데이타베이스 공유 시스템에서 버전 캐싱을 이용한 단일 노드 고장 회복 기법

(A Recovery Scheme of Single Node Failure using Version Caching in Database Sharing Systems)

조 행 래 † 정 용 석 †† 이 상 호 †††
(Haengrae Cho) (Yongseok Jeong) (Sangho Lee)

요 약 데이타베이스 공유 시스템(DSS)은 고성능 트랜잭션 처리를 위하여 여러 개의 처리 노드를 연결한 구조로서, 각 노드는 데이타베이스를 저장한 디스크를 공유한다. DSS를 구성하는 노드들이 고장날 경우 데이타베이스를 정확한 상태로 복구하기 위한 회복 과정이 필요한데 DSS에서 회복 작업은 하나의 노드로 구성된 일반적인 데이타베이스 시스템보다 많은 시간이 소요된다. 그 이유는 데이타베이스를 회복하기 위해 여러 노드에 나누어 저장된 로그들을 병합하여야 하며, 병합된 로그들을 이용하여 REDO 작업을 수행하여야 하기 때문이다. 본 논문에서는 Oracle 9i Real Application Cluster (ORAC)에서 제안된 캐쉬 연합 알고리즘의 성능을 개선한 2VC(Two Version Caching) 알고리즘을 제안한다. 2VC는 단일 노드 고장에 대한 회복 작업에서 로그 병합 과정을 생략할 수 있으므로 빠른 데이타베이스 회복을 지원할 수 있다는 장점을 갖는다. 뿐만 아니라, ORAC에서 발생하는 불필요한 디스크 기록 오버헤드를 줄임으로써 정상적인 트랜잭션 처리의 성능을 향상시킬 수 있다.

키워드 : 트랜잭션 처리, 데이타베이스 공유, 캐쉬 일관성, 데이타베이스 회복

Abstract A database sharing system (DSS) couples a number of computing nodes for high performance transaction processing, and each node in DSS shares database at the disk level. In case of node failures in DSS, database recovery algorithms are required to recover the database in a consistent state. A database recovery process in DSS takes rather longer time compared with single database systems, since it should include merging of discrete log records in several nodes and perform REDO tasks using the merged log records. In this paper, we propose a *two version caching* (2VC) algorithm that improves the cache fusion algorithm introduced in Oracle 9i Real Application Cluster (ORAC). The 2VC algorithm can achieve faster database recovery by eliminating the use of merged log records in case of single node failure. Furthermore, it can improve the performance of normal transaction processing by reducing the amount of unnecessary disk force overhead that occurs in ORAC.

Key words : transaction processing, database sharing, cache consistency, database recovery

1. 서 론

데이타베이스 공유 시스템(Database Sharing System: DSS)은 고성능 온라인 트랜잭션 처리가 필요한 응용 분야를 효율적으로 지원하기 위하여 제안된 시스템으로 디스크 계층에서 전체 데이타베이스를 공유하고, 별도의 메모리와 운영체제를 가진다[1,2]. DSS의 장점은 트랜잭션 실행시 동적인 부하 분산이 가능하고, 새로운 노드의 추가가 용이하며, 기존 노드의 고장이 전체 시스템에 큰 영향을 주지 않는다는 장점을 갖는다[3]. NAS나 SAN과 같은 네트워크 저장 장치의 발전에 따라 다양한 DSS 제품들이 출시되고 있으며, 대표적인 제품으로는 IBM DB2 데이타 공유 버전[4,5]과 Oracle 9*i* Real Application Cluster(ORAC)[6] 등을 들 수 있다.

DSS를 구성하는 각각의 노드는 자신의 버퍼에 최근

† 종신회원 : 영남대학교 전자정보공학부 교수
hrcho@yu.ac.kr
†† 비 회 원 : (주)퓨전소프트 연구원
yongs@yumail.ac.kr
††† 비 회 원 : 영남대학교 컴퓨터공학과
comman35@yumail.ac.kr
논문접수 : 2003년 6월 19일
심사완료 : 2004년 4월 9일

에 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[3,7-10].

DSS를 구성하는 노드들이 고장이 날 경우, 데이타베이스를 정확한 상태로 복구하기 위한 회복 기법이 필요하다[10-13]. DSS에서 제안된 가장 대표적인 데이타베이스 회복 기법으로 ARIES/SD[10]를 들 수 있다. ARIES/SD는 ARIES[14]를 DSS 환경으로 확장한 기법으로, 레코드와 같은 미세 단위 로킹을 지원하며 노드들 간에 갱신된 페이지를 네트워크를 통해 직접 전송함으로써 정상적인 트랜잭션 처리의 성능을 향상시킬 수 있다는 장점을 갖는다. 단일 노드나 여러 노드들의 고장이 발생할 경우, ARIES/SD는 회복을 위하여 여러 노드에 나누어 저장된 로그들을 통합한 전역 로그를 생성한 후, 전역 로그를 순차적으로 검색함으로써 REDO와 UNDO 과정을 거치게 된다. 이 과정 중에서 로그 병합과 REDO 과정은 로그 디스크와 데이타 디스크에 대한 빈번한 I/O로 인해 많은 시간이 소요된다[11,12].

ARIES/SD의 로킹 방식이나 페이지 전송 방식을 변경함으로써 데이타베이스 회복 시간을 줄일 수 있는 몇 가지 알고리즘들이 제안되었다. [15]에서는 동시성 제어 기법으로 페이지 단위 로킹을 사용할 경우 로그 병합이 필요없는 데이타베이스 회복 기법을 제안하였다. 그러나 페이지 단위 로킹은 낮은 동시성으로 인하여 레코드 단위 로킹에 비해 정상적인 트랜잭션 처리의 성능이 저하된다는 문제점을 갖는다. ORAC의 이전 버전인 Oracle Parallel Server[16]에서는 공유 디스크를 이용하여 노드들 간에 페이지를 전송함으로써 데이타베이스 회복 알고리즘의 성능을 개선하였다. 그러나 이 방법은 빈번한 공유 디스크 액세스로 인하여 정상적인 트랜잭션 처리의 성능이 저하된다는 단점을 가지며, ORAC의 경우에도 빠른 페이지 전송을 위하여 노드들 간에 갱신된 페이지를 네트워크를 통해 직접 전송하도록 변경되었다[9].

ARIES/SD와 동일한 로킹 방식과 페이지 전송 방식을 채택하면서 데이타베이스 회복 시간을 줄일 수 있는 알고리즘들도 제안되었다. [12]에서는 빈번히 갱신되는 페이지를 LRU 정책과 관계없이 강제로 디스크에 기록한다. 그 결과 고장이 발생할 경우 그 페이지에 적용할 로그의 수가 줄어들게 되고, 전체적인 REDO 시간이 빨라진다는 장점을 갖는다. 유사한 개념으로 Oracle에서는 체크포인팅을 빈번히 수행하여 고장시 회복할 페이지의 수와 로그의 수를 줄임으로써 빠른 회복을 지원할 수 있도록 하고 있다[17]. 그러나 두 방법 모두 빈번한 디스크 기록으로 인하여 정상적인 트랜잭션 처리의 성능이 저하될 수 있다는 문제점을 갖는다. 이와는 달리 [11]에서는 정상적인 트랜잭션 처리에 영향을 주지 않고 데이타베이스 회복 시간을 줄일 수 있는 알고리즘을 제안하였다. 기본 개념은 고장이 발생할 경우, 회복될 페이지의 이전 버전을 다른 노드가 캐싱하고 있다면 이를 이용하여 REDO 작업을 수행함으로써 REDO 시간을 줄인다는 것이다. 그러나 이 알고리즘은 ARIES/SD나 [12]와 같이 데이타베이스 회복을 위하여 로그 병합이 필요하다는 단점을 갖는다.

본 논문에서는 ARIES/SD와 동일한 방식의 DSS에서 빠른 데이타베이스 회복을 지원할 수 있는 2VC(Two Version Caching) 알고리즘을 제안한다. 2VC의 기본 개념은 다음과 같다. 먼저 페이지 P에 대해 디스크에 저장된 버전을 P^0라 가정하고, $P^0 \rightarrow P^1 \rightarrow ... \rightarrow P^{n-1} \rightarrow P^n$의 순서로 여러 노드에서 차례대로 갱신된다고 가정하자. P^n은 P의 최신 버전이며, P^n을 생성하여 캐싱한 노드를 P의 "소유자 노드"라 정의한다. P의 소유자 노드가 고장날 경우, ARIES/SD에서는 P^0를 디스크에서 액세스한 후 여러 노드의 로그들을 통합하여 차례대로 P^0에 적용함으로써 P^n을 생성한다. 이와는 달리 2VC에서는 P^n이나 P^{n-1}을 캐싱한 노드가 항상 존재하도록 유지하며 이러한 노드를 P의 "백업 노드"라 정의한다. 따라서 P의 소유자 노드가 고장날 경우, 백업 노드에 캐싱된 P의 버전에 대해 소유자 노드의 로그만 선택적으로 적용함으로써 P^n을 생성할 수 있다. 2VC의 장점은 다음과 같이 정리할 수 있다.

- 2VC는 단일 노드의 고장시 REDO 작업을 위하여 고장난 노드의 로그만 액세스하면 되므로 빠른 회복이 가능하다는 장점을 갖는다. 물론 소유자 노드와 백업 노드가 동시에 고장나는 경우에는 ARIES/SD와 동일한 방식으로 로그 병합을 수행해야 한다. 그러나 여러 개의 노드가 동시에 고장날 확률은 단일 노드가 고장날 확률보다 훨씬 작으므로, 단일 노드의 고장에 대한 빠른 회복은 DSS의 성능과 신뢰성을 향상시키기 위하여 우선적으로 고려되어야 한다[4,6,10].
- 뿐만 아니라, 2VC는 백업 노드를 유지하는 부담을 최소화하여 정상적인 트랜잭션 처리의 성능을 향상시킬 수 있다는 것이다. 본 논문에서는 모의실험을 통하여 2VC와 기존 기법들에 대해 정상적인 트랜잭션 처리의 성능을 비교하도록 한다.

본 논문의 구성은 다음과 같다. 2절에서는 기존에 제안된 회복 기법에 대해 살펴보고, 3절에서는 본 논문에

서 제안한 2VC의 알고리즘을 설명한다. 4절에서는 2VC의 성능을 분석하기 위하여 개발된 실험 모형을 설명하고 5절에서 실험 결과를 분석한다. 마지막으로 6절에서 결론을 맺는다.

2. 관련 연구

본 절에서는 ARIES/SD의 기본 개념을 살펴보고, ARIES/SD의 단점인 로그 병합 과정과 장기간의 데이타베이스 회복 시간을 해결할 수 있는 두 가지 방법을 설명하도록 한다.

2.1 ARIES/SD

ARIES/SD는 캐쉬 페이지에 대한 물리적 일관성을 유지하기 위하여 페이지의 소유자 노드를 나타내는 물리적 로크 개념을 이용한다[10]. 소유자가 아닌 노드 N_i가 페이지 P를 갱신하기 위해서는 먼저 P의 소유자가 되어야 하며 이를 위해 P에 대한 갱신 모드의 물리적 로크를 획득하여야 한다. 기존의 소유자 노드는 P에 대한 갱신 연산을 실행한 후 소유권을 N_i에게 넘겨주며, 이때 P의 최신 내용도 같이 전송된다. 물리적 로크를 이용함으로써 한 순간에 하나의 노드만이 소유자 노드가 될 수 있고, 여러 개의 갱신 연산들이 P에 동시에 실행될 수 없다. ARIES/SD에서는 페이지가 갱신될 때마다 로그가 생성되며, 각 로그에 대해 일련 번호(Log Sequence Number: LSN)가 할당된다. 뿐만 아니라, 데이타베이스의 각 페이지마다 그 페이지를 최종적으로 갱신한 연산에 대한 로그의 LSN을 저장하는 PageLSN 필드가 유지된다. ARIES/SD에서 특정 페이지에 대한 로그는 그 페이지가 디스크에 기록될 때, 혹은 로그를 생성한 트랜잭션이 완료할 경우나 페이지에 대한 소유자 노드가 변경될 때 로그 디스크에 기록된다.[1)]

예를 들면 그림 1에서 페이지 P는 3개의 레코드 R_1, R_2, R_3로 구성되며, N_1에 의해 R_1이 R_1'로 갱신된 후 N_2에 의해 R_2가 R_2'로 갱신된다. 그리고 마지막으로 N_3에서 R_3가 R_3'로 갱신된다. 그 결과로 N_3가 페이지 P의 소유자 노드가 되며, P의 최신 버전이 N_3에 캐싱되어 있다. 이때 노드들 간에 P가 직접 전송되므로 공유 디스크에 저장된 P에는 어떠한 갱신 사항도 반영되지 않는다.

ARIES/SD의 회복 과정은 단일 노드의 고장과 로킹 및 캐싱 정보를 관리하는 전역 로크 관리자(Global Lock Manager: GLM)를 포함한 여러 노드들이 고장난 경우로 나누어 설명된다. 단일 노드 고장의 경우, GLM으로부터 회복해야 될 페이지 정보와 REDO 단계의 시작 LSN(RedoLSN)을 전송받아 RedoLSN부터 통합 로그를 스캔하면서 회복해야 될 페이지를 디스크에서 판독하여 로그를 적용한다. 그러나 GLM이 고장난 경우에는 회복에 필요한 정보도 같이 분실되므로, 페이지가 처음으로 갱신된 시점의 LSN(RLSN)들을 주기적으로 체크포인팅하여 체크포인트 로그의 형태로 디스크에 저장한다. 회복 과정은 가장 최근에 저장된 체크포인트 로그를 판독하여 저장된 RLSN 중에서 가장 작은 RLSN부터 통합 로그를 스캔하면서 갱신된 페이지를 디스크에서 판독한 후 로그를 적용하는 작업으로 이루어진다.

예를 들면 그림 1에서 노드 N_3가 고장날 경우, 디스크에 저장된 P를 액세스한 후 여기에 $L_1 \rightarrow L_2 \rightarrow L_3$의 순서로 로그를 적용시키는 과정이 필요하다. 따라서 ARIES/SD의 경우 회복을 위하여 로그들을 LSN 순서로 정렬하는 과정이 필요하며, REDO 단계에서 방대한 양의 통합 로그 스캔을 필요로 하므로 REDO 시간이 매우 길어질 수 있다는 단점을 갖는다.

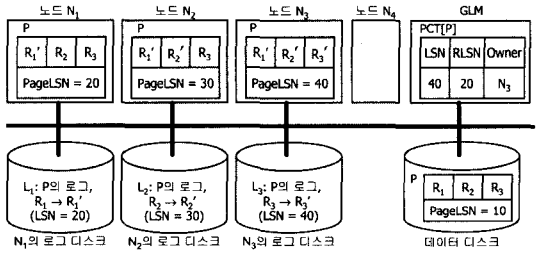

그림 1 네 개의 노드로 구성된 DSS

2.2 IBM DB2 7.0 데이타베이스 공유 버전

ARIES/SD의 문제점을 해결하기 위하여 IBM DB2 데이타베이스 공유 버전에서는 모든 노드들 간에 그룹 버퍼 풀(Group Buffer Pool : GBP)이라는 저장 공간을 공유하도록 하였다[4,8]. 각 노드에서 실행되는 트랜잭션들은 완료될 때마다 'force-at-commit' 정책에 의해 갱신한 페이지들을 GBP에 저장한다. 이때 노드와 GBP는 고속의 통신 채널로 연결되어 GBP와의 페이지 전송에 소요되는 시간을 최소화하도록 한다. 그리고 GBP에 캐싱된 데이타들은 주기적으로 디스크에 기록된다.

IBM DB2 데이타베이스 공유 버전에서는 단일 노드 고장의 경우 로그 병합 과정과 REDO 작업이 필요 없다는 장점을 갖는다. 즉, 고장난 노드에서 기존에 완료된 트랜잭션의 실행 결과들은 이미 GBP나 디스크에 반영되어 있으므로 이들을 다시 실행할 필요가 없다. 단, 완료하지 않은 트랜잭션의 실행 결과가 GBP에 반영되

1) 소유자 노드가 변경될 경우, 이전 소유자 노드에서 생성한 로그를 디스크에 기록하지 않는 방법도 [10]과 [13]에서 제안되었다. 그러나 로그를 디스크에 기록하지 않을 경우, 로그 우선 기록(Write Ahead Logging: WAL) 개념을 지원하기 위한 추가적인 오버헤드가 존재하며 회복 과정이 복잡해진다는 단점을 갖는다[10].

어 있을 경우 UNDO 작업이 필요하지만, 이는 고장난 노드의 로그만 스캔하여 수행할 수 있다. IBM DB2 데이타베이스 공유 버전의 문제점은 노드와 GBP간에 고속의 통신 채널이 지원되어야 한다는 것으로, 이를 위해 IBM S/390 Parallel Sysplex와 같은 특수한 하드웨어에서만 동작이 가능하다는 단점이 있다.

2.3 Oracle 9i Real Application Clusters(ORAC)

ORAC에서 구현한 캐쉬 연합 정책은 Oracle Parallel Server의 캐싱 정책을 개선하여 노드 간에 네트워크를 통한 직접 페이지 전송을 지원한다[9]. 캐쉬 연합 정책의 특징은 Oracle의 전통적인 다중 버전 동시성 제어 알고리즘[18]을 DSS 환경으로 확장하였다는 것이다. 예를 들면, 그림 1의 노드 N_4에서 실행 중인 트랜잭션 T_4의 타임스탬프[2]가 10이며 페이지 P를 판독하고자 할 경우를 가정하자. 이때 N_4는 GLM을 통하여 N_3에게 P에 대한 판독 요청 메시지를 전송한다.[3] N_3는 판독 요청 메시지에 포함된 T_4의 타임스탬프를 참조한 후, T_4의 타임스탬프 이전에 완료된 트랜잭션의 실행 결과만 포함된 P의 새로운 버전 P'를 생성하여 N_4에게 전송한다. 이때 P'를 P의 클론(clone)이라고 정의하는데, P'는 P에서 LSN이 10이상인 로그들의 갱신 내용을 철회한 버전이다.

노드 N_4의 트랜잭션 T_4가 갱신을 위하여 노드 N_3에게 페이지 P를 요구할 경우, N_3는 P'를 생성하지 않고 P의 최신 버전을 N_4에게 전송하며 P의 소유자 노드도 N_4로 변경된다. ARIES/SD와의 차이점은 N_3가 P를 전송한 후, P를 자신의 버퍼에서 삭제하는 것이 아니라 클론으로 계속 캐싱한다는 것이다. 이 방법의 장점은 N_3의 다른 트랜잭션이 P를 판독하고자 할 경우 캐싱된 클론을 이용하여 이를 지원할 수 있다는 것이다. 그 결과, 노드 간에 판독 연산을 위한 페이지 전송 오버헤드를 줄일 수 있다. 그림 1의 경우에서는 N_1과 N_2가 P의 클론들을 캐싱하고 있다.

ORAC의 캐쉬 연합 정책은 단일 노드의 고장시 로그 병합이 필요없다는 장점을 갖는다. 예를 들면, 그림 1에서 P의 소유자 노드 N_3가 고장날 경우, N_2에서 캐싱하고 있는 P의 클론은 N_3에서 갱신한 내용을 제외한 P의 모든 내용을 모두 포함하고 있으므로, 이를 이용하여 N_3의 P에 대한 로그만 적용함으로써 P의 최신 버전을 복구할 수 있기 때문이다.

다중 버전 동시성 제어 기법을 이용한 ORAC의 캐쉬 연합 정책은 다음과 같은 문제점을 갖는다. 첫째, 페이지 P의 이전 소유자 노드들은 P의 클론들을 "dirty" 비트가 해제되지 않은 상태에서 계속 캐싱한다[6]. 그 결과, 이전 소유자 노드에서 P가 교체될 경우 최신 버전이 아닌 페이지가 디스크에 기록될 수 있으므로 이를 해결하기 위한 알고리즘이 필요하다. 예를 들면 그림 1에서 N_1의 버퍼에서 P가 교체될 경우, ORAC은 N_3에게 이를 알리고 N_3는 캐싱하고 있는 P를 디스크에 강제로 기록한다. 이후 N_1과 N_2에 대해 무효화 메시지를 전송하여, P의 클론을 해당 노드의 버퍼에서 삭제한다. P가 여러 노드에서 자주 액세스될 경우 P의 클론을 캐싱하고 있는 노드의 수가 증가하고, 그 결과 P의 강제 기록 연산이 발생할 확률이 증가하여 빈번한 디스크 I/O가 발생할 수 있다는 단점을 갖는다.

캐쉬 연합 정책의 두 번째 문제점은 다중 버전 동시성 제어 기법에 따른 판독 연산 처리 과정에서 발생한다. 예를 들면 그림 1에서 노드 N_4의 트랜잭션 T_4가 P를 판독하고자 하며 T_4의 타임스탬프가 N_1에 캐싱된 P의 PageLSN보다도 작을 경우, N_3는 자신의 갱신 내용 외에 N_2와 N_1의 갱신 내용들을 모두 철회한 P의 클론을 생성한 후 N_4에게 전송하여야 한다. ORAC에서는 갱신되기 이전 데이타를 철회 세그먼트에 저장하고 있으므로, N_4는 N_1의 철회 세그먼트를 액세스하여 P의 클론을 생성한다. 이때 N_1의 철회 세그먼트가 디스크에 이미 기록되었다면 P의 클론을 액세스하기 위한 추가적인 디스크 I/O가 발생한다. 뿐만 아니라, N_4에서 T_4보다 나중에 실행된 트랜잭션도 P를 판독하고자 할 경우에는 최신 버전의 클론을 다시 전송받아야 하므로 동일한 페이지에 대한 여러 번의 페이지 전송이 발생할 수 있다.

이런 관점에서 본 논문에서는 IBM DB2 데이타베이스 공유 버전과는 달리 특수한 하드웨어를 필요로 하지 않으면서, ORAC의 캐쉬 연합 정책의 성능을 개선할 수 있는 캐쉬 일관성 기법과 회복 기법을 제안하도록 한다.

3. 빠른 회복을 위한 버전 관리

본 절에서는 단일 노드의 고장 시 빠른 회복을 지원할 수 있는 캐쉬 일관성 기법인 2VC(Two Version Caching) 알고리즘에 대해서 설명한다. 먼저 2VC의 기본 개념과 가정에 대해서 설명한 후, 2VC 알고리즘과 2VC에 기반을 둔 데이타베이스 회복 과정에 대해 자세히 살펴보도록 한다.

3.1 기본 개념과 가정

2VC의 로킹 및 노드 간 페이지 전송 방식은 ARIES/SD와 동일하다. 즉, 레코드와 같은 미세 단위의 로킹을 지원하며, 노드의 버퍼들 간에 네트워크를 통한 직접 페이지 전송을 지원한다. ARIES/SD에서 발생하

2) Oracle에서는 타임스탬프라는 용어 대신에 SCN(System Change Number)라는 용어를 사용하고 있으나, 본 논문에서는 보다 일반적인 용어인 타임스탬프를 사용하도록 한다.

3) ORAC에서는 GLM이라는 용어 대신에 GCS(Global Cache Service)라는 용어를 사용하고 있으나, ARIES/SD와의 비교를 위하여 ARIES/SD에서 사용한 GLM이라는 용어를 사용하도록 한다.

는 단일 노드 고장 시의 로그 병합과 장기간의 REDO 문제를 해결하기 위하여 2VC에서는 ORAC의 캐쉬 연합 정책을 다음과 같이 개선하도록 한다.

① 페이지 P에 대한 갱신 연산의 처리 과정에서 소유권이 N_i에서 N_k로 이전될 경우, N_i에서 갱신한 P의 버전 혹은 그 이후의 버전을 N_k를 제외한 하나 이상의 노드에서 항상 캐싱하도록 한다. 단, 이러한 조건은 현재 소유자에 대해서만 적용한다.

② 노드 N_1이 페이지 P를 판독하기 위하여 소유자 노드 N_2에게 P를 요청할 경우, N_2는 P의 최신 버전을 N_1에게 전송한다.

첫 번째 규칙은 P의 소유자 노드인 N_k가 고장날 경우, 직전 소유자 노드(N_i)가 갱신한 페이지나 그 이후의 페이지 버전이 다른 노드에 캐싱되어 있다면 P의 회복을 위하여 다른 노드의 로그가 필요하지 않다는 점을 이용하였다. 즉, 단일 노드 고장에 대한 빠른 회복을 위하여 최소한의 버전만 캐싱되도록 한다. 이때 주의할 것은 P의 소유자가 N_k에서 다른 노드로 다시 이전될 경우, 데이타베이스 회복의 관점에서는 N_i의 버전을 더 이상 캐싱할 필요가 없다는 점이다. 예를 들면 그림 1에서 N_3의 직전 소유자인 N_2에 대해서만 P의 이전 버전을 캐싱하도록 한다. 따라서 캐싱에 대한 조건은 항상 현재 소유자에 대해서만 적용한다. 이와는 달리 ORAC에서는 P를 갱신한 모든 노드들이 P의 버전들을 항상 캐싱하도록 함으로써 불필요한 디스크 I/O가 발생하였다.

두 번째 규칙은 ORAC의 판독 연산의 처리 과정에서 발생하는 오버헤드를 줄이기 위하여 제안되었다. 즉, 2VC는 ORAC과 같은 다중 버전 동시성 제어 기법을 지원하지 않으며, ARIES/SD나 DSS에서의 기존 캐쉬 일관성 기법[3,7,8,12,16]들이 채택한 전통적인 단일 버전 기반의 로킹을 지원한다. 따라서 2.3절에서 설명한 ORAC에서의 불필요한 트랜잭션 철회 과정이나 동일한 페이지를 여러 번 전송하는 문제를 해결할 수 있다. 단, 단일 버전 로킹의 경우 판독 연산의 실행 비용이 증가할 수 있으나, 로크 캐싱(lock caching)이나 레코드 소유자 등의 개념을 이용함으로써 판독 연산을 위한 로크나 페이지의 전송 비용을 줄일 수 있다는 것이 여러 문헌에서 제안된 바 있다[3,7,10]. 2VC는 로크 캐싱이나 레코드 소유자 등의 개념과는 독립적이므로 이러한 개념들을 2VC에 혼합하는 것이 가능하지만, 본 논문에서는 설명의 단순함을 위하여 로크 캐싱과 레코드 소유자 등에 대한 고려는 하지 않도록 한다.

ARIES/SD와 동일하게 2VC는 동시성 제어 및 캐쉬 일관성 유지를 담당하는 전역 로크 관리자(Global Lock Manager: GLM)가 존재한다고 가정한다. GLM은 동시성 제어를 위한 로킹 테이블을 유지하며, 캐쉬 일관성 유지를 위하여 페이지 캐쉬 테이블(Page Cache Table: PCT)을 유지한다. PCT는 각 노드에 캐싱되어 있는 최신 페이지들에 대한 일관성 정보를 저장하는데, 이러한 정보들은 각 노드가 GLM에게 레코드 로크를 요청하는 과정을 통하여 GLM에게 전송된다. 페이지 P_i에 대해 PCT[P_i] 항목은 다음과 같은 정보들로 구성된다.

- PageLSN - P_i를 최종적으로 갱신한 연산에 대한 로그의 LSN.
- RLSN - P_i를 처음으로 갱신한 로그의 LSN. P_i가 갱신된 후 디스크에 저장될 때 LSN으로 표현할 수 있는 가장 큰 값인 "Hi-Value"로 초기화.
- Owner - P_i에 대한 소유자 노드. 소유자 노드가 없을 경우 -1로 설정.
- Backup - Owner가 고장날 경우, P_i의 버전을 가져올 노드. 그런 노드가 없을 경우 -1로 설정.
- CopySet - P_i의 최신 버전을 캐싱하고 있는 노드들의 집합.

노드 N_i에서 실행되는 트랜잭션 T_i가 페이지 P_i에 저장된 레코드 R_i를 액세스하기 위해서는 로크 요청 메시지를 GLM에게 전송하여야 한다. 로크 요청 메시지의 구조는 다음과 같다.

- NodeID - N_i의 식별자.
- TranID - T_i의 식별자.
- RID 액세스하고자 하는 레코드 R_i의 식별자.
- PageID - 레코드를 포함하고 있는 페이지 P_i의 식별자. RID에 페이지의 식별자가 포함되어 있을 경우에는 생략 가능.
- OpCode - 액세스의 유형으로 '판독' 연산 이거나 '기록' 연산.
- PageLSN - 캐싱하고 있는 P_i의 PageLSN. N_i가 P_i를 캐싱하지 않을 경우에는 -1로 설정.
- EndLSN - OpCode가 '기록'일 경우 포함되며, N_i에서 현재까지 할당된 가장 큰 LSN.

본 논문에서는 ARIES/SD나 DSS에서의 기존 캐쉬 일관성 기법들과 같이 신뢰성있는 네트워크를 가정한다. 즉, 메시지 분실이나 네트워크 분할 등은 발생하지 않는다고 가정한다. 그리고 각각의 노드마다 그 노드에서 실행된 트랜잭션들의 로그를 저장하는 별도의 로그 디스크가 존재한다고 가정한다. 단, 특정 노드가 고장날 경우 그 노드의 회복을 위하여 다른 노드에서 고장난 노드의 로그 디스크를 액세스할 수 있다.

3.2 2VC 알고리즘

2VC 알고리즘은 '판독' 알고리즘과 '기록' 알고리즘, 그리고 '버퍼 교체' 알고리즘으로 나누어진다. 앞의 두 알고리즘은 로크 요청 메시지의 OpCode에 따라 로크 요청 메시지를 처리하는 알고리즘이며, 버퍼 교체 알고

리즘은 소유자 노드나 Backup 노드에서 페이지 교체가 발생할 경우 이를 처리하는 알고리즘이다.

3.2.1 판독 알고리즘

노드 N_i가 <N_i, T_i, R_i, P_i, '판독', PageLSN, EndLSN>의 쌍으로 구성된 로크 요청 메시지(M_{req})를 GLM에게 전송할 경우, GLM은 먼저 R_i에 대한 판독 로크를 획득할 수 있는 지를 조사한다. 판독 로크를 획득한 후 그림 2와 같은 판독 연산에 대한 캐쉬 일관성 알고리즘을 수행한다.

2VC의 판독 연산 알고리즘은 ARIES/SD와 거의 동일하다. 즉, P_i의 최신 버전을 캐싱하고 있는 노드가 없을 경우에는 디스크에서 P_i를 액세스한 후, N_i가 P_i의 최신 버전을 캐싱하고 있음을 PCT[P_i]에 등록한다(단계 1). N_i가 P_i의 최신 버전을 캐싱한 경우에는, 자신이 캐싱한 P_i의 버전을 즉시 액세스할 수 있다(단계 2). 만약 P_i의 최신 버전이 다른 노드에 캐싱된 경우에는 최신 버전을 캐싱하고 있는 노드로부터 최신 버전의 페이지를 전송받도록 한다(단계 2). 이때 최신 버전을 캐싱하고 있는 노드는 P_i의 소유자이거나, P_i에 대한 갱신 작업이 없었을 경우에는 P_i의 CopySet에 포함된 임의의 노드가 된다. 예를 들면 그림 1에서 노드 N_4가 판독을 위하여 P를 요청할 경우, GLM은 N_3를 통하여 P의 최신 버전을 N_4에게 전송하도록 하고 PCT[P].CopySet에 N_4를 추가한다.

여러 노드가 동시에 P_i에 대한 판독을 요청할 경우, 단계 1에 따라 P_i에 대한 디스크 I/O가 각 노드에서 동시에 발생할 수 있다. 이를 해결하기 위하여 GLM은 디스크 액세스 메시지를 N_i에게 보내기 전에 PCT[P_i]에 현재 디스크 I/O가 수행 중임을 나타내는 플래그를 설정한다. 이 플래그는 N_i로부터 M_{io}를 전송받은 후 해제하며, 플래그가 설정된 동안에 요청된 P_i의 판독 메시지들은 대기 큐에서 대기한다.

3.2.2 기록 알고리즘

노드 N_i가 <N_i, T_i, R_i, P_i, '기록', PageLSN, EndLSN>의 쌍으로 구성된 로크 요청 메시지(Mreq)를 GLM에 전송할 경우, GLM은 Ri에 대한 기록 로크가 허용된 후 그림 3과 같은 캐쉬 일관성 알고리즘을 수행한다.

P_i의 최신 버전을 캐싱하고 있는 노드가 없을 경우에는 디스크에서 P_i를 액세스한 후, N_i가 P_i의 최신 버전을 캐싱하며 P_i의 소유자임을 PCT[P_i]에 등록한다(단계 1). P_i를 최신 버전을 캐싱하고 있는 노드가 존재할 경우에는 P_i의 소유자 노드 존재 여부에 따라 다시 여러 가지 경우로 나누어진다. 먼저 P_i의 소유자 노드가 존재하지 않을 때에는 N_i가 P_i의 최신 버전을 캐싱하지 않을 경우 CopySet에 포함된 임의의 노드로부터 P_i의 최신 버전을 전송받은 후 P_i의 소유자 노드로 N_i를 등록한다(단계 2). 그리고 기록 연산의 결과 P_i의 내용이 변경되므로 P_i의 CopySet에 속하는 노드들에게 P_i의 무효화 메시지를 전송하고 CopySet을 공집합으로 변경한다. 무효화 메시지를 전송받은 노드들은 자신의 버퍼에 캐싱된 P_i에 대해 무효화 비트를 설정하여 더 이상 P_i를 액

1. **if** (PCT[P_i] == NULL) **// P_i의 최신 버전을 캐싱한 노드가 없음**
   - GLM은 N_i에게 로크 승인 메시지와 함께 디스크 액세스 메시지를 전송.
   - 디스크 액세스를 완료한 후, N_i는 GLM에게 <P_i, PageLSN>을 포함한 디스크 액세스 완료 메시지(M_{io})를 전송.
   - GLM은 M_{io}를 전송받은 다음 PCT[P_i]를 아래와 같이 초기화.
     - PCT[P_i].Owner = PCT[P_i].Backup = -1; PCT[P_i].CopySet = {N_i};
     - PCT[P_i].PageLSN = M_{io}.PageLSN; PCT[P_i].RLSN = Hi-Value;
2. **else if** (M_{req}.PageLSN == PCT[P_i].PageLSN) **// N_i는 P_i의 최신 버전을 캐싱**
   - GLM은 N_i에게 로크 승인 메시지를 전송.
   - 이후 N_i는 자신이 캐싱한 P_i 버전을 액세스하여 판독 연산을 처리.
3. **else // N_i는 P_i를 캐싱하지 않거나 이전 버전을 캐싱**
   - GLM은 N_i에게 로크 승인 메시지와 페이지 수신 메시지를 전송.
   - **if** (PCT[P_i].Owner ≠ -1) **// P_i의 소유자 노드가 존재**
     - GLM은 PCT[P_i].Owner에게 P_i를 N_i에게 전송하도록 요청.
   - **else // P_i의 소유자 노드가 없음**
     - GLM은 PCT[P_i].CopySet에 속하는 임의의 노드에게 P_i를 N_i에게 전송하도록 요청.
   - PCT[P_i].CopySet = PCT[P_i].CopySet ∪ {N_i}
   - N_i는 전송받은 P_i 버전을 캐싱한 후, 이를 이용하여 판독 연산을 처리.

그림 2 판독 연산에 대한 캐쉬 일관성 알고리즘

1. **if** (PCT[P_i] == NULL) **// P_i의 최신 버전을 캐싱한 노드가 없음**
   - GLM은 N_i에게 로크 승인 메시지와 함께 디스크 액세스 메시지를 전송.
   - 디스크 액세스를 완료한 후, N_i는 GLM에게 <P_i, PageLSN>을 포함한 디스크 액세스 완료 메시지(M_{io})를 전송.
   - GLM은 M_{io}를 전송받은 다음 PCT[P_i]를 아래와 같이 초기화.
     - PCT[P_i].Owner = N_i; PCT[P_i].Backup = -1; PCT[P_i].CopySet = {};
     - PCT[P_i].PageLSN = PCT[P_i].RLSN = maximum(M_{io}.PageLSN+1, M_{req}.EndLSN+1);
2. **else if** (PCT[P_i].Owner == -1) **// P_i의 소유자 노드가 없음**
   - PCT[P_i].Owner = N_i;
   - PCT[P_i].PageLSN = PCT[P_i].RLSN = maximum(PCT[P_i].PageLSN+1, M_{req}.EndLSN+1);
   - GLM은 N_i에게 로크 승인 메시지를 전송.
   - **if** (M_{req}.PageLSN ≠ PCT[P_i].PageLSN) **// N_i가 P_i의 최신 버전을 캐싱하지 않음**
     - GLM은 PCT[P_i].CopySet에 속하는 임의의 노드에게 P_i를 N_i에게 전송하도록 요청.
     - GLM은 N_i에게 페이지 수신 메시지를 전송.
   - PCT[P_i].CopySet에 속하는 노드들에게 P_i의 무효화 메시지 전송. PCT[P_i].CopySet = {}.
3. **else if** (PCT[P_i].Owner == N_i) **// N_i가 P_i의 소유자 노드**
   - GLM은 N_i에게 로크 승인 메시지를 전송.
   - PCT[P_i].CopySet에 속하는 노드들에게 P_i의 무효화 메시지 전송. PCT[P_i].CopySet = {};
   - PCT[P_i].PageLSN = maximum(PCT[P_i].PageLSN+1, M_{req}.EndLSN+1);
4. **else** **// P_i의 다른 소유자 노드가 존재**
   - **if** (M_{req}.PageLSN ≠ PCT[P_i].PageLSN) **// N_i가 P_i의 최신 버전을 캐싱하지 않음**
     - GLM은 PCT[P_i].Owner에게 P_i의 소유권 변경 및 P_i를 N_i에게 전송하도록 요청.
     - GLM은 N_i에게 로크 승인 메시지와 페이지 수신 메시지를 전송.
   - **else** **// N_i가 P_i의 최신 버전을 캐싱**
     - GLM은 PCT[P_i].Owner에게 P_i의 소유권이 변경되었음을 통보.
     - GLM은 N_i에게 로크 승인 메시지를 전송.
   - 소유권 변경을 위하여 PCT[P_i].Owner는 P_i에 대한 로그를 디스크에 저장한 후 버퍼에서 P_i의 dirty bit를 해제.
   - PCT[P_i].CopySet에 속하는 노드들에게 P_i의 무효화 메시지 전송. PCT[P_i].CopySet = {}.
   - 최종적으로 GLM은 PCT 정보를 다음과 같이 수정.
     - PCT[P_i].Backup = PCT[P_i].Owner; PCT[P_i].Owner = N_i;
     - PCT[P_i].PageLSN = maximum(PCT[P_i].PageLSN+1, M_{req}.EndLSN+1);

그림 3 기록 연산에 대한 캐쉬 일관성 알고리즘

세스하지 않도록 한다. 만약 N_i가 P_i의 현재 소유자라면 CopySet을 공집합으로 하고 P_i의 전송은 발생하지 않는다(단계 3). 기록 연산의 결과 P_i의 PageLSN도 변경되는데 GLM은 P_i의 새로운 PageLSN에 대한 예측값을 PCT에 등록한다. 예측값은 P_i의 현재 PageLSN에 1을 더한 값과 N_i에서 현재까지 할당된 가장 큰 LSN 값에 1을 더한 값 중에서 큰 값으로 설정된다.

P_i의 다른 소유자 노드가 존재할 경우에는 N_i가 P_i의 최신 버전을 캐싱하지 않을 경우 P_i의 소유자로부터 최신 버전을 우선 전송받은 후, P_i의 소유자를 Ni로 변경한다(단계 4). P_i의 기존 소유자는 P_i의 Backup 노드가 되는데 Backup 노드는 소유자 노드가 고장날 때 캐싱하고 있던 P_i의 직전 버전을 이용하여 소유자 노드의 회복을 용이하게 한다는 것이다. 기록 연산의 예로서 그림 1에서 노드 N_4가 기록을 위하여 P를 요청할 경우에는, GLM은 N_3를 통하여 P의 최신 버전을 N_4에게 전송한 후 PCT[P]의 Owner를 N_4로 Backup은 N_3로 설정한다.

CopySet은 항상 최신의 페이지 버전을 캐싱하지만 Backup은 이전 소유자의 버전을 캐싱할 수 있다는 점에서 차이가 난다. 뿐만 아니라, 동일한 소유자에서 여러 번의 기록 연산이 발생할 경우 CopySet은 항상 무효화되어야 하지만 Backup에 대해서는 무효화 과정이 발생하지 않는다. 즉, Backup은 현재 소유자가 고장날 경우 고장난 노드의 로그만 이용한 빠른 회복을 지원할 목적으로 사용된다.

3.2.3 버퍼 교체 알고리즘

```
1. 노드 Ni에서 Pi의 무효화 비트가 설정되어 있는 지 확인.
   • 설정되어 있을 경우, 즉시 버퍼에서 교체 가능.
   • 설정되어 있지 않을 경우, GLM에게 <Ni, Pi>의 쌍으로 구성된 버퍼 교체 메시지 전송.
2. GLM은 버퍼 교체 메시지를 전송받은 다음 단계 3, 4, 또는 5를 수행.
3. if (PCT[Pi].Owner == Ni)          // Ni가 Pi의 소유자 노드
   • Ni에게 Pi의 디스크 기록 메시지를 전송. 이후 Ni는 Pi를 디스크에 기록.
   • if (PCT[Pi].Backup ≠ -1)
       - PCT[Pi].Backup 노드에게 Pi의 무효화 메시지를 전송.
   • if (PCT[Pi].CopySet == {})          // Pi의 최신 버전을 캐싱한 노드가 없음
                PCT[Pi] = NULL;
   • else       PCT[Pi].Owner = PCT[Pi].Backup = -1;
4. else if (PCT[Pi].Backup == Ni)     // Ni가 Pi의 Backup 노드
   • if (PCT[Pi].CopySet ≠ {})          // CopySet에 하나 이상의 노드 포함
       - PCT[Pi].CopySet에 속하는 임의의 노드 Nk를 선택.
       - PCT[Pi].Backup = Nk; PCT[Pi].CopySet = PCT[Pi].CopySet - {Nk};
   • else       // CopySet에 노드가 없음
       - PCT[Pi].Owner 노드에게 Pi의 디스크 기록 메시지를 전송.
       - PCT[Pi].CopySet = {PCT[Pi].Owner}; PCT[Pi].Owner = PCT[Pi].Backup = -1;
5. else              // Ni가 Pi의 CopySet에 포함
   • PCT[Pi].CopySet = PCT[Pi].CopySet - {Ni};
   • if (PCT[Pi].CopySet == {} and PCT[Pi].Owner == -1)  // Pi를 캐싱한 노드가 없음
                PCT[Pi] = NULL;
```

그림 4 버퍼 교체 알고리즘

2VC 알고리즘은 노드 N_i에서 무효화 비트가 설정되지 않은 페이지 P_i가 버퍼에서 교체될 경우 그림 4와 같은 버퍼 교체 알고리즘을 수행한다.

N_i가 소유자 노드일 경우 GLM은 N_i에게 P_i의 기록을 요청한다(단계 3). 이때 P_i의 CopySet이 존재하지 않을 경우 P_i의 최신 버전을 캐싱한 노드가 더 이상 존재하지 않으므로 PCT에서 P_i 항목을 삭제한다.[4] N_i가 Backup 노드일 경우에는 P_i의 최신 버전을 캐싱하고 있는 다른 노드가 존재하는 지를 CopySet에서 검사한다(단계 4). 만약 CopySet에 노드 N_k가 존재한다면 N_k를 P_i의 Backup 노드로 변경하고 CopySet에서 N_k를 삭제한다. CopySet이 공집합일 경우에는 P_i의 소유자 노드에서 P_i를 기록한 후 소유자 노드를 CopySet에 포함시킨다. 마지막으로 N_i가 P_i의 CopySet에 포함될 경우에는 CopySet에서 P_i를 삭제한다(단계 5). 그 결과 CopySet이 공집합이면서 소유자 노드도 존재하지 않을 경우에는 P_i의 최신 버전을 캐싱한 노드가 존재하지 않으므로 PCT에서 P_i 항목을 삭제한다.

ORAC과 비교할 때 소유자 노드에 의한 P_i의 강제 기록 가능성은 매우 줄어드는데, 그 이유는 다음과 같다. 첫째, ORAC은 P_i를 갱신한 모든 이전 소유자 노드들이 P_i의 Backup 노드가 되며, 이중 하나의 노드라도 P_i를 버퍼에서 교체할 경우 소유자 노드에서 P_i의 강제 기록이 발생한다. 이와는 달리, 2VC에서는 직전 소유자 노드만 Backup 노드가 된다. 둘째, 2VC에서는 직전 소유자 노드에서 P_i를 교체하더라도 CopySet에 다른 노드가 존재한다면 Backup 노드를 변경함으로써 P_i의 강제 기록 가능성을 더욱 줄일 수 있다. 예를 들면 그림 1의 노드 N_1에서 P를 교체할 경우 ORAC은 N_3에게 P를 기록하도록 하지만 2VC에서는 N_1이 P의 Backup이 아니므로 이러한 문제점이 발생하지 않는다. 그리고 P의 Backup인 N_2에서 P의 교체가 발생할 경우에도 만약 N_4가 P의 CopySet에 포함되어 있다면 Backup을 N_4로 변경함으로써 P의 디스크 기록을 방지할 수 있다.

3.3 데이타베이스 회복 과정

데이타베이스 회복 과정은 단일 노드의 고장에 대한 회복과 여러 노드의 동시 고장에 대한 회복 과정으로 나눌 수 있다. 먼저 단일 노드 N_f가 고장날 경우, 데이타의 가용성을 위하여 다른 노드(N_r)에서 N_f에 대한 회복을 즉시 수행한다.[5] 이때 회복되어야 할 대상은 N_f가

4) 단계 4에 나타나지만 Backup 노드에서 경우에 따라 P_i의 최신 버전을 캐싱할 수도 있다. 즉, 기존의 Backup 노드에서 P_i가 교체된 결과 CopySet 중의 한 노드가 Backup 노드로 변경되고, 그 후 P_i의 갱신없이 소유자 노드에서 P_i를 교체할 경우에는 Backup 노드를 다시 P_i의 CopySet으로 환원하는 것이 가능하다. 그러나 본 논문에서는 알고리즘 기술의 단순함을 위하여 이러한 최적화 방안은 고려하지 않도록 한다.

```
1. RedoLSN = Hi-Value;
2. GLM에서 전송된 PCT에 포함된 모든 페이지 Pi에 대해 다음 과정을 수행.
   • if (PCT[Pi].Backup ≠ -1)          // Backup 노드가 존재
       - PCT[Pi].Backup에게 Pi를 Nr에게 전송하도록 요청. Nr은 Pi를 캐싱.
       - RedoLSN = minimum(RedoLSN, Pi.PageLSN+1);
   • else if (PCT[Pi].CopySet ≠ {})    // CopySet에 노드들이 존재
       - PCT[Pi].CopySet에서 Pi를 현재 캐싱하고 있는 노드 검사.
       - 만약 그런 노드가 존재한다면, 그 노드에서 Pi를 Nr로 전송.
       - RedoLSN = minimum(RedoLSN, Pi.PageLSN+1);
   • else                              // Pi를 디스크에서 액세스
       - RedoLSN = minimum(RedoLSN, PCT[Pi].RLSN);
3. RedoLSN부터 Nf의 로그 디스크에 저장된 로그를 스캔하면서 REDO 수행.
4. 완료되지 않은 Nf의 트랜잭션들에 대해 UNDO 수행.
```

그림 5 단일 노드의 고장에 대한 회복 과정

소유자로 등록된 페이지들로, 최신 버전의 페이지가 N_f의 버퍼에만 캐싱되었으며 디스크에는 반영되지 않은 페이지들이다. 이를 위하여 N_r은 GLM에게 N_f가 소유자로 등록된 페이지에 대한 PCT 정보를 전송받은 후, 그림 5와 같은 단일 노드의 회복 과정을 수행한다.

회복을 위하여 N_f의 로크 레코드의 스캔을 시작할 위치인 RedoLSN을 먼저 파악해야 한다. N_f가 소유자인 페이지들에 대해 Backup 노드나 CopySet이 존재할 경우에는 해당 노드에서 캐싱하고 있는 페이지 버전의 PageLSN을, 그런 노드가 없을 경우에는 RLSN을 고려하여 가장 작은 값으로 RedoLSN을 설정한다. 그리고 Backup 노드나 CopySet이 존재할 경우에는 페이지 버전들을 전송받아 REDO를 수행함으로써 디스크 액세스 빈도수를 줄이도록 한다. 예를 들면, 그림 4에서 노드 N_3가 고장나고 N_4가 N_3의 회복을 담당한다고 가정하자. N_3가 소유자인 페이지는 P이므로 GLM은 PCT[P]를 N_4에게 전송하고, N_4는 P의 Backup 노드인 N_2로부터 P를 전송받는다. 이때 P의 PageLSN이 30이므로 RedoLSN은 31로 설정된다. 이후 REDO 단계에서 31보다 큰 LSN을 갖는 로그 L_3를 N_3의 로그 디스크에서 액세스한 후, 이 내용을 P에 적용시킴으로써 P의 최신 버전을 생성한다.

ARIES/SD와 비교할 때, 그림 5의 회복 과정의 장점은 다음과 같다. 첫째, N_f의 회복을 위하여 로그 병합을 수행할 필요가 없다. 둘째, Backup 노드에 캐싱된 버전에 대해서는 REDO를 할 필요가 없으므로, 스캔해야 될 로그의 양이 줄어든다. 마지막으로, Backup 노드나 CopySet에 캐싱된 페이지 버전을 전송받아 REDO를 수행하므로, REDO 단계에서 데이타 페이지에 대한 디스크 액세스 수가 줄어든다.

여러 노드의 고장에 대한 회복 알고리즘은 ARIES/SD와 기본적으로 동일하므로 본 논문에서 별도로 기술하지는 않도록 한다. 그 이유는 여러 노드가 동시에 고장난 경우에는 단일 노드 고장에 대한 회복과 달리 로그 병합이 필요하기 때문이다. 즉, 페이지 P의 소유자 노드와 Backup 노드가 동시에 고장날 경우 디스크에 저장된 P의 버전을 이용하여 회복을 수행하여야 하는데, 2VC는 ARIES/SD와 마찬가지로 P가 디스크에 저장되지 않은 상태에서 여러 노드에서 갱신될 수 있다. 따라서 디스크에 저장된 P의 버전을 액세스한 후 그 이후에 실행된 P에 대한 모든 노드의 로그들을 순서대로 적용하여야 하므로, 여러 노드의 로그들을 LSN 순서로 병합하는 과정이 필요하다. 단, 이 경우에도 P의 CopySet이 고장나지 않은 노드를 포함할 경우에는 이 노드에 저장된 P의 최신 버전을 이용하는 것이 가능하다. 뿐만 아니라, P의 최신 버전은 아니지만 디스크에 저장된 P의 버전보다는 이후 버전을 캐싱한 노드가 존재할 수 있고, 이를 이용할 경우 P에 대한 REDO 작업을 최적화하는 것이 가능하다[11].

4. 성능평가 모형

본 절에서는 2VC 알고리즘의 성능을 평가하기 위한 성능평가 모형에 대해 설명한다. 모의실험을 위해서 개발한 DSS 모형이 그림 6에 나타난다. 모의실험은 미국의 MCC에서 개발한 CSIM 언어[19]를 이용하여 수행되었다.

5) 데이타 디스크와 로그 디스크는 모든 노드에 의해 공유되므로, 정상적으로 동작중인 모든 노드가 N_f에 대한 회복을 수행할 수 있다[10,11]. 그러나 N_f의 회복 작업에 의해 N_r의 트랜잭션 처리 성능이 저하되므로 정상적인 노드 중에서 현재 부하가 가장 적은 노드를 N_r로 선정하는 것이 바람직하다.

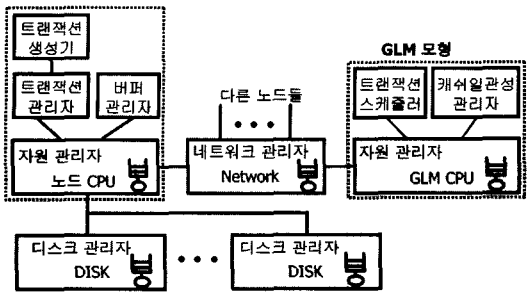

그림 6 모의실험 모형

성능 평가를 위해 개발한 DSS는 하나의 GLM과 네트워크를 통해 연결된 여러 개의 노드로 구성되고, 각 노드는 모든 디스크들을 공유한다. 각 노드마다 메모리 버퍼가 존재하며, 노드의 버퍼 관리자는 LRU 정책을 바탕으로 자신의 버퍼를 관리한다. 트랜잭션 생성기는 트랜잭션을 차례대로 생성한다. 각 트랜잭션의 실질적인 실행(로크 요청 및 캐쉬 액세스 요청 등)은 트랜잭션 관리자에 의해 수행된다.

GLM은 각 노드에서 요청된 로크 및 캐쉬 일관성을 관리한다. 이를 위해 GLM은 트랜잭션 스케줄러와 캐쉬 일관성 관리자를 갖는다. 트랜잭션 스케줄러는 레코드 단위의 2 단계 로킹 기법을 지원하며 교착 상태 해결을 위해 대기 그래프에 바탕을 둔 교착 상태 탐지 및 해결 기법을 지원한다. 캐쉬 일관성 관리자는 ARIES/SD와 ORAC의 캐쉬 연합 알고리즘, 그리고 본 논문에서 제안한 2VC를 각각 구현한다. 전체 시스템의 병목 현상을 방지하기 위하여 GLM은 다른 노드들에 비해 성능이 우수하다고 가정한다.

본 논문에서 사용한 입력 매개 변수를 표 1에서 요약한다. 각 변수들의 구체적인 값은 [7]에서 주로 참조하였다.

공유 디스크의 수는 6으로 가정했으며, DSS를 구성하는 노드의 수는 4개에서 14개까지 변경하면서 실험을 하였다. 디스크 액세스 시간은 0.01초에서 0.03초까지의 일양 분포를 따른다. 데이타베이스의 크기는 1024 페이지이며, 각 노드의 버퍼 크기는 데이타베이스 크기의 25%와 37.5%인 256 페이지와 384페이지로 설정하였다. 디스크와 CPU는 FIFO 큐를 이용하여 I/O 요청 및 로크 요청 등을 들어온 순서대로 처리한다. 네트워크 관리자는 100Mbps의 대역폭을 갖는 FIFO 서버로 구현되었으며, 노드들 간의 메시지 전송과 GLM과 각 노드 사이의 메시지 전송을 담당한다. 네트워크를 통해 메시지를 전송하는 과정을 표현하기 위해 노드의 CPU 및 GLM의 CPU는 메시지마다 FixedMsgInst만큼의 고정된 명령 수와 한 페이지가 추가될 때마다 PerByteMsgInst

표 1 입력 매개 변수

시스템 구성 변수		
GCPUSPEED	GLM CPU의 속도	100 MIPS
LCPUSPEED	노드 CPU의 속도	50 MIPS
NetBandwidth	네트워크의 데이타 전송 속도	100 Mbps
NDBMS	노드의 수	4 ~ 14
DISK_NUM	공유 디스크의 수	6
MinDiskTime	최소 디스크 액세스 시간	0.01초
MaxDiskTime	최대 디스크 액세스 시간	0.03초
DBSIZE	데이타베이스의 크기(페이지)	1024
BUFSIZE	노드의 버퍼 크기(페이지)	256, 384
오버헤드 변수		
FixedMsgInst	메시지 처리를 위한 고정 명령 수	20000
PerByteMsgInst	메시지 길이당 추가되는 명령 수	10000
ControlMsgSize	제어 메시지의 길이	256
LockInst	로크 등록/해제를 위한 명령 수	300
PerIOInst	디스크 I/O를 위한 명령 수	5000
트랜잭션 변수		
TRSIZE	트랜잭션당 평균 페이지 액세스 수	8
TRSZDV	트랜잭션 길이의 편차	0.5
WriteOpPct	갱신 연산의 비율	5%~80%

만큼의 추가적인 명령 수를 실행한다.

실험의 주요 성능 평가 지수는 단위 시간당 트랜잭션 처리량(Throughput)으로, 초당 완료되는 트랜잭션 수를 나타낸다. 그리고 소유자 노드에서의 페이지 강제 기록 현상을 분석하기 위하여 전체 디스크 액세스 수에서 기록 연산이 차지하는 비율을 나타내는 디스크 기록 비율을 보조 지수로 사용한다. 신뢰성있는 실험 결과를 산출하기 위하여 배치 평균 기법(batch mean method)을 사용하였으며, 본 논문에서 나타난 실험 결과들은 20개의 다른 seed를 이용하여 산출된 결과들의 평균값이다.

5. 실험 결과 분석

본 절에서는 DSS 성능평가 모형을 이용하여 실험한 캐쉬 일관성 기법들의 실험 결과를 분석한다. 구현한 캐쉬 일관성 기법은 2VC와 ARIES/SD(ARIES), 그리고 ORAC의 캐쉬 연합 알고리즘(ORAC)의 세 가지 기법들이다. 다양한 데이타베이스 환경에 따른 캐쉬 일관성 기법들 간의 성능 비교를 위하여 DSS를 구성하는 노드 수와 갱신 연산의 비율, 그리고 노드의 버퍼 크기 등을 변화시키면서 실험을 수행하였다.

5.1 DSS를 구성하는 노드 수의 변화

본 절에서는 DSS를 구성하는 노드 수의 변화에 따른 단위 시간당 트랜잭션 처리량(TPS)을 비교한다. 전체 DSS는 4개에서 14개의 노드로 구성되며, 각 노드는 데이타베이스 페이지의 25%인 256개의 페이지를 캐싱할

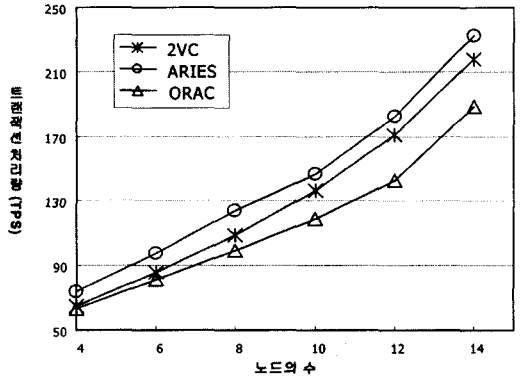

그림 7 노드 수의 변화에 따른 트랜잭션 처리량

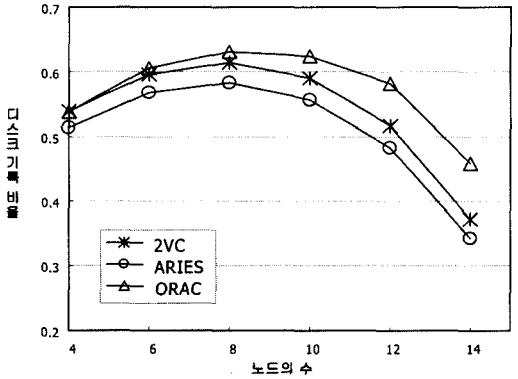

그림 8 노드 수의 변화에 따른 디스크 기록 비율

수 있는 버퍼를 갖는다. 그림 7은 갱신 연산의 비율을 40%로 고정할 경우 2VC와 ARIES, 그리고 ORAC의 트랜잭션 처리량을 나타낸다.

노드 수가 증가함에 따라 동시에 많은 수의 트랜잭션들을 실행할 수 있으며, 각 노드의 버퍼 크기를 합한 전체 메모리 크기도 커지므로 버퍼 적중율이 증가하여 모든 캐쉬 일관성 기법들의 성능이 증가한다. 예상했던 대로 캐쉬 일관성 기법들 중에서는 ARIES의 성능이 가장 좋은 것으로 나타났다. ARIES는 빠른 회복을 위한 추가적인 고려를 전혀 하지 않으므로 정상적인 트랜잭션 처리 시점에서는 부가적인 오버헤드가 없다. 2VC와 ORAC은 빠른 회복을 위하여 Backup 노드를 통한 강제 기록 연산이 발생하므로 ARIES에 비해서는 정상적인 트랜잭션 처리의 성능이 다소 감소된다.

그러나 성능 감소의 정도는 ORAC에 비해 2VC가 약 20% 정도 개선되는 효과를 나타내었으며, 노드 수가 증가할수록 2VC의 성능 개선 정도가 현저하게 나타났다. 특히 노드 수가 10을 넘어설 경우 ARIES와 2VC의 성능 차이는 10%미만으로 줄어들었지만, ORAC과는 20% 이상의 성능 차이가 발생하였다. 그 이유는 노드 수가 증가할수록 특정 페이지를 갱신한 노드들의 수가 증가하는데, ORAC의 경우에는 이들 노드들이 모두 Backup 노드로 등록되며 이중 하나의 노드라도 버퍼 교체를 요청할 경우 소유자 노드에서 강제 기록이 발생하기 때문이다. 이와는 달리 2VC에서는 직전 소유자 노드만 Backup으로 등록되므로 소유자 노드에 의한 강제 기록 빈도수가 상대적으로 줄어든다. 그림 8은 전체 디스크 액세스 수에서 기록 연산이 차지하는 비율을 나타내는데, 노드 수가 증가할수록 ORAC에서의 기록 비율이 2VC나 ARIES에 비해 커짐을 알 수 있다. 뿐만 아니라, 노드 수가 클 경우 Backup 노드에서 버퍼 교체가 발생하더라도 이를 대치할 CopySet이 존재할 가능성도 높아지므로, 2VC의 디스크 기록 비율은 노드 수가 증가할수록 ARIES와 비슷해짐을 알 수 있다.

노드 수가 작을 경우에는 ORAC과 2VC의 성능이 거의 동일한 것으로 나타났다. 그 이유는 그림 8에서도 나타나듯이 두 기법들 간의 디스크 기록 비율에 큰 차이가 없기 때문이다. 즉, 노드 수가 작으므로 ORAC에서 Backup 노드가 여러 개 존재할 가능성이 크지 않으며, 2VC에서 Backup 노드 외에 CopySet에 추가적인 노드가 존재할 가능성도 줄어든다.

5.2 갱신 연산의 비율 변화

본 절에서는 갱신 연산의 비율 변화에 따른 단위 시간당 트랜잭션 처리량을 비교한다. 전체 DSS는 8개의 노드로 구성하고 각 노드는 256개의 페이지를 캐싱할 수 있는 버퍼를 갖는다. 그림 9는 갱신 연산의 비율이 5%에서 80%까지 변화할 경우 2VC와 ARIES, 그리고 ORAC의 트랜잭션 처리량을 나타낸다. 그림 10은 그림 9의 실험 환경에서 각 캐쉬 일관성 기법에 대한 디스크 기록 비율의 변화를 나타낸다.

갱신 연산 비율이 5%나 10%로 매우 작을 경우에는 모든 캐쉬 일관성 기법들이 거의 동일한 성능을 나타내

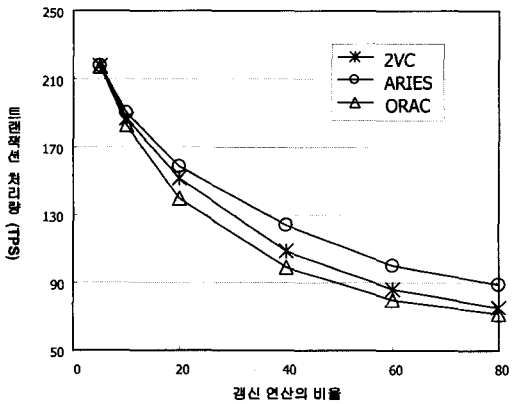

그림 9 갱신 연산 비율의 변화에 따른 트랜잭션 처리량

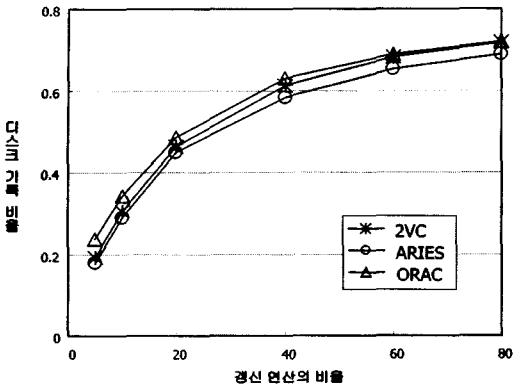

그림 10 갱신 연산 비율의 변화에 따른 디스크 기록 비율

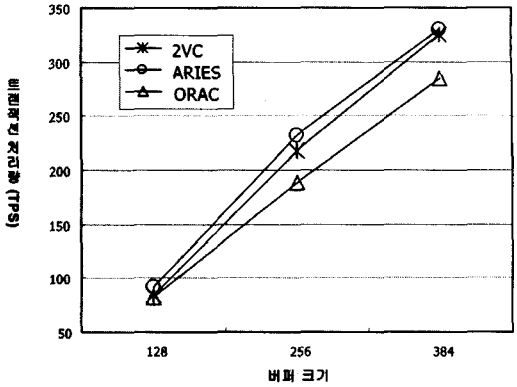

그림 11 버퍼 크기의 변화에 따른 트랜잭션 처리량 (노드 수 = 14)

었다. 이 경우에는 특정 페이지가 디스크에 기록되지 않은 상태에서 여러 노드에서 갱신될 확률이 작기 때문에, 2VC와 ORAC에서 소유자 노드에 의한 강제 기록 현상이 별로 발생하지 않는다. 갱신 연산 비율이 증가함에 따라 ORAC의 성능이 먼저 감소하기 시작하며, 갱신 연산 비율이 20%~40% 범위에서는 2VC에 비해 10% 정도 성능이 감소하였다. 그림 10에 나타나듯이 이 범위에서는 2VC의 디스크 기록 비율이 ARIES와 거의 비슷한데, 그 이유는 갱신 연산 비율이 상대적으로 높지 않으므로 Backup 노드에서 버퍼 교체가 발생하더라도 이를 대치할 CopySet이 존재할 가능성이 높기 때문이다. 그러나 갱신 연산 비율이 60%를 넘어섬에 따라 2VC의 성능은 ORAC과 비슷해져서, ARIES에 비해 2VC는 18%, ORAC은 25% 정도 성능이 낮았다. 이 경우에는 대부분의 트랜잭션 연산이 갱신 연산이므로 판독 연산을 통한 CopySet 노드를 포함할 가능성이 매우 작아진다. 따라서 Backup 노드에서 버퍼 교체가 일어날 경우에는 거의 대부분 소유자 노드에서 강제 기록 연산이 발생한다. ORAC에서는 Backup 노드의 수가 2VC에 비해 많으므로 버퍼 교체에 따른 강제 기록 연산의 가능성이 더욱 커진다.

5.3 노드의 버퍼 크기 변화

본 절에서는 각 노드의 버퍼 크기를 변화시켜 캐쉬 일관성 기법들의 성능을 비교한다. 그림 11은 갱신 연산 비율을 40%, 노드 수를 14로 고정한 후, 노드의 버퍼 크기를 데이타베이스 크기의 12.5%(128)와 25%(256), 그리고 37.5%(384)로 변화시켰을 때의 트랜잭션 처리량을 나타낸다.

버퍼 크기가 증가함에 따라 버퍼 히트율이 커지고, 그 결과 모든 캐쉬 일관성 기법들의 성능이 향상되었다. 특히 2VC의 성능이 ORAC에 비해 큰 폭으로 개선되었는데, 그 이유는 노드의 버퍼는 LRU 정책에 의해 페이지 교체를 하므로 버퍼 크기가 커질수록 갱신된 페이지가 버퍼에 상주하는 시간이 길어지기 때문이다. 즉, 단 하나의 Backup 노드를 갖는 2VC의 경우, 버퍼 크기가 증가할수록 Backup 노드에서 버퍼 교체가 일어나기 전에 소유자 노드가 변경될 가능성이 커지므로 강제 기록 현상이 줄어든다. 뿐만 아니라, 버퍼 크기가 증가함에 따라 CopySet에 포함된 노드 수도 증가하고, 그 결과 버퍼 크기가 384일 경우 2VC와 ARIES의 성능이 거의 동일하였다. 이와는 달리, ORAC에서는 페이지를 갱신한 모든 노드들이 Backup 노드로 결정되므로 버퍼 크기가 증가하더라도 가장 오래된 Backup에서는 결국 버퍼 교체가 발생하고 이때 페이지의 강제 기록이 발생한다. 따라서 ORAC에서는 버퍼 크기 증가에 따른 성능 개선이 다른 기법들보다 크지 않았다.

6. 결 론

본 논문에서는 DSS 환경에서 단일 노드의 고장 시 빠른 회복을 지원할 수 있는 캐쉬 일관성 기법인 2VC (Two Version Caching) 알고리즘을 제안하였다. 2VC는 소유자 노드의 고장을 대비하여 추가적인 Backup 노드에 페이지 버전을 유지함으로써 다음과 같은 장점을 갖는다. 첫째, 단일 노드의 회복에 있어서 로그 병합이 필요하지 않으므로, 전통적인 회복 기법인 ARIES/ SD에 비해 빠른 데이타베이스 회복을 지원할 수 있다. 일반적으로 단일 노드의 고장이 여러 노드의 동시 고장보다 빈번하므로, 단일 노드의 회복에서 우수한 성능을 보이는 것은 매우 중요하다. 둘째, 단 하나의 Backup 노드만 유지함으로써 여러 개의 Backup 노드를 갖는 ORAC의 캐쉬 연합정책에서 발생하는 빈번한 디스크 저장 오버헤드를 피할 수 있다. 따라서 ORAC에 비해 정상적인 트랜잭션 처리의 성능을 향상시킬 수 있다.

2VC의 성능을 분석하기 위하여 DSS를 위한 모의실험 환경을 개발하였고, 다양한 시스템 구성 하에서 실험을 하였다. 실험 결과 2VC는 DSS를 구성하는 노드 수가 증가하거나, 혹은 각 노드의 버퍼 크기가 클 경우 ORAC에 비해 성능 개선 효과가 현저하였다. 뿐만 아니라, 노드 수와 버퍼 크기가 충분히 큰 경우에는 정상적인 트랜잭션 처리량의 관점에서 ARIES/SD와 큰 차이가 나지 않았다. 메모리 기술의 발전에 따라 노드 버퍼의 크기가 지속적으로 증가하며, SAN이나 NAS와 같은 네트워크 저장 기술의 발전에 따라 DSS 규모가 급속히 확대되고 있는 추세를 감안할 때, 2VC의 이러한 성능 특징은 매우 긍정적인 것으로 판단된다.

참 고 문 헌

[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] M. Yousif, "Shared-Storage Clusters," *Cluster Computing*, vol.2, no.4, pp.249-257, 1999.

[3] E. Rahm, "Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems," *ACM Trans. on Database Syst.*, vol.18, no.2, pp.333-377. 1993.

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

[5] R. Yevich and S. Lawson, *DB2 Universal Database for OS/390*, Prentice Hall, 2002.

[6] *Oracle 9i Real Application Clusters Concepts Release 1*, Oracle Corp., part A89867-02, 2001.

[7] H. Cho, "Cache Coherency and Concurrency Control in a Multisystem Data Sharing Environment," *IEICE Trans. on Information and Syst.*, vol.E82-D, no.6, pp.1042-1050, 1999.

[8] J. Josten, C. Mohan, I. Narang, and J. Teng, "DB2's Use of the Coupling Facility for Data Sharing," *IBM System J.*, vol.36, no.2, pp.327-350, 1997.

[9] T. Lahiri et al., "Cache Fusion: Extending Shared-Disk Clusters with Shared Caches," in: *Proc. 27th Int. Conf. VLDB*, pp.683-686, 2001.

[10] 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 Int. Conf. VLDB*, pp.193-207, 1991.

[11] 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.

[12] A. Dan, P. Yu, and A. Jhingran, "Recovery Analysis of Data Sharing Systems under Deferred Dirty Page Propagation Policies," *IEEE Trans. on Parallel and Distributed Syst.*, vol.8, no.7, pp. 695-711, 1997.

[13] R. Rastogi et al., "Distributed Multi-Level Recovery in Main-Memory Databases," *Distributed and Parallel Databases*, vol.6, no.1, pp. 41-71, 1998.

[14] C. Mohan et al., "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," *ACM Trans. on Database Syst.*, vol.17, no.1, pp.94-162, 1992.

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

[16] *Oracle 7 Parallel Server Concepts and Administration*, Oracle Corp., part A42522-1, 1996.

[17] T. Lahiri et al., "Fast-Start: Quick Fault Recovery in Oracle," in: *Proc. ACM SIGMOD*, pp.593-598, 2001.

[18] *Oracle 9i Database Concepts Release 1*, Oracle Corp., part A88856-02, 2001.

[19] H. Schwetman, *CSIM User's Guide for use with CSIM Revision 16*, MCC, 1992.

조 행 래

1988년 서울대학교 컴퓨터공학과 학사
1990년 한국과학기술원 전산학과 석사
1995년 한국과학기술원 전산학과 박사
1995년~현재 영남대학교 전자정보공학부 부교수. 관심분야는 분산/병렬 데이타베이스, 트랜잭션 처리, DBMS 개발 등

정 용 석

2001년 영남대학교 컴퓨터공학과 학사
2003년 영남대학교 컴퓨터공학과 석사
2003년~현재 (주)퓨전소프트 정보기술연구소 연구원. 관심분야는 분산/병렬 데이타베이스, 데이타베이스 회복 등

이 상 호

2003년 영남대학교 컴퓨터공학과 학사
2003년~현재 영남대학교 컴퓨터공학과 석사과정. 관심분야는 분산/병렬 데이타베이스, 실시간 트랜잭션 처리 등