을 주기억 데이타베이스[5, 6, 12]로 구성하였다. 백업 로그는 디스크에 유지하지 않고 배터리 백업 로그(battery backup log)를 이용하므로써 디스크 입출력을 또한 배제시켰다. 디스크에는 회복시 필요한 데이타베이스 사본을 저장하고, 클라이언트는 필요한 데이타 페이지를 메모리 캐쉬를 이용하여 저장 사용하므로써, 통신 비용 감소와 페이지 처리로 인한 서버의 오버헤드를 감소시키도록 하였다. 본 연구의 특징은 시스템 파손이 발생할 때 회복 속도를 개선하는데 중점을 둔 것이다. 그래서 로깅은 배터리 백업 로그를 이용하고, 클라이언트도 자신의 상태를 수시로 파악하여(자체적으로 검사점 실행) 현재 진행중인 트랜잭션의 정보와 갱신된 데이타 페이지에 대한 정보를 주기적으로 서버에게 전송하도록 하였다. 서버는 모든 클라이언트로부터 최근의 상태 정보를 전송받으므로써 클라이언트 파손시에 그 정보를 이용하여 빠른 회복이 가능하도록 하였다. 또한, 로그는 클라이언트에서 발생하는 모든 갱신 연산에 대해 서버가 클라이언트로부터 해당 로그 레코드를 전송받아 배터리 백업 로그에 기록하도록 하였다. 그리고 데이타베이스 백업시 데이타베이스 내의 모든 데이타를 동시에 기록하지 않고 페이지 단위로 갱신이 발생된 데이타만을 기록하도록 하여 더욱 효율적인 운영이 이루어지도록 하였다. (그림 1)은 본 연구에서 제안한 서버의 환경을 나타낸다.

서버의 주기억 장치에는 전체 데이타베이스와 배터리 백업 로그, 그리고 각각의 트랜잭션에 대한 개인 로그를 저장하고, 배터리 백업 로그에는 주기억 데이타베이스에 변경이 가해진 재수행 로그 레코드



(그림 1) 서버의 환경
(Fig. 1) Server environment

만을 기록 유지한다. 또한 데이타 페이지에 대한 갱신 여부를 기록하고, 현재 실행중인 트랜잭션을 기록하는 상태 정보 테이블을 유지 관리한다.

본 논문에서 로그는 개인 로그와 배터리 백업 로그를 사용하여 관리하고, 트랜잭션이 처음 실행될 때 각각의 트랜잭션을 위해 개인 로그에 기록한 뒤 트랜잭션 처리가 정상완료되면 배터리 백업 로그에 재수행(redo) 로그만을 기록한다. 철회수행 로그는 사용자에 의해 수행 중이던 트랜잭션을 철회하고자 할 때만 사용하고 배터리 백업 로그에는 기록하지 않는다. 각 트랜잭션의 재수행 로그는 배터리 백업 로그에 저장된 뒤 일정 양이 되면 검사점을 실시하여 로그의 내용을 삭제한다. 데이타베이스도 디스크에 하나의 사본이 유지되도록 하며, 검사점 실행 시마다 맵 테이블의 포인터를 이용하여 갱신된 데이타 페이지만 기록하므로써 모든 데이타 페이지가 기록되는 것보다는 오버헤드를 감소시켰다.

이 논문에서 제안하는 상태 정보 테이블(status information table)은 정상적으로 실행시 데이타 페이지에 대한 갱신 여부를 기록하고, 시스템에 고장이 발생할 경우 재시동되어 회복 동작을 실시할 때 트랜잭션의 상태를 파악하는데 주로 사용된다. 또한 트랜잭션의 상태를 회복시의 동작과 정상 처리 동작으로 구분한다. 회복시에는 로그에 기록된 가장 최근의 검사점 실행 레코드에 기록된 내용으로 초기화 한후, 검사점 로그 레코드부터 로그 레코드를 분석한다. 분석 과정에서 트랜잭션의 상태가 완료나 철회로 간주되면 테이블에서 이들 내용을 삭제하고, 시작 레코드가 발견되면 해당 트랜잭션을 트랜잭션 테이블에 등록하는 방법으로 트랜잭션의 상태를 추적한다. 정상 동작중에는 트랜잭션 관리자에 의해 관리된다. 이 테이블은 TransID, 트랜잭션 상태, 그리고 LastLSN 등으로 구성된다. LastLSN은 트랜잭션에 의해 가장 최근 갱신이 발생된 로그 레코드의 LSN이 기록된다. 또한 페이지의 식별자를 기록하는 PageID, 페이지를 처음으로 갱신 상태로 만든 로그 레코드의 LSN을 기록하는 Firstupdate_LSN이 있다.

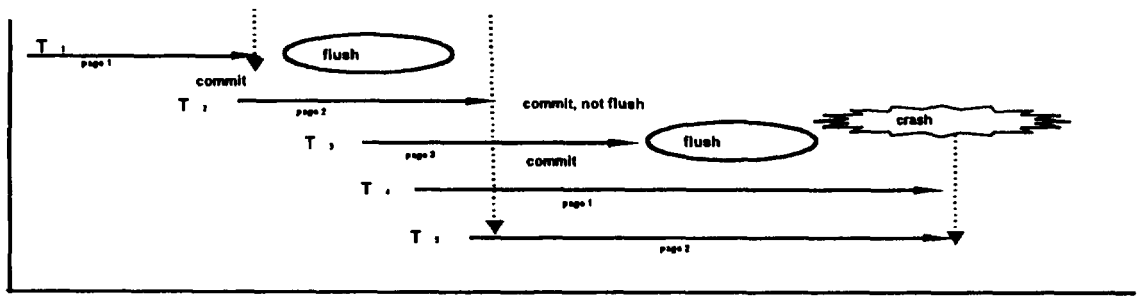
3.1 상태 정보 테이블의 구성

위에서 설명한 상태 정보 테이블의 구성과 생성 내용은 다음과 같다.

위 〈표 1〉의 상태 정보 테이블을 보면 T1 트랜잭션은 현재 트랜잭션의 처리가 완료된 상태이고, 데이타 페이지를 처음으로 갱신한 로그 의 LSN(Firstupdate_LSN)은 0이다. T2의 경우도 트랜잭션의 처리가 완료된 상태이고, 처음 갱신한 로그의 LSN은 0이다. T3도 현재 완료되었고, 바로 전에 발생된 로그의 LSN은 0이다. 그러므로 위의 상태 정보 테이블을 참고하면

〈표 1〉 상태 정보 테이블의 구성
〈Table 1〉 Structure of status information table

TransID	Client_LSN	pageID	LRC	Status	Firstupdate_LSN	LastLSN
T1	C1.100	1	1	C	0	100
T2	C2.200	2	1	C	0	200
T3	C3.300	3	1	C	0	300
T4	*C3.400*	*1*	*2*	*U*	*100*	*400*
T5	*C1.500*	*2*	*2*	*U*	*200*	*500*



(그림 2) 처리를 위한 가정
(Fig. 2) Assumption for processing

T1, T2, T3는 정상완료되었고, T4와 T5의 트랜잭션이 갱신중 이라는 것을 알 수 있다. 또한 가장 나중에 발생된 로그 LSN이 500이라는 것도 알 수 있다. 또한 위의 테이블에서 페이지 1, 2, 3의 경우, 처음으로 갱신 연산을 실시한 로그의 LSN이 100, 200, 300이며 현재 갱신이 이루어지거나 마지막으로 갱신 연산을 실시한 로그 LSN이 각각 400, 500이라는 것을 알 수 있다. 물론 상태 정보 테이블에 추가적으로 들어가는 것도 있지만 위의 테이블을 이용하면 현재 처리가 완료되었거나, 진행중인 트랜잭션의 상태와 갱신이 진행중인 페이지와 그 것에 해당하는 로그 LSN을 알 수 있다.

3.2 클라이언트/서버 간의 회복 기법

이 장에서는 클라이언트가 서버에게 트랜잭션의 처리를 위해 해당 데이타 페이지를 요청하여 처리할 때 발생되는 갱신 동작을 클라이언트와 서버의 동작으로 나누어 설명하고, 클라이언트가 해당 데이타 페이지를 갱신할 때 상태 정보 테이블에 기록하는 과정을 예와 함께 설명한다. 또한 검사점 실행 기법에 대해서도 개선된 검사점을 제안하여 설명하였다. 마지막으로 클라이언트나 서버 사이트 파손시 회복 동작에 대하여 기술하였다.

본 논문에서 제안한 갱신 동작, 상태 정보 테이블 구성을 위해 다음과 같은 가정을 하였다.(그림 2)

1) 세개의 트랜잭션 T1, T2, T3가 현재 실행 중임.
2) 네번째 트랜잭션 T4가 T1이 사용중인 데이타 페이지를 신청하면서 들어 오고 있다. T1은 정상완료되고 검사점이 발생되어 그 결과를 디스크에 기록한 후 데이타 페이지를 T4에게 할당한다.
3) 다섯번째 트랜잭션 T5가 들어 오고 있다.
4) T2는 정상완료는 했으나 디스크에 기록하지 않은 상태에서 T5에게 해당 데이타 페이지를 할당한다.
5) T3는 정상완료 후에 디스크에 결과를 기록.
6) T4, T5가 실행중에 시스템 파손이 발생함.

3.2.1 갱신 동작

클라이언트/서버 간의 갱신 과정은 다음과 같다. 클라이언트는 갱신에 필요한 데이타 페이지를 서버에게 요청하여 갱신 연산을 실행한다. 이때 갱신 연산이 완료되면 클라이언트는 서버에게 생성된 로그 레코드를 즉시 전송한다. 그리고 해당 데이타 페이지는 서버의 요청이 있거나 자신의 메모리 캐쉬의 상태가 다 찬 상태가 될 경우에만 서버에게 데이타 페이지를 전송한다. 이렇게 하므로써 갱신 연산이 끝난 후 매번 데이타 페이지를 서버에게 보냈다가 다시 전송받는 전송 비용을 감소시킬 수 있고, 한 클라이언트에서 다른 트랜잭션이 그 페이지를 사용하게 하므로써 메모리 캐쉬의 효과도 높일 수 있기 때문이다. 아래의 동작은 모든 갱신 연산 동작이 정상완료된다는 가정하에서 실시된다.

[클라이언트의 갱신 동작]

1) 갱신에 필요한 해당 페이지가 메모리 캐쉬에 존재하는 가를 조사한다.
2) 없으면 서버에게 요청한다.
3) 요청한 페이지가 도착하거나, 메모리 캐쉬에 존재하면 갱신 연산을 실시한다.
4) 갱신에 해당하는 로그 레코드를 생성하여 클라이언트의 LRC 필드 값과 갱신된 페이지가 포함된 위치의 LastLSN 필드 값에 현재 Client_LSN 값을 기록한다.
5) 클라이언트는 생성된 로그 레코드를 Client_LSN을 이용하여 서버에게 전송한다.

[서버의 갱신 동작]

1) 클라이언트로부터 전송받은 로그 레코드를 Client_LSN을 통하여 배터리 백업 로그에 기록한다.
2) 상태 정보 테이블을 트랜잭션 식별자(TransID)와 로그 레코드의 pageID를 이용하여 조사한다.
   2.1) 테이블에 없으면
      2.1.1) 트랜잭션 식별자를 이용하여 등록한 뒤 Firstupdate_LSN(가장 최근 것의 LSN)에 로그 레코드의 LSN을 기록한 뒤, PrevLSN 값을 0으로 초기화한다.
      2.1.2) 상태 정보 테이블에 해당 pageID가 존재하지 않으면 pageID를 이용하여 등록한다.
   2.2) 테이블에 있으면
      2.2.1) PrevLSN에 Firstupdate_LSN을 기록한 뒤, Firstupdate_LSN에 로그 레코드

의 LSN을 기록한다.

2.2.2)상태 정보 테이블에 해당 pageID가 존재하면 갱신 연산이 정상완료된 것으로 본다.

이 논문에서 Client_LSN을 제안한 이유는 클라이언트가 로그 레코드를 생성하여 상태 정보 테이블에 로그 레코드의 LSN을 갱신된 페이지의 LastLSN으로 기록할 경우 베터리 백업 로그가 서버에 있으므로써 서버는 해당 클라이언트가 생성한 로그 레코드를 알기위해서는 클라이언트들과 많은 메시지 전송을 해야하는 문제점이 있다. 그러므로 이런 통신 비용의 감소를 위해 클라이언트가 서버에게 데이타 페이지를 요청할 때 페이지와 함께 각각의 클라이언트에 해당하는 Client_LSN을 부여한다. 즉, 서버의 로그에 기록이 이루어질 때 서버는 Client_LSN을 이용하여 어느 클라이언트에서 생성한 로그 LSN 인가를 알 수 있도록 하므로써 통신 비용을 절감할 수 있도록 하였다.

다음은 로그 레코드 기록 과정, 상태 정보 테이블 등록 과정 등을 앞의 예를 이용하여 보면 〈표 2〉, 〈표 3〉과 같다.

〈표 2〉 로그 레코드의 구성 1
〈Table 2〉 Log record 1

Client_LSN	Type	TransID	prevLSN	pageID
C1.100	U	T1	0	1
C2.200	U	T2	0	2
C3.300	U	T3	0	3

〈표 3〉 로그 레코드의 구성 2
〈Table 3〉 Log record 2

Client_LSN	Type	TransID	prevLSN	pageID
C1.100	C	T1	0	1
C2.200	C	T2	0	2
C3.300	C	T3	0	3
C2.400	*U*	*T4*	*100*	*1*
C1.500	*U*	*T5*	*200*	*2*

위의 로그 레코드 구성 1은 초기에 세 개의 트랜잭션이 실행중인 상태를 나타내며, 구성 2는 다시 두 개의 트랜잭션이 들어와 실행되는 상태를 나타낸 것이다. 이 경우 트랜잭션 T1, T2, T3의 상태는 전부 정상완료를 나타내고 있다. 이때 세 개의 트랜잭션이 상태 정보 테이블에 등록되는 과정은 다음과 같다. 먼저 세 개의 트랜잭션 T1과 T2, T3에 대한 등록 여부를 확인한 뒤 등록되어 있지 않으면 등록을 진행한다. 초기에는 처음으로 트랜잭션이 실행되기 때문에 아무 문제 없이 상태 정보 테이블에 등록을 할 수 있다. 과정은 〈표 4〉와 같다.

〈표 4〉 상태 정보 테이블의 구성 1
〈Table 4〉 Status Information Table 1

TransID	Client_LSN	pageID	LRC	Status	Firstupdate_LSN	LastLSN
T1	C1.100	1	1	U	100	100
T2	C2.200	2	1	U	200	200
T3	C3.300	3	1	U	300	300
·	·	·	·	·	·	·
·	·	·	·	·	·	·

이때 위의 테이블에 등록된 트랜잭션은 모두 초기 상태이기 때문에 prevLSN의 값이 전부 0을 가리킨다. 이들 트랜잭션이 정상완료되면 상태 정보 테이블에 등록을 한다. 위의 테이블에는 지금 LSN이 100, 200, 300인 로그 레코드가 각각의 페이지 1, 2, 3을 갱신하여 등록이 되어있고 이들 페이지는 갱신 완료된 상태로 볼 수 있다.

아래 〈표 5〉의 테이블 구성 2를 보면 트랜잭션 T4가 T1이 정상완료된 상태에서 등록을 실시하려는 과정을 보면 페이지 1에 대해 등록을 실시한다. 이때 테이블에 등록 여부를 확인하면 페이지 1이 등록되어

〈표 5〉 상태 정보 테이블의 구성 2
〈Table 5〉 Status Information Table 2

TransID	Client_LSN	pageID	LRC	Status	Firstupdate_LSN	LastLSN
T1	C1.100	1	1	C	0	100
T2	C2.200	2	1	C	0	200
T3	C3.300	3	1	C	0	300
T4	*C2.400*	*1*	*2*	*U*	*100*	*400*
T5	*C1.500*	*2*	*2*	*U*	*200*	*500*

있으므로 LRC의 값을 2로 하고 Firstupdate_LSN 값을 이전에 갱신을 실시한 트랜잭션의 로그 레코드의 LSN인 100으로 등록을 한다.

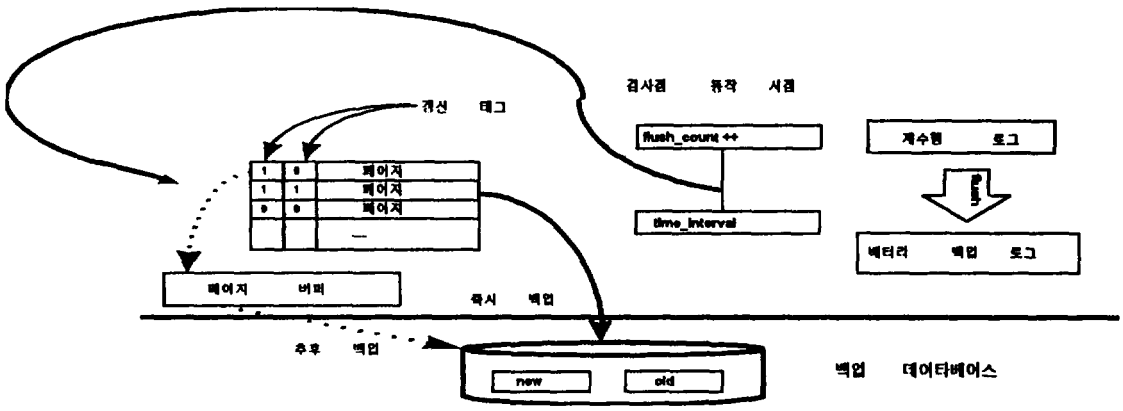
3.2.2 검사점 실행 및 백업

본 논문에서 제안한 검사점 실행은 기존의 퍼지(fuzzy) 검사점의 문제점을 개선하여 실행하였다. 개선된 퍼지 검사점은 기존의 검사점 기법인 트랜잭션 일치와 액션 일치 검사점 기법이 검사점 실행을 위해 동기화의 비용이 요구되고, 트랜잭션의 처리가 일시 중단되는 등의 문제점이 발생되고 있으나, 퍼지 검사점 기법은 일시 중단 문제나 동기화에 따른 비용 문제를 해결하였다. 그러나 퍼지 검사점 기법은 위 두 기법의 문제점은 해결하였으나 트랜잭션 처리의 원자성을 보장하지 못해 검사점 실행 이후 데이타베이스의 일치성을 유지하지 못한다는 단점이 발생한다.

본 연구에서는 이러한 단점을 해결하기 위해서 검사점 실행이 실시될 때 처리가 완료되지 않은 트랜잭션은 갱신 태그를 이용하여 갱신이 완료될 때 까지 디스크에 기록하지 않고 페이지 버퍼에 일시 저장했다가 트랜잭션 처리가 완료될 때 디스크 데이타베이스에 기록하므로 써 데이타베이스의 일치성을 보장하도록 하였다. 검사점 신호는 재수행 로그가 배터리 백업 로그에 기록된 횟 수에 의해서 발생되도록 하였다. 재수행 로그가 배터리 백업 로그에 기록된 횟 수가 많을수록 주기억 데이타베이스에 많은 갱신이 일어났다는 것을 의미하고, 이때 디스크 데이타베이스에 백업이 수행되도록 한다. 또한 데이타베이스를 페이지로 나누어 각 페이지 대해 변경 유무를 조사하여 변경된 페이지에 대해서만 디스크에 기록한다. 따라서, 검사점 실행시 모든 데이타베이스에 대한 접근이 중지되는 것이 아니고, 디스크에 기록할 필요가 없는 페이지에 대해서는 트랜잭션 처리가 가능하다[14, 15].

다음은 검사점 실행 알고리즘이다.

1) 검사점 실행 전에 조건을 검사한다.
2) 1)의 조건을 만족하지 않으면 트랜잭션 처리를 계속하고 1)의 조건이 만족되면 검사점 실행을 실시한다.
3) 검사점 실행 신호를 보내고 로그에 기록한다.
4) 디스크에 기록되지 않은 연산 중 가장 오랜된 것의 위치를 검사점 식별자에 기록한다.
5) 데이타베이스의 일치성을 보장하기 위해
   5.1) 갱신 태그의 상태가 11이면 페이지에 대한 갱신이 완료된 것으로 간주하여 디스크에 백업을 실시한다.
   5.2) 갱신 태그의 상태가 10이면 페이지에 대한 갱신이 진행중이므로 페이지 버퍼에 임시로 저장했다가 태그의 상태가 11이 되면 디스크에 기록한다.
   5.3) 갱신 태그의 상태가 01이면 페이지에 대해 트랜잭션 처리 요청이 들어오는 상태이므로 다음 검사점 실행시에 확인한다.



(그림 3) 검사점실행
(Fig. 3) Checkpointing

5.4) 갱신 태그의 상태가 00이면 어느 페이지에도 요청이 발생하지 않았으므로 아무런 액션도 취하지 않는다.

6) 백업을 실시할 페이지를 디스크의 한 위치에 할당한다.

7) 가장 최근에 갱신된 것을 표시하기 위해 검사점 실행 식별자를 저장한다.

9) 검사점 실행이 끝났음을 로그에 기록한다.

검사점 실행 측면에서는 서버와 마찬가지로 클라이언트도 검사점 실행과 유사하게 자신의 상태 정보를 파악하도록 제안하였다. 클라이언트는 자신의 상태 정보를 파악하여 서버에게 주기적으로 자신의 정보를 전송하도록 하여 클라이언트가 파손될 경우에 서버가 클라이언트에 대한 정보를 파악하고 있어 빠른 회복을 할 수 있도록 하였다. 또한, 클라이언트는 상태 정보 파악시 메모리 캐쉬의 상태 즉, 클라이언트가 갱신한 데이타 페이지 정보와 현재 활동 중인 트랜잭션의 상태를 기록하여 그에 대한 정보를 서버에게 전송한다. 서버는 모든 클라이언트에 대한 가장 최신의 정보를 유지하고 있으며, 자신이 파손될 경우에 대비해 주기적으로 검사점을 실행하는데 상태 정보 테이블에 대한 정보를 기록한다.

3.2.3 클라이언트나 서버 파손시의 동작

클라이언트/서버 환경에서는 크게 클라이언트의 파손, 서버의 파손, 그리고 서버와 클라이언트가 동시에 파손이 되는 경우가 있는데 클라이언트와 서버의 파손에 대해서만 회복 알고리즘을 제안하였다.

서버의 배터리 백업 로그에는 클라이언트가 갱신 연산을 실시한 결과를 기록하여 전송한 로그 레코드가 기록되어 있으며, 디스크 데이타베이스에는 검사점 실행시 기록한 최근의 데이타베이스 내용을 유지하고 있다. 또한 서버가 유지 관리하는 상태 정보 테이블은 시스템에 고장이 발생할 경우, 회복 동작을 실시할 트랜잭션의 상태를 파악하는데 주로 사용된다. 회복시에는 로그에 기록된 가장 최근의 검사점 기록 로그 레코드에 기록되어 있는 상태 정보 테이블의 내용을 이용하여 검사점 로그 레코드부터 로그 레코드를 분석한다. 서버는 자신의 파손으로부터 스스로 복구될 수 있어야 하기 때문에 주기적으로 검사점을 실시한 뒤 상태 정보 테이블에 자신의 정보와 서버에서 실행 중인 트랜잭션 상태 정보를 기록한다. 자신의 파손으로부터 회복되는 동안 서버는 상태 정보 테이블에 있는 갱신된 데이타 페이지에 대해서도 회복을 실시한다. 또한, 서버에서 실행중인 임의의 트랜잭션에 대해서는 철회수행을 실시한다.

클라이언트에 의해 갱신된 데이타 페이지는 서버의 검사점 실행이 마지막으로 실행되기 전에는 일반적으로 회복이 제대로 이루어지지 않는다. 그리고 클라이언트가 갱신한 데이타 페이지는 서버에게 전송되고, 서버는 검사점이 실행되고 난 후에 클라이언트에 의해 갱신된 데이타 페이지에 대한 권한을 부여받기 때문이다. 이러한 문제점을 해결 하기 위해서 본 논문은 다음과 같은 동작을 추가, 제안하였다. 클라이언트에서 파손이 일어날 때의 회복 동작은 본 연구에서 클라이언트가 자체적으로 상태 정보 파악을 수행하게 하므로 써 좀 더 효율적으로 회복이 이루어지도록 하였다. 클라이언트에서 상태 정보 파악을 실시하는 경우는 다음과 같다.

각 클라이언트는 주기적으로 자신의 상태를 파악한다. 클라이언트의 상태 파악 레코드에는 클라이언트 메모리 캐쉬의 상태 즉, 서버에는 갱신 결과가 알려지지 않은 갱신된 데이타 페이지의 LSN을 포함한 정보가 들어 있다. 그리고 클라이언트에서 현재 실행중인 트랜잭션의 상태도 포함하고 있다. 만약 클라이언트가 파손되면 서버는 클라이언트의 파손을 인지한 뒤 파손된 클라이언트에 대하여 복구를 시작한다. 이때 서버는 클라이언트가 전송한 상태 정보 기록을 토대로 하여 로그 분석 및 재 수행을 실시하여 복구를 한다. 이 단계에서는 파손된 클라이언트가 기록한 로그 레코드도 함께 처리된다. 그리고 이들 데이타 페이지에 대한 최종 상태가 이미 서버의 상태 정보 테이블에 존재하면, 재수행을 실시하지 않는다. 만약 로그 레코드의 LastLSN의 값이 데이타 페이지의 현재 LastLSN 보다 크다면 재수행 실시가 필요하다. 만약 서버가 파손되어 복구가 이루어지는 동안 어떤 클라이언트가 파손되면 서버는 복구후 클라이언트를 정상 처리를 시작하기 전의 상태로 복구시킨다.

4. 결 론

이 논문에서 제안된 환경의 가장 큰 특징은 ESM-CS, ARIES/CSA와는 달리 서버의 구성이 주기억 데이타베이스 시스템으로 구성되고, 로그의 구성도 배터리 백업 로그를 이용하여 서버의 오버헤드 감소 및 디스크 입출력을 최소화 하였다. 그리고 클라이언트와 클라이언트 간의 직접적인 데이타 페이지 전송을 허용하지 않는다.

따라서, 이 논문은 ESM-CS, ARIES/CSA등 기존 기법에 비해 다음과 같은 장점을 갖는다.

1) 클라이언트가 자신의 상태를 주기적으로 파악하여 가장 최근의 상태 정보를 서버에 전송하므로써, 해당 클라이언트의 파손시 빠른 회복이 이루어진다.
2) 갱신된 데이타 페이지에 대한 로그 레코드가 서버에 도착할 때 데이타가 갱신된 것으로 간주한다.
3) Client_LSN을 제안 이용하므로 써 서버가 어느 클라이언트에서 갱신이 발생한 것인가를 쉽게 인식하도록 하였다.
4) 검사점 기법은 퍼지 검사점 기법을 개선하여 검사점 실행에 따른 비용을 감소시키고, 데이타 베이스의 일치성을 보장하였으며, 검사점 실행시 트랜잭션의 처리가 중단되지 않도록 하여 트랜잭션 처리율을 높였다.
5) 트랜잭션의 처리에 있어서도 갱신 동작에 필요한 로그의 구성 및 상태 정보 테이블을 제안하여 예와 함께 제시하므로써 처리 과정에 필요한 로그 관리 및 데이타 페이지에 대한 관리가 쉽도록 하였다.

앞으로의 연구는 본 논문의 결과를 중심으로 제시된 문제점을 해결하고, 제시한 기법에서 발생될 수 있는 문제점이나 오버헤드를 최소화 하기위한 계속적인 연구가 필요하다. 검사점 실행에 있어 최적화된 로그 검사점 시기를 설정하는 것도 주요 과제이다. 또한 클라이언트/서버 DBMS 환경에서 본 연구에서 제안한 기법의 정당성 증명 및 이를 적용할 경우 발생되는 문제점을 해결 할 수 있는 연구가 필요하다. 기존의 ESM-CS와 ARIES/CSA 기법의 환경인 디스크 데이타베이스 시스템과의 다른 주기억 데이타베이스 시스템을 적용 할 경우의 효율성을 증명하는 연구와 이를 실제 주기억 장치 시스템에서 설계 및 구현하는 것도 주요한 과제이다.

참 고 문 헌

[1] Bernstein, P. A., V. Hadzilacos, and N. Goodman, "Concurrent Control and Recovery in Database Systems," Addison-Wesley, 1987.

[2] M. Carey, M. J. Franklin, M. Livny, E. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architectures," Proc., ACM SIGMOD., pp. 357-366, 1991.

[3] A. Delis and N. Roussopouls, "Modern Client-Server DBMS Architectures," Proc. ACM SIGMOD RECORD, 20, 3, Sep, 1991.

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

[5] M. H. Eich, "Main Memory Database Research Directions", Tech. Rep., 88-CSE-35, Comp. Sci. Dep., Southern Methodist University, 1988.

[6] M. H. Eich, "Foreword Main Memory Databases :Current and Future Research Issues," IEEE TKDE, vol. 4, no. 6, pp. 507-508, 1992.

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

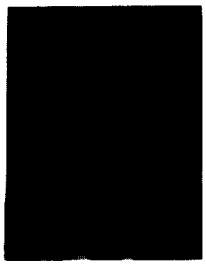
[8] J. Gray, & Reuter, A., "Transaction Processing: Concepts and techniques," Morgan Kaufmann, San Maeteo, 1993.

[9] T. Haerder & A. Reuter, "Principles of Transaction-Oriented Database Recovery," ACM Computing Surveys 15, 4, December 1983.

[10] C. Mohan & Don Haderle, et al, "ARIES:A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," ACM TODS, Vol. 17, No. 1, march 1992.

[11] C. Mohan and I. Narang, "ARIES/CSA:A Method For Database Recovery in Client-Server Archite-

ctures," Proc., ACM SIGMOD., pp. 55-66, 1994.
[12] Kenneth Salem & Hector Garcia-Molina, "Main Memory Database Systems: An Overview," IEEE TKDE, vol. 4, no. 6, 1992.
[13] Sang H. Son, "Real-Time Database Systems: Issues and Approaches," ACM SIGMOD RECORD, Vol. 17, No. 1, 1988.
[14] Eui In Choi, Kyung Chang Kim, Hae Chull Lim, "A Recovery Algorithm Using Main-Memory Databases in a Client/Server Environment," Proceedings of JTC-CSCC '94, pp. 868-872, 1994.
[15] E. I, Choi, H. C, Lim, "Recovery Technique Based on Fuzzy Checkpoint in a Client/Server DataBase System," IEEE Compsac'96, pp. 542-547, 1996.



고 병 오

1986년 충남대학교 계산통계학과 졸업(학사)
1989년 홍익대학교 전자계산학과 졸업(이학석사)
1996년 홍익대학교 전자계산학과 졸업(이학박사)
1993년 3월~1994년 2월 영동전문대학 전자계산과 전임강사
1994년 3월~1997년 8월 세명대학교 정보처리학과 조교수
1997년 9월~현재 공주 교육대학 컴퓨터 교육학과 전임강사
관심분야: 실시간데이타베이스, 주기억데이타베이스 클라이언트/서버 데이터베이스

최 의 인

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 조교수
관심분야: 실시간데이타베이스, 주기억데이타베이스, 클라이언트/서버 데이타베이스