□ 기술해설 □

ODBMS 구조

클라이언트-서버 DBMS: 연구 동향

중앙대학교 강 현 철*

목 차

1. 서 론

워크스테이션, LAN 및 GUI 기술의 지속적인 발전과 그로 인한 가격 하락으로 인해 다수의 워크스테이션을 LAN으로 연결한 환경에서 GUI 기반의 클라이언트-서버 소프트웨어 구조는 1990년대의 컴퓨팅 환경을 구성하는 주요 요소가 되었다. 클라이언트-서버 소프트웨어의 작동 구조는 서버 프로세스는 데이타, 컴퓨팅, 화일 등을 관리 및 서비스해주고 클라이언트 프로세스는 서버에게 요구하여 제공받은 서비스를 이용하여 응용을 수행하고 GUI를 통하여 사용자와 접촉하는 것이다.

데이타베이스 시스템도 종래의 중앙 집중 환경에서 운용되는 것보다 클라이언트-서버 환경에서 데이타 서버로서 여러 클라이언트 사이트의 응용이 요구하는 데이타를 서비스 및 관리해주는 기능을 더 중요시하게 되었다. 이미 많은 상용 관계 DBMS들이 서버 DBMS의 기능을 갖추어 클라이언트-서버 환경을 지원하고 있으며, Iris, ORION, O2, GemStone, Objectivity, Ontos, ObServer, ObjectStore 등의 객체 지향 DBMS들도 워크스테이션들을 LAN으로 연결한 환경에서 클라이언트-서버 소프트웨어 구조를 지원하고 있다.

본 논문에서는 클라이언트-서버 DBMS에 관한 연구 동향을 조사하고 제안된 기법들을 정리하여 소개한다. 2절에서는 클라이언트-서버 DBMS의 구조 및 모델을 분류하여 기술하고, 3절에서는 클라이언트-서버 DBMS에 관한 최근의 국외 연구 동향을 조사하여 설명한다. 4절에서는 결론을 맺는다.

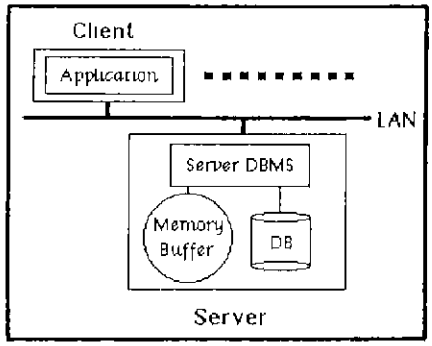
2. 클라이언트-서버 DBMS의 분류

클라이언트-서버 DBMS와 관련된 초기의 연구들 [1,2,3]은 클라이언트-서버 DBMS의 개념 및 구조 또는 참조 모델 등이 정립되지 않은 상태에서 수행되었다. 이들 연구 이후 클라이언트-서버 DBMS에 관한 연구가 축적되면서 클라이언트-서버 DBMS의 구조 및 모델이 분류되고 그 개념이 정립되어 가고 있다. 본 절에서는 클라이언트-서버 DBMS의 구조 및 모델을 분류해보도

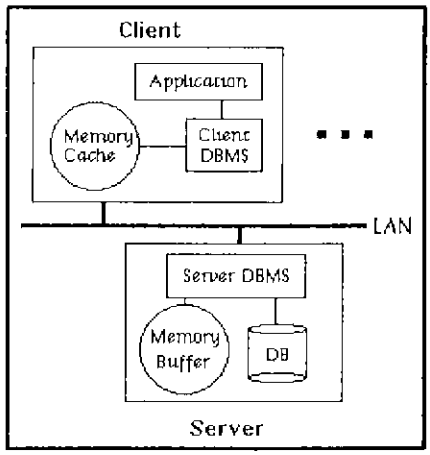
* 종신회원

록 한다. 클라이언트-서버 DBMS를 분류하는 기준으로는 다음과 같은 것들이 있다:

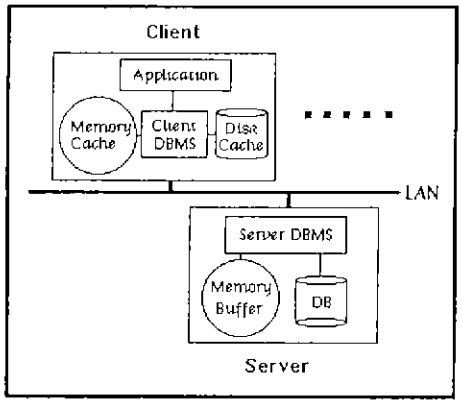
(1) 캐쉬: 클라이언트 사이트에 데이타 캐쉬의 존재 여부 및 캐쉬의 종류



(a) CSNC 구조



b) CSMC 구조



(c) CSDC 구조

그림 1 클라이언트-서버 DBMS 구조

(2) 서버의 데이타 서비스: 클라이언트의 트랜잭션이 데이타베이스 서버에게 요구하는 서비스의 형태

(3) 데이타 전송: 클라이언트와 서버 간의 데이타 전송의 단위

2.1 캐쉬에 따른 분류

클라이언트 사이트에 캐쉬가 존재하는지의 여부는 클라이언트-서버 DBMS의 분류에 있어 가장 중요한 기준이다. 클라이언트 사이트에 캐쉬가 존재하지 않는 경우와 존재하는 경우로 우선 나눌 수 있고, 캐쉬가 존재하는 경우는 다시 메모리 캐쉬만 존재하고 디스크 캐쉬는 존재하지 않는 경우와 둘다 존재하는 경우로 나눌 수 있다[1]. 즉, 그림 1과 같이 모두 세가지 형태의 클라이언트-서버 DBMS 구조를 생각할 수 있다. 클라이언트-서버 DBMS의 연구자들 간에 이들 세가지 구조를 지칭하는 적절한 용어가 제시되지 않았다. 본 논문에서는, 캐쉬가 전혀 존재하지 않는 구조를 캐쉬 부재 클라이언트-서버 DBMS 또는 CSNC (Client-Server DBMS with No Cache), 메모리 캐쉬만 존재하는 구조를 메모리 캐쉬 클라이언트-서버 DBMS 또는 CSMC (Client-Server DBMS with Memory Cache), 그리고 메모리 캐쉬 및 디스크 캐쉬가 함께 존재하는 구조를 디스크 캐쉬 클라이언트-서버 DBMS 또는 CSDC (Client-Server DBMS with Disk Cache)라 부르도록 한다[2].

CSNC 구조는 주로 상용 관계 DBMS들이 사용하고 있는 방식이다. 이 경우 클라이언트 사이트에서는 응용 프로그램이 수행되면서 서버 사이트로는 데이타 접근을 요구하여 응용에 필요한 데이타를 받아 본다. 즉, 클라이언트 머쉰의

1) 메모리 캐쉬는 주기억 장치 캐쉬를 의미한다. 클라이언트-서버 DBMS의 연구에서 많은 연구자들이 주기억 장치 캐쉬를 디스크 캐쉬와 구분하여 메모리 캐쉬란 용어로 부르고 있다. 본 논문에서는 이하 따로 언급이 없는 한 주기억 장치를 메모리라 불러 디스크와 구분하도록 한다.

2) [Deli 92,93]에서는 CSNC를 표준 클라이언트-서버(Standard Client-Server), CSMC를 RAD-UNIFY 형태의 클라이언트-서버, 그리고 CSDC를 향상된 클라이언트-서버(Enhanced Client-Server)라 불렀다.

처리 및 저장 능력은 트랜잭션을 포함하는 응용의 수행 (GUI를 통한 사용자와의 접촉(interaction), 수행 결과의 전달(presentation), 서버로의 데이타 요구 등)에 모두 할당되며 서버 데이타베이스의 관리 및 처리는 전적으로 서버가 담당하는 것이다[4].

메모리 캐쉬는 트랜잭션의 요청에 의하여 클라이언트 사이트로 전송되어 온 데이타를 이후의 트랜잭션들이 재사용할 수 있도록 클라이언트의 메모리 버퍼에 저장해 놓은 것이다. 즉, 한 트랜잭션이 완료되더라도 그것이 접근했던 데이타를 메모리 캐쉬에 남겨 두어 이후의 트랜잭션들이 재사용하도록 하는 것인데 이를 많은 연구자들이 트랜잭션 간 캐쉬(inter-transaction cache)라 부른다. CSMC 구조는 메모리 캐쉬를 유지함으로써 클라이언트 머쉰의 처리 및 저장 능력을 활용하고 서버 DBMS의 부하를 그만큼 감소시키는 효과를 얻는다. 대신에 클라이언트 사이트에서는 메모리 캐쉬의 일관성(cache consistency)을 유지해야 하는 부담을 안게 된다. CSMC 구조에서 클라이언트 사이트의 트랜잭션은 데이타를 접근할 수 있는 소스(source)를 세군데 갖게 된다. 읽기 접근의 경우 접근 프로토콜은 먼저 클라이언트의 메모리 캐쉬를 검색하고 미스(miss)가 발생하면 서버의 메모리 버퍼를 검색하고 그것도 미스하면 서버의 디스크를 접근하게 된다. 즉, 기억장치 계층 (memory hierarchy) 은 그림 2와 같다.

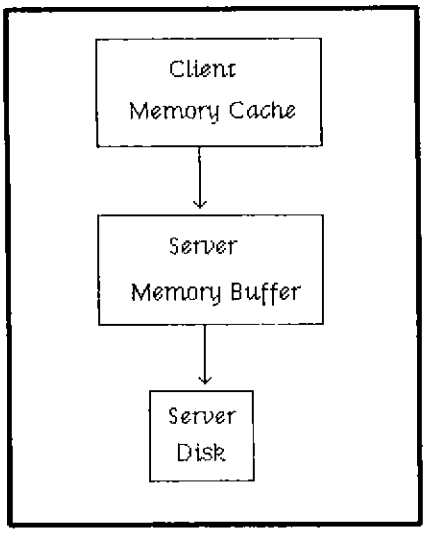


그림 2 CSMC 구조의 기억장치 계층

CSMC 구조에서는 CSNC 구조와는 달리 클라이언트 사이트에 메모리 캐쉬를 관리하고 데이타 처리를 수행할 수 있는 DBMS가 필요하다. 이 클라이언트 DBMS의 기능으로 인하여 서버 DBMS의 부하를 줄일 수 있으며 메모리 캐쉬의 양이 많아질수록 서버 DBMS는 단지 클라리언트가 요구하는 데이타를 전송해주고 동시성 제어를 통하여 데이타의 일관성을 유지하는 역할만 수행하는 형태로 축소될 수 있다[5].

메모리 캐쉬는 클라이언트 사이트가 고장나거나 서버와의 접속이 종료될 경우 (즉, 오프 라인 상태) 캐쉬된 데이타를 모두 상실하게 된다[3]. 즉, 클라이언트 사이트가 고장에서 회복되거나 다시 서버 머쉰과 접속되어 온라인 상태로 바뀌었을 때 메모리 캐쉬는 비어 있는 것이다. 이에 반하여 디스크 캐쉬는 클라이언트 사이트의 고장이나 접속 종료 후에도 캐쉬된 데이타를 계속 유지한다. 즉, 메모리 캐쉬는 비영구적 캐쉬(nonpersistent cache)인데 반해, 디스크 캐쉬는 영구적 캐쉬(persistent cache)인 것이다.

CSDC 구조는 CSMC 구조에 영구적 캐쉬 기능을 추가한 것이다[2,6-8]. CSDC 구조의 클라이언트 DBMS는 메모리 캐쉬 뿐만 아니라 디스크 캐쉬까지 관리하여야 하므로 CSMC 구조의 클라이언트 DBMS보다 더 복잡한 기능을 갖게 된다. 반면에 클라이언트 사이트가 메모리 캐쉬에 비하여 훨씬 큰 용량의 영구적 디스크 캐쉬를 확보하고 있으므로 CSMC 구조에 비하여 클라이언트의 머쉰의 처리 및 저장 능력을 더욱 활용하여 서버의 부하를 더 줄일 수 있고 서버 사이트의 고장시에도 클라이언트 디스크 캐쉬에 기반한 독자 처리 기능을 가져 데이타 가용성(availability)을 높일 수 있다.

디스크 캐쉬가 존재할 경우 클라이언트의 트랜잭션이 접근할 수 있는 데이타 소스로는 클라

3) 클라이언트 사이트가 서버와의 접속을 종료한다는 것은 사용자가 클라이언트 머쉰의 전원을 끄고 서버 머쉰 및 LAN으로부터 오프 라인시키는 것을 의미한다. 실세계의 사무실에서 데스크 톱 워크스테이션을 클라이언트 머쉰으로 사용하는 사용자는 매일 출퇴근시 자신의 워크스테이션을 켰다 끄는 것이 보통이다.

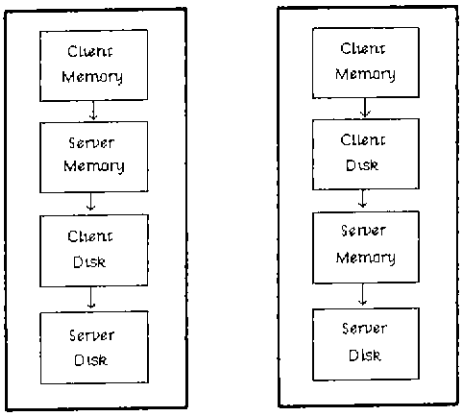
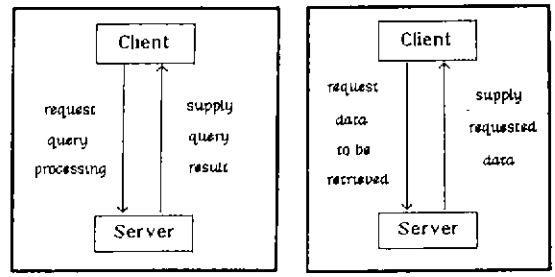


그림 3 CSDC 구조의 기억장치 계층

이언트 메모리, 클라이언트 디스크, 서버 메모리, 그리고 서버 디스크의 네가지가 존재한다. 따라서 기억장치 계층을 그림 3(a)와 (b)처럼 두가지로 생각할 수 있다[8]. 그림 3(a)는 클라이언트에서 디스크 I/O를 수행하는 비용보다 네트워크를 통하여 서버 메모리를 접근하는 비용이 더 낮은 환경에서의 메모리 계층을 나타내고 그림 3(b)는 그 반대 경우를 나타낸다. 읽기 접근의 경우 접근 프로토콜은 먼저 클라이언트의 메모리 캐쉬를 검색하고 미스가 발생하면 서버 메모리 버퍼 및 클라이언트 디스크 캐쉬를 기억장치 계층에 따른 순서로 검색하고 그것에서 모두 미스가 발생하면 마지막으로 서버의 디스크를 접근하게 된다.

2.2 서버의 데이타 서비스에 따른 분류

클라이언트의 트랜잭션이 서버에게 요구하는 서비스의 형태에는 질의를 서버에게 전송하여 그 결과를 넘겨 받는 형태와 질의를 처리하는 데 필요한 데이타를 요청하여 넘겨 받는 형태가 있다. [9]에서는 전자를 질의 전송 (query shipping), 그리고 후자를 데이타 전송 (data shipping)이라 불렀다(그림 4 참조). 질의 전송은 상용 관계 DBMS들이 주로 사용하는 방법으로서 일반적으로 질의 처리가 전적으로 서버 사이트에서 이루어지고 클라이언트의 트랜잭션은 그 결과만 받아본다. 따라서 클라이언트 사이트에 데이타가



(a) 질의 전송 (b) 데이타 전송

그림 4 서버의 데이타 서비스 형태

캐쉬되지 않는 것이 보통이다(즉, CSNC 구조).

질의 전송의 경우에는 클라이언트 사이트에서 질의 처리 또는 데이타 검색이 수행되지 않는 것이 보통이지만, 데이타 전송의 경우에는 클라이언트 트랜잭션이 검색해야 할 데이타를 서버로부터 전송받으므로 클라이언트 DBMS에 의한 질의 처리가 수행되어 서버의 부하를 줄여 주게 된다. 그리고 전송된 데이타는 질의 전송과는 달리 캐쉬되는 것이 보통이다(즉, CSMC 또는 CSDC 구조). 데이타 전송의 경우 클라이언트와 서버 간의 데이타 전송 단위는 [10]의 분류에 따르면 객체, 디스크 페이지, 그리고 화일 등이 있는데 이중 페이지가 가장 많이 연구되고 있으며 클라이언트-서버 구조를 지원하는 객체지향 DBMS들이 가장 많이 사용하는 방식이다[8,9,11-13].

질의 전송은 CSNC 구조에서 사용되는 것이 보통이지만 [2,6,7]에서는 CSDC 구조에서의 질의 전송을 제안하였다. 전송된 질의 결과는 클라이언트의 디스크 캐쉬에 저장되었다가 이후 수행되는 클라이언트 트랜잭션들이 동일하거나 유사한 질의를 제기하였을 경우 재사용된다. 서버 데이타의 변경으로 캐쉬된 질의 결과가 정확성을 잃으면(outdated) 이의 재사용시 서버로부터 질의 결과가 캐쉬된 이후의 변경 사항만을 전송 받아 캐쉬된 질의 결과에 반영한다. 이를 캐쉬의 누증 갱신(incremental refresh)이라 부른다. 이를 위하여 서버 DBMS는 질의 결과의 캐쉬 이후 발생하는 데이타의 변경 사항을 따로

4) 페이지를 단위로 클라이언트에 데이타를 전송하는 서버를 [Ston 90]에서는 블록 서버(block server)라 불렀다.

기록해 두어야 한다.

2.3 데이타 전송에 따른 분류

[10]에서는 클라이언트와 서버 간의 데이타 전송의 단위를 기준으로 클라이언트-서버 DBMS를 분류하였다. 이 연구에서는 객체 지향 DBMS의 관점에서 서버가 클라이언트로 전송하는 데이타의 단위를 객체, 디스크 페이지, 그리고 화일의 세가지로 구분하였다. 객체를 전송 단위로 하는 구조를 객체 서버 구조라 하고, 마찬가지로 나머지 두가지 구조는 각각 페이지 서버, 화일 서버 구조라 한다. 그런데 이 세 구조는 모두 CSMC 구조에 해당하는 것들이다.

객체 서버 구조는 서버와 클라이언트가 모두 객체의 의미(semantic)를 이해하고 있는 경우로서 객체의 검색은 서버가 담당한다. 반면 페이지 서버 구조는 객체의 의미가 드러나지 않는 경우로서 클라이언트도 자신의 DBMS로 객체 검색을 수행하며 데이타의 캐쉬는 객체 또는 페이지 또는 둘다의 형태로 이루어 진다. 화일 서버 구조는 페이지 서버 구조를 단순화한 것으로서 클라이언트 사이트에서 Sun의 NFS와 같은 원격 화일 서비스를 통하여 서버의 디스크 페이지를 직접 접근하는 기능이 추가된 것이다.

3. 연구 동향

3.1 초기 연구 및 최근 연구 내용

클라이언트-서버 DBMS 구조가 데이타베이스 연구자들(database community)에 의해 주목 받기 시작한 것은 80년대 중반부터이다. [1]에서는 DBMS의 기능들을 LAN으로 연결된 전위(front-end) 머쉰과 후위(back-end) 머쉰의 두 머쉰으로 분리할 경우 이러한 분리가 없이 한 머쉰에서 중앙집중 DBMS를 수행시키는 경우에 대하여 얼마나 성능이 향상되는지를 측정하는 연구를 수행하였다. INGRES의 기능을 모두 여섯 층(layer)으로 나누고 모든 층을 전위 머쉰에 위치시키는 경우 (즉, 중앙집중 INGRES로서 후위 머쉰은 존재하지 않는 경우)에 대하여 하부의 한 층씩 그 기능 수행을 후위 머쉰으로 이동시킬 경우 성능에 미치는 영향을 측정하였다.

[2]에서는 한대의 메인프레임에 여러대의 워크스테이션을 LAN으로 연결한 환경에서 메인프레임 데이타베이스로부터 워크스테이션 데이타베이스로 데이타를 캐쉬하고 그 일관성을 유지하는 기능을 제공하는 ADMS± 라는 DBMS를 제안하였다. 데이타 캐쉬는 질의 전송의 형태로 수행된다. 즉, 메인프레임 데이타베이스에 대하여 수행된 질의 결과인 뷰(view)가 워크스테이션 데이타베이스로 캐쉬되어 재사용되는 것이다. 데이타 변경은 메인프레임에서만 허용되고 따라서 캐쉬 일관성 유지를 위하여 뷰 인덱싱(view indexing) [14]에 기반한 뷰의 누증 갱신(incremental view refresh) 기법[15]을 이용한다. 이 기법은 메인프레임의 데이타 변경 사항 중 캐쉬된 뷰에 영향을 미치는 부분만을 워크스테이션으로 전송함으로써 데이타 전송 부담을 줄이고, 캐쉬된 뷰를 생성하기 위하여 메인프레임에서 질의의 반복 수행을 피하게 해 주는 장점을 지닌다. 이를 위하여 메인프레임 DBMS는 서버 데이타의 변경 사항을 변경 백로그(update backlog)에 기록한다.

[3]에서는 CAD/CAM 등의 공학 설계 환경을 효율적으로 지원할 수 있는 워크스테이션과 서버로 구성된 AIM-P 데이타베이스 시스템에서의 객체 버퍼 관리 기법을 연구하였다 설계 응용에서 서버 데이타베이스로부터 객체를 워크스테이션으로 캐쉬(check-out)하여 조작한 후 변경된 객체의 새 버젼을 다시 서버 데이타베이스로 저장(check-in)하는 데 있어 서버와 여러대의 워크스테이션들이 상호작동(cooperate)하는 기법을 제안하였다.

이들 초기 연구들 이후 90년대에 접어들면서 클라이언트-서버 DBMS에 관한 연구가 아주 활발히 진행되고 있다. 최근의 주요 연구 내용을 대별하면 다음과 같다:

(1) 클라이언트-서버 DBMS 구조들의 성능 및 확장성(scalability) 비교
(2) 트랜잭션 관리 기법
(3) 캐쉬 일관성 유지 기법
(4) 캐쉬 관리 기법

3.2 클라이언트-서버 DBMS 구조 비교

클라이언트-서버 DBMS 구조들의 성능 및 확장성(scalability) 비교 연구는, 2절에서 분류한 여러 유형의 클라이언트-서버 DBMS 구조들의 성능 (서버의 처리율(throughput), 질의 처리의 응답시간(response time) 등)을 다양한 질의 및 변경 부하(query/update workloads)에 대하여 비교하고 클라이언트 수의 증가가 어느 정도 성능 저하를 초래하는지에 따라 이들 클라이언트-서버 DBMS 구조들의 확장성을 비교하는 것이다. 이하 이들 비교 연구를 간단히 살펴보도록 한다.

[6]에서는 CSNC 구조와 CSDC 구조를 기반으로 하는 네가지의 클라이언트-서버 DBMS 구조를 질의 전송에 대하여 비교하였다. 이들 구조는 SCS(Standard Clent-Server), CS-MD (Client-Server with Multiple Disks), E-CS (Enhanced Client-Server), 그리고 E-CS-LB(E-CS with Log Buffer)이다. SCS는 본 논문에서 제시한 용어로는 CSNC에 해당된다. CS-MD는 CSNC의 변형으로서 서버에 다수의 디스크를 두고 서버 데이타베이스를 이들 디스크 각각에 완전 중복 저장(full replication)하여 서버 디스크 I/O를 병렬 수행할 수 있도록 한 구조이다. E-CS는 CSDC에 해당되는데 캐쉬된 질의 결과가 해당되는 서버 데이타의 변경으로 정확성을 잃으면 3.1절에서 언급한[2]의 기법으로 누증 갱신한다. E-CS-LB는 E-CS의 변형으로서 클라이언트에 캐쉬된 질의 결과를 누증 갱신할 때 필요한 서버 데이타의 변경 백로그를 서버의 메모리 버퍼에 저장해 두는 구조이다. CS-MD 및 E-CS-LB 구조는 서버의 성능을 높임으로써 전체 시스템의 확장성을 제고하려는 구조다. 이들 구조의 비교는 큐잉 모델에 기반한 시뮬레이션으로 수행되어 CSNC 구조에 비하여 CSDC 구조의 우월성을 입증하였다.

[7]에서도 질의 전송에 대하여 CSNC, CSMC, 그리고 CSDC 구조에 해당하는 세가지의 클라이언트-서버 DBMS 구조의 성능 및 확장성을 역시 큐잉 모델에 기반한 시뮬레이션을 통하여 비교하였다. 결과는 CSDC 구조가 가장 우수한 것으로 나타났다.

[10]에서는 CSMC 구조에 해당하는 세가지의 클라이언트-서버 DBMS 구조 (객체 서버, 페이지 서버, 그리고 화일 서버 구조)의 성능을 질의 전송이 아닌 데이타 전송에 대하여 비교하였다. 즉, CSMC 구조에서 클라이언트와 서버 간의 데이타 전송 단위가 무엇이냐에 따른 성능 비교이다. WiSS의 여러층(layer)들을 이용하여 이들 세 구조를 프로토타이핑한 시스템들과 단일 WiSS의 네 시스템에서의 질의 응답 시간을 비교하였다[6].

3.3 트랜잭션 관리

클라이언트-서버 DBMS에서 트랜잭션 관리 기법으로는 캐쉬 일관성 유지 기법과 결합되어 동시성 제어 기법이 많이 연구되고 있다[12,16,17]. 연구자들이 캐쉬 일관성 유지와 동시성 제어를 같은 뜻으로 혼용하기도 할 정도로 둘은 따로 생각할 수 없는 문제이다[17]. 동시성 제어로는 거의 대부분의 연구가 로킹(locking) 기법을 사용하고 있는데 3.4절에서 설명하겠지만 클라이언트 사이트로 데이타를 캐쉬하는 것과 더불어 캐쉬된 데이타에 대한 록(lock)도 함께 캐쉬하여 관리하는 록 회수(Callback Locking) 기법이 대표적이다[8,13,17]. 동시성 제어에 대해서는 3.4절에서 캐쉬 일관성 유지와 함께 설명하도록 한다.

한편 클라이언트-서버 DBMS의 고장 회복 기법은 아직 연구가 미흡한 상태다. 현재로는 [9]의 연구가 유일하다. 이 연구에서는 페이지 서버 구조로 되어 있는 클라이언트-서버 버젼의 EXODUS 저장 관리자(storage manager)에서 ARIES [18] 기반의 고장 회복 기법을 구현하고 평가하였다.

3.4 캐쉬 일관성

데이타베이스 서버로부터 클라이언트 사이트

5) 상세한 비교 결과는 해당 논문을 참조.

6) 이 비교에서는 7개의 WiSS 레코드로 구성된 복합 객체(complex object) 1500개로 구성된 데이타베이스를 대상으로 하는 Altair 복합 객체 벤치마크를 개발하여 사용하였다.

로 캐쉬된 데이타는 캐쉬된 이후 서버 데이타베이스의 해당 데이타 사본(data copy)과의 일관성을 유지해야 한다. 데이타 캐쉬는 클라이언트-서버 DBMS의 성능 향상을 가져 오지만 그 대신 클라이언트에 캐쉬된 데이타의 일관성을 유지해야 하는 부담을 낳는 것이다. 이 캐쉬 일관성 유지는 90년대에 들어 클라이언트-서버 DBMS 연구에서 가장 많은 주목을 받고 있는 중요한 문제이다.

캐쉬 일관성 유지 문제는 데이타의 일관성을 유지하는 동시성 제어 문제와 함께 해결하여야 한다. 클라이언트-서버 DBMS 연구에서 캐쉬 일관성 유지와 관련하여 가장 많이 이용되고 있는 동시성 제어 기법은 로킹이다. 본절에서는 먼저 CSMC 구조에서 메모리 캐쉬 일관성 유지 기법을 설명한 후 이를 바탕으로 CSDC 구조에서의 디스크 캐쉬 일관성 유지 기법을 설명하도록 한다.

3.4.1 메모리 캐쉬 일관성

2.1절에서 설명하였듯이 CSMC 구조에서 데이타의 읽기 프로토콜은 클라이언트 메모리 캐쉬, 서버 메모리 버퍼, 그리고 서버 디스크의 순서로 데이타를 검색하는 것이다. 데이타의 변경 프로토콜은 약간 더 복잡하다. 클라이언트 트랜잭션은 클라이언트 사이트에서 캐쉬된 데이타를 변경한다. 변경할 데이타가 캐쉬에 없으면 물론 먼저 서버에 요구하여 읽어 온 후에 변경한다. 트랜잭션이 승인(commit)될 때 클라이언트 DBMS는 변경된 데이타 및 해당 로그를 모두 서버로 보내고 2단계 승인 프로토콜 등을 이용하여 서버와 해당 클라이언트 간에 승인을 완료한다.

이상의 읽기/변경 프로토콜은 로킹을 생략하고 기술한 것이다. 클라이언트 트랜잭션은 데이타를 읽거나 변경하는 데 적절한 록을 서버로부터 획득한 후 해당 작업을 수행해야 한다. 따라서 서버 DBMS가 록 관리자(lock manager)의 역할을 수행한다. 이때 메모리 캐쉬에 캐쉬되는 것은 비단 서버 데이타베이스의 데이타에 국한되어야 하는 것은 아니다. 데이타와 더불어 그 데이타에 대한 록도 캐쉬할 수 있다.

데이타 캐쉬의 주된 이유는 서버로 데이타 요구 메시지를 전송하는 것을 포함하는 네트워크 접근 부담을 줄이고 아울러 원하는 데이타가 서버의 메모리에 없을 경우 수행해야 되는 디스크 I/O를 피하기 위한 것이다. 그런데 록의 캐쉬 없이 데이타만 캐쉬하는 경우에는 클라이언트 트랜잭션이 캐쉬된 데이타를 재사용하고자 할 때 그 데이타의 정확성(validity)을 매번 서버를 통하여 점검해야 할 것이다. 이 점검은 서버로 캐쉬된 데이타의 버전 번호를 포함하는 메시지를 보내고 서버는 해당 데이타의 버전 번호와 비교하여 정확성 여부를 판단한 후 캐쉬된 데이타가 캐쉬 이후 변경되었을 경우 최신의 데이타를 전송한다. 이 점검 과정은 달리 말하면 최신의 데이타 및 그에 대한 록을 요청하는 과정이라 할 수 있는데, 비록 클라이언트 메모리 캐쉬에 캐쉬된 데이타가 아직 최신의 값이라고 판정되어 서버로부터 그 데이타를 다시 전송받지 않는다 할지라도 메시지 전송 부담은 피할 수 없다.

데이타와 더불어 록을 함께 캐쉬하여 메모리 캐쉬에 존재하는 데이타의 정확성을 항상 보장할 수 있는 방법이 있다. 이를 록 회수(Callback Locking) 기법[17]이라 한다. 이 기법을 사용하면 클라이언트 트랜잭션은 서버와의 접촉 없이 캐쉬된 데이타를 캐쉬된 록 모드에 따라 바로 접근할 수 있다. 그러나 만약 읽기를 위한 공유 록(shared lock)과 함께 캐쉬된 데이타를 클라이언트 트랜잭션이 변경하고자 할 때는 서버에 전용 록(exclusive lock)을 요청해야 한다. 이 경우 서버는 해당 데이타가 캐쉬되어 있는 다른 모든 클라이언트 사이트에 메시지를 보내어 캐쉬된 록을 회수(callback)한다. 각 클라이언트는 록 회수가 요청된 데이타를 사용 중이 아니면 곧바로 회수에 응하고 사용 중이면 사용이 끝난 후 회수에 응한다. 회수에 응한다는 것은 해당 데이타를 캐쉬에서 제외시키는 것을 의미한다. 서버는 관련된 모든 클라이언트로부터 회수에 응한다는 메시지를 받으면 전용 록을 요청한 클라이언트로 하여금 록을 캐쉬하도록 허용한다. 록

7) 이 버전 번호의 예를 들면 페이지 서버 구조의 경우 ARIES가 사용하는 페이지 로그 일련 번호(PageLSN: page log sequence number) [Moha 92]가 있다.

회수 기법에는 읽기 용 록만 캐쉬하는 읽기 록 회수 기법(Callback Read Locking)[17]과 읽기 및 변경 용 록을 캐쉬하는 쓰기 록 회수 기법(Callback Write Locking)[8]이 있는데 주로 전자가 사용된다.

메모리 캐쉬의 일관성을 유지하는 기법을 유형별로 분류해보자. 위에서 설명하였듯이 데이타의 캐쉬와 더불어 록도 함께 캐쉬하는지의 여부에 따라 캐쉬에 존재하는 데이타의 정확성을 서버에 확인해야 하는지의 여부가 결정된다. 록을 캐쉬하면 이 확인 과정이 필요 없고, 록을 캐쉬하지 않으면 접근할 때마다 확인해야 한다. [8]에서는 전자의 방식을 예방(avoidance) 기법, 후자의 방식을 점검(detection 또는 check on access) 기법이라 불렀다. 예방 기법은 캐쉬되어 있는 데이타 중 부정확한 데이타가 생기는 것을 사전에 예방한다는 것이고, 점검 기법은 그러한 보장이 없으므로 매번 접근할 때마다 서버를 통하여 정확성 여부를 점검해야 한다는 것이다.

캐쉬 일관성 유지 기법의 두번째 분류 기준은 상기의 점검 기법을 수행하는 방법이다. 이에는 비관적(pessimistic) 기법과 낙관적(optimistic) 기법이 있다. 전자는 데이타를 접근하기 전에 반드시 서버에 접근하여 적절한 록을 얻는 것이고 후자는 캐쉬된 데이타가 아직 정확한 것이며 원하는 록도 획득하게 될 것이라는 가정하에 데이타 접근을 일단 시작한 후 점검 과정을 뒤로 미루는 것이다.

캐쉬 일관성 유지 기법의 세번째 분류 기준은 클라이언트 트랜잭션이 변경한 데이타가 승인시 서버에 전송될 때 서버가 다른 클라이언트에 캐쉬된 그 데이타 사본들에 대하여 어떤 조치를 취하는가 하는 것이다. 이에는 아무런 조치를 취하지 않는 무조치(no action) 기법 외에 무효화(invalidation) 기법과 변경 전파(propagation) 기법이 있다[12]. 무효화 기법은 다른 클라이언트들로 하여금 자신의 메모리 캐쉬에서 해당 데이타를 삭제하도록 하는 것이고, 변경 전파 기법은 변경 내용을 전달하여 서버를 비롯하여 그 데이타를 저장하고 있는 모든 클라이언트가 최신의 정확한 값을 갖도록 하는 것이다. 무효화 기법은 특히 대상 클라이언트의 수가 많을 경우 변경

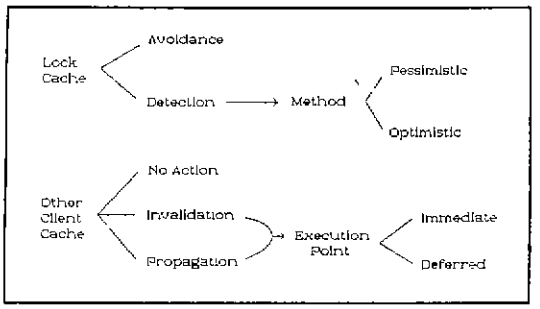


그림 5 캐쉬 일관성 유지 기법의 분류 기준

전파 기법이 지불해야 하는 높은 비용을 없애는 장점 대신 전체 시스템으로 볼 때 데이타 캐쉬가 줄어드는 단점을 갖는다. 변경 전파 기법의 장단점은 그 반대다.

캐쉬 일관성 유지 기법의 네번째 분류 기준은 무효화 및 전파 기법의 수행 시기이다. 즉, 클라이언트 트랜잭션이 승인될 때 즉시 무효화 또는 변경 전파가 일어나느냐 아니면 그것이 지연될 수 있느냐 하는 구분이다. 전자를 즉시 수행(immediate) 기법, 후자를 지연 수행(deferred) 기법이라 한다[2].

이상의 네가지 분류 기준을 그림 5에 나타내었다. 최근 제시된 캐쉬 일관성 기법들을 살펴보고 이들을 상기의 분류 기준으로 나누어 보도록 하자. [12]에서는 다음 다섯가지 캐쉬 일관성 유지 알고리즘을 제시하고 성능을 비교하였다:

(1) Basic 2 Phase Locking (B2PL)
(2) Caching 2 Phase Locking (C2PL)
(3) Optimistic 2 Phase Locking-Invalidation (O2PL-I)
(4) Optimistic 2 Phase Locking-Propagation (O2PL-P)
(5) Optimistic 2 Phase Locking-Dynamic (O2PL-D)[8]

이들 알고리즘은 분산 데이타베이스에서 중복 저장된 데이타 사본들(replica)의 일관성을 유지하는 기법인 주 사본 로킹(primary copy locking)과 read-one/write-all 프로토콜에 기반을 두고 고안된 것들이다. 즉, (1)과 (2)는 서버 사이트를 주 사본 사이트로 간주하는 주 사본 로킹 그리고

8) 이는 시스템 파라미터에 따라 O2PL-I와 O2PL-P를 혼용하는 방법이다.

(3)-(5)는 read-one/write-all 프로토콜을 따른 것이다. B2PL은 CSNC 구조에서 적용되는 방법이고 나머지 방법들은 모두 CSMC 구조에 적용되는 것들로서 이들을 분류해보면 다음과 같다:

(1) C2PL: 점검, 비관적, 무조치
(2) O2PL-I: 점검, 낙관적, 무효화, 즉시 수행
(3) O2PL-P: 점검, 낙관적, 전파, 즉시 수행
(4) O2PL-D: 점검, 낙관적, 무효화 또는 전파, 즉시 수행

[17]에서는 다음 다섯가지 캐쉬 일관성 유지 알고리즘을 제시하고 성능을 비교하였다:

(1) 2 Phase Locking (2PL)
(2) Certification (C)
(3) Callback Locking (CBL)
(4) No Wait Locking (NWL)
(5) No Wait Locking with Notification (NWL-N)

2PL은 [12]의 C2PL에 해당한다. C는 전통적인 낙관적 동시성 제어 기법을 적용한 것으로서 타당성 점검(validation) 단계를 서버 사이트에서 수행한다. CBL은 읽기용 록(read lock)만을 캐쉬 대상으로 한다. NWL은 2PL을 수정한 것으로서 캐쉬된 데이타의 정확성 점검을 서버에 의뢰한 결과를 기다리지 않고 캐쉬된 데이타를 사용하는 낙관적 방법이다. 이미 변경된 데이타를 접근하였을 경우 서버가 이 사실을 알려줘 트랜잭션을 철회시킨다. NWL-N은 NWL의 트랜잭션 철회율을 줄이기 위하여 서버가 전파 기법을 사용하는 방법이다. 이들 방법들은 모두 CSMC 구조에 적용되는 것들로서 이들을 분류해보면 다음과 같다:

(1) 2PL: 점검, 비관적, 무조치
(2) C: 점검, 낙관적, 무조치
(3) CBL: 예방, 무효화, 즉시 수행
(4) NWL: 점검, 낙관적, 무조치
(5) NWL-N: 점검, 낙관적, 전파, 즉시 또는 지연 수행

3.4.2 디스크 캐쉬 일관성

데이타 전송을 사용하는 CSDC 구조에서 디스크 캐쉬 일관성 유지는 CSMC 구조에서 메모리 캐쉬 일관성을 유지하는 기법을 원용하면 된다. 단, CSDC 구조에서 디스크 캐쉬의 일관성을 유지하는 기법 중 점검 기법의 중요성이 CSMC 구조의 경우에 비하여 상대적으로 높아지는 특성을 고려해야 한다[8]. 이는 메모리 캐쉬에 비하여 훨씬 용량이 큰 디스크 캐쉬의 일관성을 유지하기 위하여 예방 기법을 사용할 경우 디스크에 캐쉬된 데이타의 무효화 또는 변경 전파를 위하여 디스크 I/O를 수행해야 하는 부담이 커지기 때문이다[9].

질의 전송을 사용하는 CSDC 구조에서 디스크 캐쉬 일관성 유지는 서버 데이타베이스의 변경 사항 중 캐쉬된 질의 결과에 영향을 미치는 부분만을 클라이언트 사이트로 전송하여 질의 결과를 누증 갱신하는 ADMS± [2]의 기법이 질의 결과를 서버에서 처음부터 다시 생성하여 클라이언트로 캐쉬하는 기법에 비하여 효율적이다[6, 7].

CSDC 구조에서 디스크 캐쉬 일관성 유지를 위하여 반드시 고려되어야 할 것은 클라이언트 사이트가 고장이나 서버와의 접속 종료로 인하여 오프 라인 상태가 되었다가 다시 온 라인 상태로 돌아왔을 때 그동안 서버에서 수행된 데이타 변경을 디스크에 캐쉬된 데이타에 반영하는 기법이다.

데이타 전송을 사용하는 CSDC 구조에서는 만약 디스크 캐쉬 일관성 유지를 위하여 점검 기법을 사용하고 있으면 클라이언트 머쉰이 온라인 상태로 돌아온 직후 별다른 재시작 조치(restart procedure)를 필요로 하지 않는다. 그러나 반대로 예방 기법을 사용하고 있는 경우라면 누증 갱신, 무효화, 또는 변경 전파 등의 기법을 사용하여 디스크 캐쉬에서 부정확한 데이타를 모두 최신의 값으로 갱신해 주어야 한다[8]. 질의 전송을 사용하는 CSDC 구조에서는 ADMS± 에서처럼 누증 갱신이라는 점검 기법을 사용하는 것이 효율적이므로 그 경우 클라이언트 머쉰이 온라인 상태로 돌아온 직후 별다른 재시작 조치를 필요로 하지 않는다[2].

3.5 캐쉬 관리

9) 메모리 캐쉬의 데이타를 무효화 또는 변경하는 데는 디스크 I/O가 필요없다.

캐쉬 관리란 클라이언트-서버 DBMS에서 데이타 캐쉬를 수행 및 이용하는 정책 및 기법을 총칭하는 것이다. 3.4절에서 기술한 데이타 읽기 및 변경 프로토콜, 트랜잭션 승인 프로토콜, 동시성 제어 기법, 그리고 캐쉬 일관성 유지 기법들이 이에 포함된다. 본 절에서는 3.4절에서 언급된 기본적인 캐쉬 관리 기법 외의 것들을 알아보도록 한다.

[13]에서는 전역 메모리 관리 (global memory management) 기법을 제안하였다[10]. CSMC 구조의 경우 전체 시스템을 통하여 데이타는 서버의 디스크, 서버의 메모리, 그리고 각 클라이언트의 메모리에 분산되는데 이들 기억 장치 전부를 전체 시스템의 논리적인 전역 메모리로 관리함으로써 시스템 성능을 높이고자 하였다. 즉, 각 클라이언트의 트랜잭션이 접근할 수 있는 데이타 소스는 자신의 클라이언트 메모리 캐쉬와 서버의 메모리 및 디스크 뿐만 아니라 원격 클라이언트의 메모리 캐쉬까지 포함하게 되는 것이다. 이는 일반적으로 각 클라이언트의 트랜잭션 관점에서 자신의 메모리 캐쉬와 서버의 데이타만을 접근 대상으로 하는 전통적인 기법에 비하여 캐쉬의 용량을 확대하는 것으로서 캐쉬 미스를 줄이고 서버 디스크 I/O를 줄이는 효과를 얻으려는 기법이다.

[8]에서는 CSDC 구조에서의 캐쉬 관리 기법을 제안하였다. CSDC 구조는 메모리 캐쉬와 디스크 캐쉬가 모두 존재하므로 2.1절에서 설명한 것처럼 기억장치 계층 구조가 그림 3(a)와 (b) 두가지가 존재한다. 그리고 3.4.2절에서 설명한 것처럼 CSMC 구조에 비하여 예방 기법 못지 않게 점검 기법의 중요성이 높아진다.

이와 같이 두가지 기억장치 계층과 점검 및 예방의 두가지 디스크 캐쉬 일관성 유지 기법을 조합하면 다음과 같이 총 네가지의 캐쉬 관리 기법을 도출할 수 있다. 단, 이들 기법들은 모두 클라이언트의 메모리 캐쉬 일관성 유지를 위하여는 가장 보편적인 예방 기법인 읽기 록 회수 기법을 사용한다.

(1) LD/A: Local Disk First/Avoidance
(2) SM/A: Server Memory First/Avoidance
(3) LD/D: Local Disk First/Detection
(4) SM/D: Server Memory First/Detection

3.4절에서 기술된 캐쉬 관리의 기본적인 가정은 서버 데이타베이스는 항상 최신의 변경 내용을 저장하고 있다는 것이다. 이는 클라이언트 트랜잭션이 승인될 때 수정된 데이타를 서버 사이트로 전송하는 승인 프로토콜을 준수함으로써 보장된다. 그러나 이러한 요건은 서버에 부담으로 작용할 뿐만 아니라 여러 클라이언트 트랜잭션들이 서버 접근을 경쟁하는 정도를 높이게 된다. [8]에서는 상기의 요건을 완화하여 트랜잭션 승인시 수정된 데이타를 클라이언트가 서버로 전송하지 않고 후에 그 데이타에 대한 접근 요구가 다른 클라이언트로부터 있을 경우 서버의 요청에 따라 서버로 전송하도록 하는 기법을 제안하였다. 트랜잭션 승인시 수정된 데이타를 서버로 전송하지 않으면 데이타에 따라 가장 최신의 내용을 서버가 아닌 특정 클라이언트가 보관하게 되므로 그 클라이언트 사이트가 오프 라인 상태에 있는 경우 해당 데이타의 최신 내용을 접근할 수 없다는 문제가 생길 수 있다. 이 문제는 쓰기 록 회수 기법과 WAL 프로토콜을 통하여 클라이언트 트랜잭션이 수정한 데이타에 대한 로그를 서버에 저장하는 기법으로 해결한다.

4. 결 론

본 논문에서는 클라이언트-서버 DBMS에 관한 최근의 연구 동향을 조사하여 소개하였다. 클라이언트-서버 DBMS는 90년대의 핵심 데이타베이스 기술의 하나로서 이에 대한 80년대 중반의 초기 연구들 이후 90년대에 접어들어 많은 연구가 축적되어 가고 있다. 이들 연구는 주로 다수의 워크스테이션들이 LAN으로 연결된 클라이언트-서버 환경에서 클라이언트 캐쉬의 일관성 유지 기법, 캐쉬 관리 기법, 다양한 클라이언트-서버 DBMS 구조들의 성능 비교, 트랜잭션 관리 기법 등에 관해 수행되고 있다. 90년대에 들어 그동안 가장 활발한 연구 대상이 되어온 클라이언트-서버 DBMS 구조는 클라이언트 사이트에 디스크 캐쉬는 존재하지 않고 메모리 캐쉬만 존

10) 전역 메모리에서 메모리는 주기억 장치 뿐만 아니라 클라이언트 및 서버의 디스크도 함께 포함한다.

제하는 모델이었으나 앞으로 90년대 중반부터는 디스크 캐쉬가 존재하는 클라이언트를 지원하는 클라이언트-서버 DBMS 구조가 연구자들로부터 많은 주목을 받을 것으로 예상된다.

참고문헌

[1] R. Hagman and D. Ferrari, "Performance Analysis of Several Back-End Database Architectures," ACM TODS, **11**, 1, Mar. 1986, pp. 1~26.

[2] N. Roussopoulos and H. Kang, "Principles and Techniques in the Design of ADMS+-," IEEE Computer, **19**, 12, Dec. 1986, pp. 19~25.

[3] K. Kuspert *et al.*, "Cooperative Object Buffer Management in Advanced Information Management Prototype," Proc. VLDB, 1987, pp. 483~492.

[4] The Committee for Advanced DBMS Function, "Third-Generation Database System Manifesto," ACM SIGMOD Record, **19**(3), Sep. 1990, pp. 31~44.

[5] W. Rubenstein *et. al.*, "Benchmarking Simple Database Operations," Proc. ACM SIGMOD, 1987, pp. 387~394.

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

[7] A. Delis and N. Roussopoulos, "Performance Comparison of Three Mordern DBMS Architectures," IEEE TSE, **19**, 12, Feb. 1993, pp. 120~138.

[8] M. Franklin *et. al.*, "Local Disk Caching for Client-Server Database Systems," Proc. VLDB, 1993, pp. 641~654.

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

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

[11] M. Stonebraker, "Architectures of Future Database Systems," IEEE Data Engg. Bulletin, **13**(4), Dec. 1990, pp. 18~23.

[12] M. Carey *et. al.*, "Data Caching Tradeoffs in Client-Server DBMS Architectures," Proc. ACM SIGMOD, 1991, pp. 357~366.

[13] M. Franklin, *et. al.*, "Global Memory Management in Client-Server DBMS Architectures," Proc. VLDB, 1992, pp. 596~609.

[14] N. Roussopoulos, "View Indexing in Relational Databases," ACM Trans. on Database Systems, 7, 2, Jun. 1982, pp. 258~290.

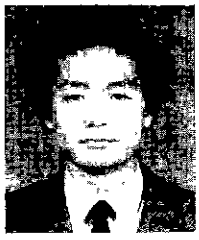
[15] N. Roussopoulos, "An Incremental Access Method for ViewCache: Concept, Algorithms, and Cost Analysis," ACM Trans. on Database Systems, **16**, 3, Sep. 1991, pp. 535~563.

[16] K. Wilkinson and M. Neimat, "Maintaining Consistency of Client-Cached Data," Proc. VLDB, 1990, pp. 122~133.

[17] Y. Wang and L. Rowe, "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture," Proc. ACM SIGMOD, 1991, pp. 367~376.

[18] C. Mohan *et al*, "ARIES: A Transaction Recovery Method Suppoting Fine Granularity Locking and Partical Rollbacks Using Write-Ahead Logging," ACM TODS 17, 1, Mar. 1992, pp. 94~162.

강 현 철



1983 서울대학교 컴퓨터공학과 졸업(공학사)
1985 U of Marland at College Park, Computer Science(M S)
1987 U of Marland at College park, Computer Science(Ph. D.)
1988 ~현재 중앙대학교 컴퓨터공학과 부교수
관심 분야 : 분산 데이타베이스, 클라이언트-서버 DBMS, DBMS 저장시스템