공간 데이타베이스에서 클라이언트 뷰의 일관성 제어 기법

(The Consistency Control of Client Views in Spatial Database)

임 덕 성 † 반 재 훈 † 문 상 호 †† 홍 봉 희 †††
(Duk-Sung Lim)(Chae-Hoon Ban)(Sang-Ho Moon)(Bong-Hee Hong)

요 약 클라이언트/서버 환경에서 서버의 공간 데이타를 검색, 접근하기 위한 공간 질의는 대용량의 공간 객체에 대해 복잡한 공간 연산을 수행한다. 따라서 클라이언트가 자주 이용하는 공간 질의를 매번 서버에서 처리하는 경우에 서버의 부하가 증가하며 질의 응답시간도 길어지게 된다. 그러므로 이러한 공간 질의를 뷰로 정의하고 클라이언트에 실체화하면 질의를 효율적으로 처리할 수 있다. 이 경우에 뷰를 유도한 서버의 소스 객체의 변경에 따라 클라이언트의 실체화된 뷰 객체의 일관성을 유지해야 한다. 본 논문에서는 클라이언트/서버환경에서 질의 수행 속도를 향상시키기 위해 실체화된 공간뷰 개념을 적용하여 클라이언트 뷰를 정의하고 유형을 분류한다. 그리고 실체화된 클라이언트 뷰의 일관성 제어를 위해 재수행 방법에 비해 변경 속도가 빠른 점진적 변경 방법을 이용한다. 이를 위해 일관성 제어시 필요한 추가정보를 기술하고, 추가정보를 이용한 일관성 제어 알고리즘을 제시한다. 그리고, 제시한 실체화 방법과 일관성 제어 알고리즘의 실현을 위하여 객체지향 GIS시스템인 고딕(GOTHIC)에서 설계 및 구현하고 성능을 평가한다.

Abstract Spatial queries which have to access and retrieve the server's spatial data in a client/server environment may include complex spatial operations which are performed on a number of spatial objects. If spatial queries frequently used by clients should always be executed in a server, it takes a long time to receive query results in a client. To improve the speed of processing the same spatial queries issued by a client, we define the spatial queries as views and materialize the query results in a client side. The overhead of client view materialization is that materialized spatial views in a client side should be maintained in a consistent state in response to the update of source spatial objects. This paper defines client views based on the concept of materialized spatial views to improve the speed of client's query processing. To control the consistency of materialized client views, this paper applies an incremental update method that is much faster than the existing re-materialization method. This paper proposes the consistency control algorithms using the additional information required to control the consistency of materialized view objects. The implementation of the client view system in GOTHIC shows client view materialization is better than the other strategies.

1. 서 론

최근 LAN, WAN 환경에서 네트워크 속도의 향상과 빠른 처리 속도를 가진 PC의 등장으로 클라이언트/서버 컴퓨팅 환경이 보편화되고 있다[3]. 특히 지리정보시스템(GIS)에서는 대용량의 공간 데이타를 취급하므로 서버에 공간 데이타를 저장 및 관리하고 다수의 클라이언트들이 검색하는 클라이언트/서버 컴퓨팅 환경이 효율적이다. 그러나 그림 1과 같이 다수의 클라이언트가 하나의 서버에 접근할 경우에는 클라이언트의 질의 횟수가 증가함에 따라 서버의 부하가 증가하고 응답 시간이 길어지는 문제점이 발생한다. 따라서 클라이언트가 서버의 공간 데이타를 효율적으로 접근하는 방법의 제시가 필요하다.

† 학생회원 : 부산대학교 컴퓨터공학과
dsleem@hyowon.pusan.ac.kr
jhpan@hyowon.pusan.ac.kr
†† 정 회 원 : 위덕대학교 컴퓨터공학과 교수
shmoon@mail.uiduk.ac.kr
††† 종신회원 : 부산대학교 컴퓨터공학과 교수
bhhong@hyowon.pusan.ac.kr
논문접수 : 2000년 5월 19일
심사완료 : 2000년 12월 4일

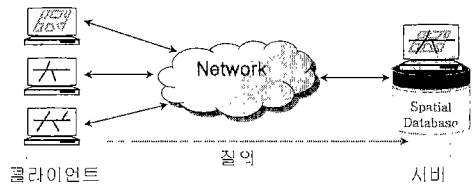


그림 1 클라이언트/서버 환경

클라이언트/서버 환경에서 질의 수행 속도를 향상시키기 위해 캐쉬 또는 중복 질의 방법이 제시되었다[2,3]. 일반적인 캐쉬의 경우에는 서버의 데이타를 페이지 또는 객체 단위로 클라이언트에 저장하여 검색 및 조작 속도를 향상시킨다. 그러나 프로그램이 종료되면 캐쉬에 저장된 데이타의 영속성을 보장하지 못한다. 중복 질의는 질의 결과를 저장하여 동일한 질의에 대해 저장된 질의 결과를 이용하므로 질의 수행 속도가 빠르지만 서버의 데이타가 변경되면 클라이언트에 저장된 데이타의 일관성을 보장하지 못하는 단점이 있다.

공간 데이타베이스에서 뷰는 다양한 사용자의 관점을 지원하기 위한 질의의 다른 형태이다[17]. 뷰의 개념을 확장한 클라이언트 뷰는 클라이언트/서버 GIS환경에서 효율적인 공간 질의 처리를 위하여 자주 사용하는 질의를 뷰로 정의한 것이다. 이러한 클라이언트 뷰는 빠른 질의 처리를 수행하기 위해서 실체화하고 클라이언트에 저장하는 것이 필요하다[2,5,22].

실체화된 뷰는 뷰를 유도한 서버의 소스 객체의 변경에 따라 일관성을 유지해야하는 문제가 있다. 예를 들어 그림 2와 같이 시간 t_0에 서버 객체 A와 B가 중첩(overlap)되는 영역을 클라이언트 뷰로 정의하여 실체화한 객체를 CVO(client view object)로 가정하자. 그리고 시간 t_1에 뷰를 유도한 서버 객체 A가 A′으로 변경되면 클라이언트에 저장된 뷰 객체의 일관성이 파괴된다. 이 경우에는 서버 객체 A의 변경에 따라 클라이언트에 저장된 뷰 객체 CVO를 CVO′으로 변경해야 한다.

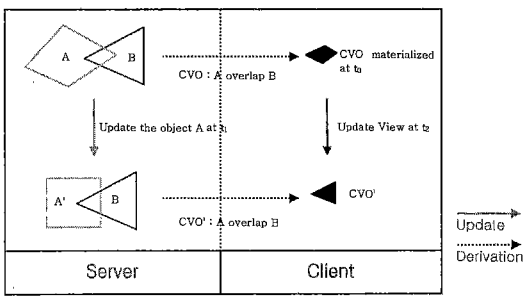


그림 2 실체화된 클라이언트 뷰의 일관성 제어

기존의 실체화된 뷰의 일관성 제어 방법으로는 재수행과 점진적 변경 방법이 있다[8,17]. 재수행 방법은 소스 객체가 변경되었을 때, 이미 실체화된 뷰 객체를 모두 삭제하고 다시 뷰 객체를 생성하는 방법이다. 반면에 점진적 변경 방법은 소스 객체가 변경될 때, 이로부터 유도된 뷰 객체만을 변경한다. 따라서 점진적 변경 방법은 재수행 방법에 비해 변경 속도가 빠르고 효율적이지만 변경 알고리즘이 복잡하다.

본 논문에서는 먼저 클라이언트/서버환경에서 질의 수행 속도를 향상시키기 위해 실체화된 공간뷰 개념을 확장하여 클라이언트 뷰를 정의하고 유형을 분류한다. 클라이언트 뷰는 실체화과정에서 공간관련성연산자(spatial relationship operator)를 포함하는 selection과 기하생성연산자(geometry generator operator)를 포함하는 projection을 누가 담당하느냐에 따라 분류된다. 그리고 클라이언트 뷰의 실체화 방법은 뷰 객체가 소스 객체의 모든 속성을 복사하여 저장하는 값 복사 방법을 사용한다. 특히 기하생성연산자를 포함하는 클라이언트 뷰의 경우는 뷰 객체가 기하 데이타를 포함하기 때문에 기하 데이타에 대한 접근 속도를 향상시킬 수 있다.

본 논문에서는 클라이언트 뷰의 일관성 제어를 위해 재수행 방법에 비해 변경 속도가 빠른 점진적 변경 방법을 이용한다. 이를 위해 일관성 제어시 필요한 추가정보를 기술하고, 일관성 제어 알고리즘을 제시한다. 그리고, 제시한 실체화 방법과 일관성 제어 알고리즘의 실현을 위하여 객체지향 GIS시스템인 고딕(GOTHIC)에서 설계 및 구현하고 성능 평가를 수행한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 관련 연구를 설명하고, 3장에서는 클라이언트 뷰의 개념과 유형에 대하여 설명한다. 4장에서는 클라이언트 뷰의 실체화 및 일관성 제어 방법과 세부 알고리즘을 기술한다. 5장에서는 실험을 통해 구현된 클라이언트 뷰 시스템의 성능을 평가한다. 마지막으로 6장에서는 결론 및 향후 연구를 기술한다.

2. 관련연구

실체화된 뷰의 일관성 제어에 관한 연구는 실체화 방법에 따라 크게 식별자-유지 방법과 값 복사방법에 관한 연구로 나뉘어진다. 관계형 데이타베이스에서는 일반적으로 기본테이블의 레코드 값을 복사하여 뷰를 실체화한다. 그러나 [1,2,5]에서는 튜플 식별자를 저장하여 뷰를 실체화하고 기본 테이블의 변경에 따른 일관성 제어를 위한 점진적 변경 방법이 연구되었다. [1]에서는 서버의 데이타를 다수의 워크스테이션에서 처리하기 위

한 접근 모델을 제시하였다. 즉 지역(local) 데이타베이스를 포함하는 워크스테이션에서 질의 수행 속도의 향상을 위하여 서버에서 처리된 질의 결과를 지역 데이타베이스에 뷰-캐쉬(view-cache)의 형태로 저장하였다. 이때 튜플 식별자만을 실체화시킴으로써 뷰의 관리 및 유지를 효율적으로 처리하였다. 특히 뷰-캐쉬에 대한 질의를 수행하는 경우에 접근 경로를 전역 경로(global subpath)와 지역 경로(local subpath)로 나누고 전역 경로를 처리하는 경우에는 질의 수행 속도 향상을 위해 동일 바인딩(identity binding)과 유도 바인딩(derivation binding)을 적용하여 지역 경로로 바꾸어 처리한다. 그리고 실체화된 뷰 객체의 일관성 제어를 위해 점진적 변경 방법을 제시하였다. [2]에서는 분산 환경에서 뷰-캐쉬의 일관성 제어 시점에 따라 즉시 변경 전파(immediate update broadcast), 주기적 변경 전파(periodic update broadcast), 사건-트리거 변경(event-triggered update), 차후 변경(deferred update)의 4가지 전략을 비교하고, 일관성 제어시 질의 최적화를 위해 small-join방법을 제시하였다. 그러나, 공간 데이타베이스에서 공간뷰는 사용자의 관점에 따라 서로 다른 기하 객체의 표현을 위해 새로운 기하 객체를 생성하는 것이 필요하다. 따라서 위 논문들에서 제시된 방법을 적용할 경우 튜플 식별자만을 저장하기 때문에 뷰에 대한 검색이 있는 경우 소스 데이타를 클라이언트로 가져와야 하는 전송 비용과 매번 소스 데이타를 이용하여 기하 데이타를 재계산해야 하는 단점이 있다.

객체지향 데이타베이스에서 실체화된 뷰의 일관성 제어 방법이 연구되었다. MultiView에서 실체화된 뷰 객체는 중복 저장을 피하기 위하여 Object-Slicing기법을 적용하여 소스 객체에 대한 식별자를 유지한다. 그리고 실체화된 뷰 객체를 변경하기 위하여 점진적 변경 알고리즘을 제시하였다[13]. 이 알고리즘에서는 등록 서비스(registration-service)를 이용하여 변경된 소스 객체와 관련된 뷰 클래스를 선택한다. 즉 유도 계층 구조(derivation hierarchy)를 별도로 구성하고, 이 정보를 이용하여 뷰 클래스 정의에 있는 Where 조건문을 검사하는 횟수를 줄였다. 그러나 MultiView에서는 식별자-유지 방법을 이용하므로 뷰 객체에 대한 질의 수행시 관계형 데이타베이스와 마찬가지로 기하 데이타를 재계산 해야하므로 질의 수행시간이 길어지는 단점이 있다.

공간 데이타베이스에서 뷰에 대한 질의 수행 속도를 향상시키기 위해 값 복사 방법을 이용한 실체화 방법이 제시되었다[17]. 이 논문에서는 전통적인 뷰를 공간 데이타베이스에 적용하기 위해 공간뷰의 개념을 정의하고 실체화 및 일관성 유지를 위한 방법을 제시하였다. 특히 일관성 유지를 재계산과 점진적 변경으로 나누고 성능의 향상을 위해 점진적 변경 알고리즘을 제시하였다. 이 방법을 클라이언트/서버 환경에 적용할 경우에 다음과 같은 문제점이 발생한다. 첫째, 다수의 클라이언트들이 서버에 실체화된 공간뷰에 접근하는 경우에 서버에 많은 부하가 걸리게 된다. 따라서 클라이언트/서버 환경에서의 공간뷰 설계시 서버의 부하를 줄이는 방법에 대한 연구가 필요하다. 둘째, 공간뷰를 클라이언트에서 정의 및 실체화하기 때문에 클라이언트와 서버간의 작업 배분이 필요하다. 마지막으로 클라이언트에 실체화된 공간뷰에 대한 새로운 일관성 유지 방법이 필요하다. 즉 소스 객체는 서버에, 뷰 객체는 클라이언트에 저장되어 있으므로 기존 방법과 다른 일관성 제어 방법이 필요하다.

3. 클라이언트 뷰의 개념 및 유형

이 장에서는 공간뷰의 기본 개념과 클라이언트/서버 환경에서 설계 가능한 공간뷰를 분류하고 각각을 설명한다.

3.1 용어 정의

클라이언트/서버 환경에서의 공간뷰를 제시하기에 앞서 관련된 용어들을 먼저 정의한다. 이 용어들은 대부분 관련 연구들마다 조금씩 차이가 있으나 대부분이 공통적으로 사용되고 있다[1,11,17].

- 뷰-정의 질의(view-defining query) : 뷰에 대한 정의문으로 뷰 객체를 유도하는데 사용된다. 뷰-정의 질의를 뷰 사상(view mapping) 함수라고도 한다.
- 뷰 클래스(view class) : 뷰-정의 질의에 의해 유도되는 가상 클래스이다. 뷰 클래스는 한 개 이상의 기본 클래스 또는 다른 뷰 클래스로부터 유도된다.
- 뷰 객체(view object) : 뷰-정의 질의에 의해 유도되는 뷰 클래스의 인스턴스로 실제 데이타베이스에는 저장되지 않는다.
- 기본 클래스(base class) : 사용자가 정의한 실제 클래스(real class)로서 데이타베이스에 저장된 인스턴스를 가진다.
- 기본 객체(base object) : 기본 클래스에 대한 인스턴스로 실제 데이타베이스에 저장된다.
- 소스 클래스(source class) : 뷰가 정의될 때 사용되는 클래스로, 뷰-정의 질의의 FROM 절에 지정된다. 기본 클래스 뿐만 아니라 뷰 클래스도 소스 클래스가 될 수 있으며, 하나의 뷰 클래스에 대하여 하나 이상의 소스 클래스가 존재한다. 이 소스 클래스를 다른 관련 연구에서는 루트 클래스(root class)라고도 한다.

• 소스 객체(source object) : 소스 클래스의 인스턴스로 뷰 객체를 유도하기 위해 사용된 객체이다. 기본 객체 뿐만 아니라 뷰 객체가 소스 객체가 될 수 있다.

• 기하-사상 함수(geometry-mapping function) : 공간뷰의 기하데이타를 유도하는데 사용되고, 뷰-정의 질의의 SELECT절에서 이용한다.

• 동일 바인딩(identity binding) : 서버 소스 객체와 클라이언트에 중복 저장된 객체와의 관계로, 클라이언트에서 소스 객체를 동일한 형태로 전송 받는 경우 발생한다.

• 유도 바인딩(derivation binding) : 서버의 소스 객체로부터 유도된 클라이언트의 뷰 객체와의 관계로, 클라이언트에서 소스 객체의 부분 집합을 전송 받는 경우 발생한다.

3.2 공간뷰 개념

관계형 뷰와 객체지향 뷰는 selection(σ)과 projection(π)으로 구성된 뷰-정의 질의를 이용하여 정의된다. 특히 객체지향 데이타베이스에서 뷰 클래스 VC_i의 뷰-정의 질의는 소스 클래스 SC_i들의 카티젼 곱(cartesian product)에 대한 σ와 π로 다음과 같이 정의된다.

$$VC_i = \pi(\sigma(SC_1 \times SC_2 \times \cdots SC_n))$$

공간 데이타베이스에서 공간뷰는 기존 뷰 개념에서 공간 객체에 대한 연산자인 공간 연산자를 필요로 한다. 공간 연산자는 기하연산자(GO:Geometry Operator), 공간관련성연산자 (SRO:Spatial Relationship Operator), 그리고 기하생성연산자(GGO:Geometry Generation Operator)로 분류된다. 세 연산자 중에서 공간뷰에 적용할 수 있는 연산자는 기하-사상 함수에 사용되는 기하생성연산자와 프레디킷에 사용되는 공간관련성연산자이다. 따라서 소스 클래스 SC_i로부터 유도된 공간뷰 클래스 SVC_i는 그림 3과 같이 정의된다.

그림 3에서 뷰-정의 질의의 π는 기하 데이타와 비기하 데이타에 대한 projection으로 구성되고, 기하 데이타에 대한 projection은 기하생성연산자로 정의된다. 그리고 기하 데이타에 대한 selection은 공간관련성연산자로 정의된다.

$$\begin{aligned} SVC_i &= \pi(\sigma(SC_1 \times SC_2 \times \cdots \times SC_n)) \\ &= \pi_{nongeom,geom}(\sigma(SC_1 \times SC_2 \times \cdots \times SC_n)) \\ &= \pi_{nongeom,geom}(\sigma_{nongeom \wedge geom}(SC_1 \times SC_2 \times \cdots \times SC_n)) \\ &= \pi_{nongeom,GGO}(\sigma_{nongeom \wedge geom}(SC_1 \times SC_2 \times \cdots \times SC_n)) \\ &= \pi_{nongeom,GGO}(\sigma_{nongeom \wedge SRO}(SC_1 \times SC_2 \times \cdots \times SC_n)) \end{aligned}$$

SVC_i 공간뷰클래스
SC_i : 소스 클래스
π: projection
σ: selection
geom : 기하속성
nongeom : 비기하속성
GGO : 기하생성연산자
SRO : 공간관련성연산자

그림 3 공간뷰의 정의

3.3 클라이언트 뷰의 정의와 분류

클라이언트 뷰는 클라이언트/서버 GIS환경에서 효율적인 공간 질의 처리를 위하여 자주 사용하는 질의를 뷰로 정의한 것이다. 클라이언트 뷰는 실체화과정에서 공간관련성연산자를 포함하는 selection과 기하생성연산자를 포함하는 projection을 누가 담당하느냐에 따라 표 1과 같이 분류된다.

표 1 클라이언트 뷰의 분류

구분 / 유형	뷰정의문 저장위치	뷰실체화비용 σ	뷰실체화비용 π	뷰실체화비용 저장위치	일관성제어 담당	추가정보 CDR	추가정보 VDR
유형1	S, C	S	S	S	S	S	S
유형2	S, C	S	S	C	S, C	S	S
유형3	C	C	C	C	C	C	C
유형4	C	S	C	C	S, C	S	C

S : 서버
C : 클라이언트

실체화비용은 뷰-정의 질의를 수행하여 질의 결과를 클라이언트로 전송하는 뷰 검색비용과 검색 결과를 뷰 객체로 생성하여 저장하는 비용으로 이루어진다. 먼저 공간뷰 SVC에 대한 검색 비용 $cost(VR(SVC))$는 다음과 같이 정의된다.

$$\begin{aligned} Cost(VR(SVC)) = Cost(\sigma_{nongeom}) + Cost(SRO) + Cost(\pi) \\ + Cost(GGO) + Cost(Trans) \end{aligned}$$

여기서 $cost(SRO)$는 공간관련성연산자를 처리하기 위한 비용, $cost(GGO)$는 기하생성연산자 처리비용, 그리고 $cost(Trans)$는 질의 결과의 전송 비용이다.

클라이언트에서 질의 수행 속도를 향상시키기 위해 뷰를 실체화시킬 경우에는 실체화 비용 $cost(M(SVC))$는 뷰 검색 비용 $cost(VR(SVC))$와 질의 결과를 이용하여 뷰 객체를 생성하는 비용 $cost(Create(SVO))$와 저장하는 비용 $cost(Save(SVO))$가 추가된다.

$$\begin{aligned} Cost(M(SVC)) = Cost(\sigma_{nongeom}) + Cost(SRO) + Cost(\pi) + Cost(GGO) \\ + Cost(Trans) + Cost(Create(SVO)) + Cost(Save(SVO)) \end{aligned}$$

3.3.1 유형1

유형 1의 클라이언트 뷰는 서버가 공간뷰를 실체화하고 일관성을 유지하는 형태로서 뷰의 유도 관계는 그림 4와 같다. 그림 4(a)는 하나의 소스 클래스로부터, 그림 4(b)는 두개의 소스 클래스로부터 유도된 유형 1의 클라이언트 뷰에 대한 유도 관계를 나타낸다.

서버의 소스 객체 SO_1으로부터 유도된 뷰 객체 VO_1은 서버에서 실체화되어 서버 내에서 유도 관계가 성립한다. 반면 클라이언트가 실체화된 뷰를 요청할 경우 서

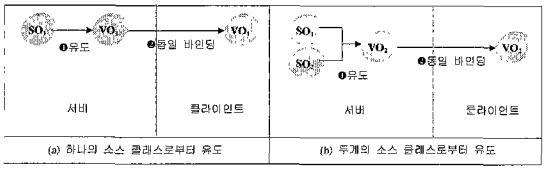


그림 4 뷰 유도관계(유형1)

버는 이미 저장된 실체화된 뷰 객체들을 클라이언트로 보내어 서버의 VO_1과 클라이언트의 VO_1간의 동일 바인딩 관계만을 유지한다. 유형 1의 경우에서 서버의 소스 데이타가 변경될 경우 서버 내에서 뷰의 일관성을 유지하고, 클라이언트에서는 재요청에 의한 방법으로 일관성을 유지하게 된다. 이 경우에 클라이언트는 단순히 새로운 공간뷰를 정의하는 역할을 한다. 서버가 다수의 클라이언트에서 요청한 공간뷰를 실체화하고 일관성을 유지하므로 서버에 많은 부하가 걸리는 단점을 가진다. 이 경우 서버와 클라이언트의 비용 분담은 다음과 같다.

$$Cost_{Server} = Cost(\sigma_{nongeom}) + Cost(SRO) + Cost(\pi) + Cost(GGO) + Cost(Create(SVO)) + Cost(Save(SVO))$$

$$Cost_{Client} = 0$$

서버의 비용은 서버에서 유지되는 실체화된 공간뷰와 동일한 비용을 수반한다. 그리고 전송 비용은 $Cost(Transfer) \propto \sum_{i=1}^{n} SVO_i$와 같다. 즉, 전송비용은 서버에 실체화된 클라이언트 뷰 객체의 크기에 비례한다. 여기서 n은 클라이언트 뷰 객체의 수이다.

3.3.2 유형2

유형 2의 클라이언트 뷰는 서버가 뷰 객체를 생성하고 클라이언트는 뷰 객체를 저장하는 형태로서 뷰의 유도 관계는 그림 5와 같다.

단계1에서 서버의 소스 객체 SO_1으로부터 유도된 뷰 객체 VO_1은 서버에서 실체화되어 서버 내에서 유도 관계가 성립하고, 실체화된 클라이언트 뷰는 클라이언트와 서버사이에서 동일 바인딩관계가 성립하는 유형 1과 동일하다. 그러나 단계2에서 서버에서 실체화된 클라이언트 뷰가 삭제되면서 서버의 소스 객체와 실체화된 클라

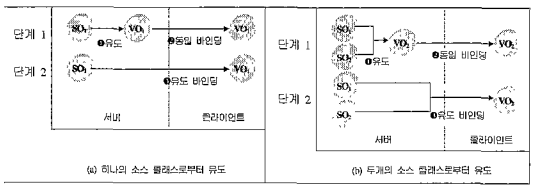


그림 5 뷰 유도관계(유형 2)

이언트 뷰 사이에는 유도 바인딩 관계가 성립한다.

서버의 소스 객체가 변경될 경우 서버는 변경된 객체의 정보를 이용하여 변경된 객체로부터 유도되는 뷰 객체를 클라이언트에 전송하여 일관성을 유지한다. 이 경우 서버와 클라이언트의 비용 분담이 다음과 같다.

$$Cost_{Server} = Cost(\sigma_{nongeom}) + Cost(SRO) + Cost(\pi) + Cost(GGO) + Cost(Create(SVO))$$

$$Cost_{Client} = Cost(Save(SVO))$$

유형 2는 서버가 클라이언트 뷰 객체를 생성하고 생성된 객체를 클라이언트로 전송한 후에 삭제한다. 그리고 클라이언트는 뷰에 대한 질의 수행을 서버에 요청하지 않고 직접 클라이언트에 있는 실체화된 클라이언트 뷰를 이용하여 질의를 수행하므로 질의 처리 속도가 빠르다. 그러나 뷰-정의 질의에 포함된 고비용의 공간관련 연산자 및 기하생성연산자들을 서버에서 처리해야 한다. 전송 비용은 유형 1과 동일하다.

3.3.3 유형3

유형 3의 클라이언트 뷰는 뷰의 실체화 및 일관성 제어를 클라이언트가 담당하는 형태로서 뷰의 유도 관계는 그림 6과 같다.

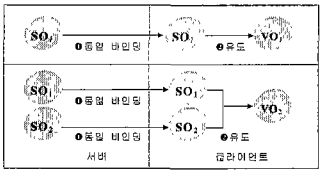


그림 6 뷰 유도관계(유형 3)

클라이언트에서 실체화된 클라이언트 뷰를 생성시키기 위해서는 먼저 소스 객체들을 서버로부터 전송 받는다. 이 경우 서버의 소스 객체 SO_1과 클라이언트로 전송된 소스 객체 SO_1 사이에는 동일 바인딩 관계가 성립한다. 클라이언트는 서버로부터 전송된 소스 객체들로부터 실체화된 뷰 객체 VO_1을 생성하여 유도 바인딩 관계를 생성한다. 유형 3의 경우에서 서버의 소스 객체가 변경되면 서버는 변경된 객체의 정보를 클라이언트에 전송하고, 클라이언트는 변경된 소스 객체를 이용하여 이로부터 유도된 뷰 객체의 일관성을 유지한다. 이 경우 서버와 클라이언트의 비용 분담이 다음과 같다.

$$Cost_{Server} = 0$$

$$Cost_{Client} = Cost(\sigma_{nongeom}) + Cost(SRO) + Cost(\pi) + Cost(GGO) + Cost(Create(SVO)) + Cost(Save(SVO))$$

유형 3은 클라이언트 뷰를 유도하기 위한 소스 클래스의 모든 객체를 클라이언트로 전송해야 한다. 전송 비

용은 $Cost(Transfer) \propto \sum_{i=1}^{m} \sum_{j=1}^{n} SO_i^j$와 같다. 여기서 m은 소스 클래스의 수이고, n은 i번째 소스 클래스의 객체 수이다.

유형 3의 클라이언트 뷰는 클라이언트에서 실체화를 위한 고비용의 공간 연산과 실체화된 클라이언트 뷰의 일관성 유지를 수행한다. 따라서 서버에서 공간 연산을 처리하지 않는 장점을 가지지만 대용량의 소스 객체를 전송하며 클라이언트가 공간 연산, 뷰 객체 생성, 그리고 일관성 유지를 수행해야 하는 단점을 가진다.

3.3.4 유형4

유형 4의 클라이언트 뷰는 뷰의 실체화 및 일관성 제어를 클라이언트와 서버가 분담하는 형태로서 뷰의 유도 관계는 그림 7과 같다.

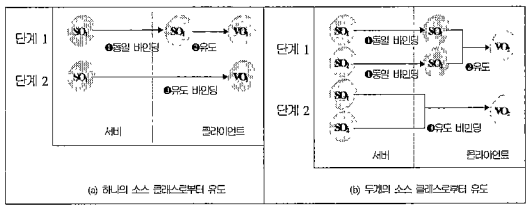


그림 7 뷰 유도관계(유형 4)

그림 7은 서버의 소스 객체 SO_i으로부터 클라이언트에 저장된 실체화된 뷰 객체 VO_i간의 유도 관계를 나타낸다. 유형4는 2단계로 구성된다. 단계1은 유형3과 유사하다. 그러나 동일 바인딩의 대상이 뷰 정의문의 프레디킷에 만족하는 후보 객체이다. 단계2는 클라이언트에 저장된 후보객체들이 삭제되면서 서버의 소스 객체와 실체화된 클라이언트 뷰 사이에는 유도 바인딩 관계가 성립한다. 유형 4의 경우에서 서버의 소스 객체가 변경되면 서버는 변경된 객체의 정보를 이용하여 클라이언트에 전송할 후보 객체를 생성하고, 클라이언트는 후보 객체를 이용하여 이로부터 유도된 실체화된 뷰 객체의 일관성을 유지한다. 즉, 유형 4는 기하생성연산자를 포함한 클라이언트 뷰 객체의 생성과 저장은 클라이언트가, 공간관련성연산자를 포함한 selection은 서버가 담당하는 형태로서 서버와 클라이언트의 비용 분담이 다음과 같다.

$$Cost_{Server} = Cost(\sigma_{non(geom)}) + Cost(SRO)$$

$$Cost_{Client} = Cost(\pi) + Cost(GGO) + Cost(Create(SVO)) + Cost(Save(SVO))$$

그리고 전송 비용은 $Cost(Transfer) \propto \sum_{i=1}^{m} \sum_{j=1}^{n} SO_i^j$와 같다. 여기서 m은 소스 클래스의 수이고, n은 i번째 소스 클래스에서 프레디킷을 만족하는 소스 객체의 수이다.

유형 4는 유형 3과 비교할 경우에 클라이언트의 부하가 줄고 유형 1, 유형 2와 비교할 경우에 서버의 부하가 줄어든다. 본 논문에서는 클라이언트 뷰 객체의 생성을 서버와 클라이언트가 분담하며, 클라이언트 뷰 객체를 클라이언트에 저장하는 유형 4에 대하여 실체화 및 일관성 제어 방법을 제시한다.

4. 클라이언트 뷰의 실체화 및 일관성 유지

4장에서는 클라이언트 뷰의 실체화 작업을 클라이언트와 서버 작업으로 나누고, 실체화된 클라이언트 뷰의 일관성 제어 방법을 설명한다.

4.1 클라이언트 뷰의 실체화

뷰의 실체화 방법에는 식별자-유지 방법과 값-복사에 의한 방법이 있다[17]. 클라이언트/서버 환경에서 식별자-유지 방법을 이용하여 객체에 접근하는 방법은 비효율적이다. 그 이유는 뷰 객체를 유도하는 소스 객체의 식별자만을 유지하므로 뷰에 대한 질의 수행시 서버에 저장된 소스 객체에 접근해야 하기 때문이다. 특히 소스 객체의 기하를 변형하여 뷰를 유도하는 경우에는 뷰에 대한 질의 수행시마다 기하 데이타를 획득하기 위해 공간 연산을 수행하므로 질의 수행시간이 매우 길어진다. 따라서 질의 수행 속도를 향상시키고 네트워크의 부하를 줄이기 위해 클라이언트 뷰의 실체화 방법은 실제 데이타를 저장하는 값-복사 방법을 이용한다.

본 논문에서 제안하는 클라이언트 뷰의 실체화는 서버 작업과 클라이언트 작업으로 분류된다. 서버 작업은 클라이언트에서 요청하는 뷰-정의 질의를 수행하여 질의 결과를 클라이언트에 전송하고 일관성 제어를 위한 추가 정보를 생성하는 것이다. 이 경우에 질의 수행 결과는 뷰-정의 질의를 만족하는 소스 객체(쌍)들의 데이타이다.

클라이언트 작업은 아래와 같이 세단계로 수행된다. 첫째, 사용자의 관점에 따라 정의된 클라이언트 뷰를 실체화하기 위해 서버에 전달할 뷰-정의 질의를 생성한다. 이러한 뷰-정의 질의는 클라이언트에 저장된다. 둘째, 서버로부터 전송된 후보 객체를 이용하여 클라이언트 뷰 객체를 생성하고 저장한다. 마지막으로 소스 객체 변경시 실체화된 클라이언트 뷰 객체의 일관성 제어를 위한 추가 정보를 생성한다. 그림 8은 클라이언트에서 후보객체를 이용하여 클라이언트 뷰 객체와 VDR을 생성하는 세부적인 실체화 알고리즘이다.

실체화 알고리즘은 서버에서 전송된 후보 객체 집합 $\{C_i\}$에서 후보 객체로부터 유도된 클라이언트 뷰 객체의 유무를 VDR정보를 이용하여 검색하기 위해 *Find*

*VDR()*을 호출한다. 검색된 VDR정보가 없다면 새로운 클라이언트 뷰 객체를 생성하고 뷰 유도관련성 정보를 추가한다. 검색된 VDR정보가 존재하는 경우에는 뷰 유도 관련성 중 1:M관계나 N:M관계에 해당하므로 뷰 객체를 생성하지 않고 VDR정보만 수정하게 된다.

```
CVD   : Client View Definiton          t     : Time
{C_i} : Candidate Object Set           CVO   : Client View Object

Procedure Materialize(CVD, t, {C_i})
begin
    for each C_i in {C_i}
    do
        if (FindVDR(C_i))   // 후보객체로 유도된 뷰 객체가 존재할 경우
            UpdateVDR(CVD, TS, VDR_k, C_i);
        else                //후보객체로 유도된 뷰 객체가 없는 경우
            SVO = CreateSVObject((CVD, C_i);
            InsertVDR(CVD ,t , SVO, C_i);
    end for;
end;
```

그림 8 실체화 알고리즘

4.2 클라이언트 뷰의 일관성 제어

서버에서 소스 객체가 변경될 경우에 이로부터 유도된 실체화된 클라이언트 뷰 객체의 일관성 제어가 필요하다. 뷰의 일관성 제어 방법은 소스 객체가 변경되었을 때, 이미 실체화된 뷰 객체를 모두 삭제하고 다시 뷰 객체를 생성하는 방법인 재수행 방법에 비해 변경 속도가 빠른 점진적 변경 방법을 사용한다. 그리고 변경시기에 따른 일관성 제어 방법은 모든 클라이언트가 On-line인 것을 요구하는 즉시 변경 대신에, 뷰에 대한 질의 요구가 발생하는 경우에 일관성을 제어하는 요구시-처리 방법을 사용한다.

4.2.1 추가 정보

클라이언트/서버 환경에서의 클라이언트 뷰의 일관성 제어를 위한 추가 정보로서 유도 관련성(derivation relationship)과 변경 로그(update log)를 이용한다. 유도 관련성은 소스 클래스와 클라이언트 뷰 클래스간의 클래스 유도 관련성(CDR: Class Derivation Relationship)과 소스 객체와 뷰 객체간의 뷰 유도 관련성(VDR: View Derivation Relationship)이다[17]. 특히 본 논문에서는 차후 변경(Deferred Update)을 이용하기 위해 CDR에 타임스탬프(timestamp)의 기능을 추가하였다.

클래스 유도 관련성은 소스 객체가 변경되었을 때, 변경된 소스 객체의 클래스로부터 유도된 뷰 클래스들을 찾기 위해 모든 뷰-정의 질의의 From절을 분석하는 비효율적인 방법을 개선한 방법이다[17]. 본 논문에서는 기존의 클래스 유도 관련성을 확장한다.

```
Struct  CD1  {
String              SourceClassName;
Sequence<CD2>       Attributes;
}
Struct  CD2  {
String              AttributeName;
Sequence<String>    SpatialViewNames;
Time                TimeStamp;
}
```

그림 9 클래스 유도 관련성의 구조

그림 9는 확장된 클래스 유도 관련성으로 소스 객체 변경시 이로부터 유도된 클라이언트 뷰의 변경시간을 CD2의 타임스탬프에 저장한다. 타임스탬프의 사용 목적은 클라이언트에서 일관성 제어 요청이 있는 경우 서버에서 변경로그를 검색하여 변경된 객체가 뷰의 소스 클래스에 속하는지 검사하는 비용을 감소시킨다. 즉, 클라이언트 뷰의 소스 클래스 정보인 CD1을 검색하여 이로부터 유도된 CD2의 타임스탬프와 실체화된 클라이언트 뷰의 가장 최근 일관성 제어시간을 비교하여 변경 여부를 빠르게 검사할 수 있다.

실체화된 클라이언트 뷰의 일관성 제어를 위해 소스 객체가 변경된 경우에 직접적으로 영향받는 뷰 객체를 검색하기 위한 방법으로 뷰 객체와 소스 객체의 식별자들로 구성되며, 뷰 객체와 대응되는 소스 객체들간에 유도 관계를 나타내는 뷰 유도 관련성을 이용한다. 뷰 유도 관련성은 뷰 객체와 소스 객체들간의 인스턴스 수를 나타내는 카디널리티 비율에 따라 일대일(1:1), 다대일(N:1), 일대다(1:N), 다대다(N:M)로 분류된다.

변경 로그는 서버에서 변경된 객체에 대한 정보를 클라이언트에게 제공하기 위한 추가적인 정보로서 <Update-type, Oid, Timestamp> 의 구조를 가진다. Update-type은 변경된 소스 객체의 변경 형태로서 삽입, 삭제, 수정의 3가지이다. Oid는 변경된 소스 객체의 식별자를 나타내고, 타임스탬프는 객체가 변경된 시간을 알려주는 정보이다.

타임스탬프는 크게 두 가지 목적을 위해서 사용된다. 첫째, 변경 로그에서 동일한 객체의 변경시 중복되는 일관성 유지 작업을 피하기 위하여 사용된다. 일관성 유지에서 요구시-처리 방법은 클라이언트의 요청이 있는 경우 일관성을 유지하므로 동일 객체가 여러 번 변경될지라도 마지막으로 변경된 경우에 대해서만 일관성 유지 작업을 수행한다.

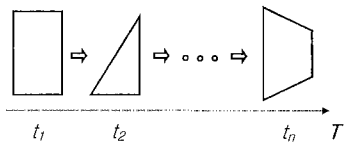


그림 10 가하 객체의 변경

예를 들어, 그림 10과 같이 t_1상태에서 기하객체가 t_2에서 변경되었을 때 즉시 변경에서는 뷰 객체의 일관성 제어를 위한 작업이 소스 객체의 변경 직후 수행된다. 따라서 t_n인 경우 소스 객체에 대한 변경 횟수에 대한 총 비용은 다음과 같다.

$$Cost = \sum_{i=1}^{n} k_i$$

여기서 n은 소스 객체의 변경 횟수, k_i는 i번째 뷰 객체의 변경시간을 나타낸다. 그러나 타임스탬프를 이용하는 경우 t_1에서 실체화 후 t_n에서 뷰의 일관성을 유지할 경우 최대 $\sum_{i=0}^{n-1} K_i$ 만큼의 비용을 감소시키는 효과가 있다. 그러나 최악의 경우 소스 객체의 변경 마다 일관성을 유지하기 위한 작업을 수행한다면 즉시 변경과 동일한 비용을 수반한다.

둘째, 실체화된 클라이언트 뷰의 일관성 제어 과정에서 소스 객체의 변경 유무를 빠르게 검사하기 위한 추가 정보인 클래스 유도 관련성에서 사용된다. 즉 변경된 객체를 저장하고 있는 변경 로그를 모두 검색하지 않고 클래스 유도 관련성에 저장된 타임스탬프를 이용하여 변경 유무를 효율적으로 검사한다.

4.2.2 일관성 제어 프로토콜

클라이언트 뷰의 일관성 검사를 위해 클라이언트는 서버와의 연결을 필요로 하고, 서버는 일관성 유지를 위해 추가적인 정보를 클라이언트로 전송해야 한다. 그림 11은 클라이언트와 서버의 일관성 제어를 위한 일련의 프로토콜을 나타낸다.

본 논문에서는 CDR에 대한 기본적인 공용 잠금(shared-lock), 전용 잠금(exclusive-lock)을 사용한다. 클라이언트에서 일관성 제어를 위해 먼저 CDR에 대한 공용 잠금을 서버에 요청한다.

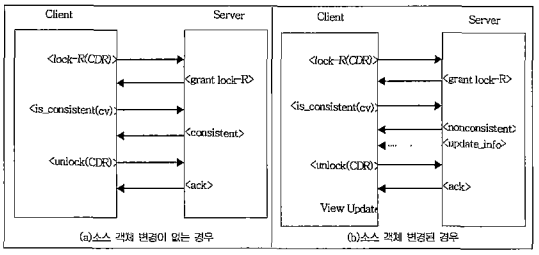


그림 11 일관성 제어 Protocol

만약 서버가 객체의 변경 정보를 저장하기 위해 CDR에 대해 전용 잠금을 한 경우에는 잠금 호환성에 의해 클라이언트는 잠금을 할 수 없다. 따라서 이 경우에 클라이언트는 일정시간이 경과한 후 재시도한다. 클라이언트가 CDR에 대해 공용 잠금을 획득한 경우에는 일관성을 유지하려고 하는 클라이언트 뷰의 정의문을 포함한 <*is_consistent*>메시지를 서버로 전송한다. 서버에서 소스 객체의 변경 유무를 검사한 뒤 소스 객체가 변경되지 않은 경우에는 그림 11(a)와 같이 <*consistent*>메시지를 클라이언트로 전송한다. 그림 11(b)와 같이 소스 객체가 변경된 경우에는 <*nonconsistent*>메시지와 변경 정보를 클라이언트로 전송한다. 메시지를 받은 클라이언트는 CDR에 대한 공용 잠금을 반환한다.

4.2.3 일관성 제어 알고리즘

그림 12는 클라이언트/서버 환경에서 실체화된 클라이언트 뷰의 일관성 제어를 위한 전체적인 수행 과정이다. 일관성 제어 과정은 크게 4단계로 나눌 수 있다.

1단계는 서버에 저장된 소스 객체의 변경시에 CDR과 변경 로그를 수정하는 단계이다. 서버는 변경된 소스 객체로부터 유도된 뷰가 존재하는지를 CDR을 이용하여 검사한다. CDR의 시간 정보가 변경된 경우 실체화된 클라이언트 뷰의 일관성 제어가 필요하게 되므로 변경 객체의 정보를 변경 로그에 기록하게 된다. 변경 로그의 기록은 크게 객체의 삽입, 삭제, 변경의 3가지로 분류된다. 객체가 삽입된 경우에는 변경 정보를 변경 로그에 추가한다. 객체가 수정된 경우에는 변경 로그의 검색이 필요하다. 이것은 동일 객체의 삽입, 수정, 삭제의 경우 각 변경 시기마다 변경 로그를 생성하여 각 변경 정보에 따라 불필요한 계산을 수행하는 단점을 개선한다. 변경 로그의 검색은 변경된 객체의 식별자를 가진 변경 로그의 삽입과 수정에 대한 검색을 의미한다. 객체가 수정된 경우 검색 유형은 3가지로 구분된다. 첫째, 삽입의 변경 유형이 존재하는 경우 변경 유형을 삽입에서 수정의 형태로 변경하고 시간 정보를 수정한다. 둘째, 수정의 변경 유형이 존재하는 경우 시간 정보만 수정한다. 마지막으로 이전 변경 정보가 검색되지 않을 경우에는 변경 로그에 새로운 로그 정보를 추가한다. 객체가 삭제된 경우에 변경 로그의 검색은 변경 유형의 삽입과 수정의 경우로 객체가 수정된 경우와 동일하다. 그러나 로그 정보의 유형이 삭제의 형태로 변경되고 시간 정보의

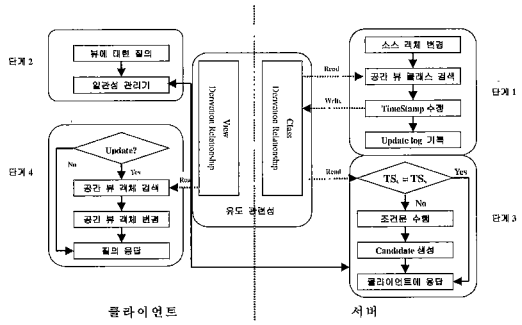


그림 12 일관성 제어 시나리오

수정이 필요하다.

2단계는 클라이언트가 뷰에 대한 질의를 요청하는 경우로서 실체화된 클라이언트 뷰 객체의 일관성을 검사하기 위해 서버로 변경 정보를 요청하는 단계이다. 이 단계에서 필요한 정보는 일관성을 유지하려는 뷰의 이름, 소스 클래스의 이름, 그리고 실체화된 클라이언트 뷰 객체의 최종 변경시간이 필요하다.

3단계는 서버에서 클라이언트의 일관성 제어에 대한 요청을 처리하는 단계이다. 서버는 CDR에 저장된 소스 객체의 변경시간(TS_s)과 실체화된 클라이언트 뷰의 변경시간(TS_c)을 비교한다. 소스 객체가 변경되지 않은 경우(TS_c = TS_s)에는 실체화된 클라이언트 뷰 객체의 유효성이 보장된다. 그러나 소스 객체가 변경된 경우(TS_c < TS_s)에는 실체화된 클라이언트 뷰 객체는 유효하지 않다. 따라서 일관성이 파괴된 경우에는 일관성 제어를 위하여 서버에서 뷰-정의 질의를 수행하여 생성된 후보 객체를 클라이언트로 전송한다. 특히, 후보 객체는 뷰-정의 질의의 질의 결과로서 생성되기 때문에 뷰-정의 질의를 재수행하여 후보 객체를 생성하는 것은 많은 시간을 필요로 한다. 따라서 본 논문에서는 질의 최적화 방법인 small-join방법[2]을 적용한다.

마지막 단계는 서버로부터 전송된 후보 객체를 이용하여 실체화된 클라이언트 뷰 객체를 변경한다. 뷰 객체의 변경은 VDR을 이용하여 변경된 소스 객체로부터 유도된 뷰 객체를 검색하고, 검색된 뷰 객체를 수정한다.

실체화된 클라이언트 뷰 객체에 대한 일관성 제어는 서버 작업과 클라이언트 작업으로 분류된다. 서버 작업의 경우 그림 13과 같이 클라이언트의 일관성 제어 요청에 대해 CDR정보를 이용하여 소스 클래스의 인스턴스들의 변경 유무를 검사하기 위해 먼저 *GetSource Attribute()*를 호출하여 뷰 정의문으로부터 소스 클래스의 애트리뷰트들을 추출한다. 추출된 애트리뷰트들은 소

CVD	Client View Definiton	SA	: Source Attribute
t_s	: the time of updating object in Server	QR_i	: Query Result
t_c	: the time of materializing in Client	OID	: Object ID
$\{C_i\}$	: Candidate Object Set		

```
Procedure IsConsistent(CVD, t_c)
begin
        {SA_i} = GetSourceAttribute(CVD);
        for each  SA_i  in  {SA_i}
        begin
                t_s = GetTimeStampInCDR(SA_i);
                if ( t_s > t_c )  // 소스 객체가 변경된 경우
                begin
                        {QR_i}= SpatialQuery(CVD),
                        {OID_k} = GetUpdatedObjectInLog(CVD, t_c);
                        {C_i} = GetCandidateFromQueryResult({QR_i}, {OID_k});
                        Exit,
                end if;
        end for;
end;
```

그림 13 일관성 유지 알고리즘

스클래스의 정보를 포함한다. 그리고 각 애트리뷰트의 변경 유무를 검사하기 위해 *GetTime StampInCDR()*을 호출하여 애트리뷰트의 최종 변경시간을 CDR로부터 검색한다. 만약 검색된 최종 변경시간과 실체화된 클라이언트 뷰 객체의 최종 변경시간을 비교하여 검색된 최종 변경시간이 클 경우(일관성이 파괴된 경우)에는 뷰-정의 질의를 수행한 후 질의 결과에서 삽입, 삭제, 수정된 객체를 추출하여 이를 클라이언트로 전송한다.

소스 객체의 변경은 클라이언트 뷰의 종류와 소스 객체의 변경 유형에 따라 서로 다른 정보를 클라이언트로 전송한다. 하나의 소스 클래스로부터 유도된 뷰의 경우에는 삽입, 수정된 객체가 프레디킷에 만족할 경우에만 전송하고, 삭제된 객체는 객체 식별자만을 전송한다. 두 개 이상의 소스 클래스로부터 유도된 뷰의 경우에는 삽입, 수정된 객체는 SpatialQuery함수에서 small-join을 이용한 공간 조인을 수행하여 서버의 공간관련성연산자의 수행 비용을 최적화한다. 삭제된 객체의 경우에는 객체 식별자만을 전송한다.

클라이언트 작업은 소스 객체가 변경된 경우 후보 객체들을 이용하여 [17]에서 제시한 점진적 변경 알고리즘에 따라 실체화된 클라이언트 뷰 객체를 변경한다.

5. 성능 평가

5.1 실험데이타 및 환경

본 논문에서 제시한 일관성 제어 방법의 성능 평가를 위하여 이용한 데이타는 정확한 성능 평가를 위하여 현재 지자체에서 사용하고 있는 실제 데이타를 이용한다.

클래스의 수는 총 71개로서 건물, 구역, 등고선 등 다양한 공간 객체들로 구성되어 있으며 총 객체 수는 131,534개이다. 그리고 서버의 CPU 속도는 400 MHz이고 메모리는 512M인 디지탈(DEC)사의 ALPHA 스테이션을 이용하고, 클라이언트는 CPU속도 133MHz이고 메모리 32M인 PC를 이용한다.

성능 평가를 실시한 클라이언트 뷰는 표 2와 같다. [클라이언트 뷰1]은 분류코드가 '아파트'인 건물로 정의되는 속성 질의이고 기하데이타는 소스 객체의 기하 데이타를 복사한다. [클라이언트 뷰2]는 '반지동'이라는 행정구역내에 있는 가로등을 정의하는 공간 질의로서 WITHIN연산자를 사용하고 기하 데이타로 가로등의 기하 데이타를 버퍼링한다. [클라이언트 뷰3]은 CROSSES 연산자를 사용하는 공간 질의로서 교차점을 버퍼링하는 기하를 정의하는 클라이언트 뷰이다.

표 2 실험 대상 공간뷰의 예

	클라이언트 뷰 정의문
1	Spatial_View 아파트(고유번호,본번,부코드,분류코드,시설물명,geom) As SELECT 건물.고유번호,건물.본번,건물.부코드,건물.분류코드,시설물명,None(건물) FROM 건물 WHERE 건물.분류코드="4115"
2	Spatial_View 반지동내의가로등(고유번호,번호,부코드,geom) As SELECT 가로등.고유번호,가로등.번호,행정동경계.부코드, Buffer(가로등,2.0) FROM 가로등,행정동경계 WHERE 행정동경계.부코드="3000140000" AND WITHIN(가로등,행정동경계)
3	Spatial_View 상수관로공사시주의할지점(도로번호,종류,폭,geom) As SELECT 가로등전선관.도로번호,가로등전선관.종류,상수관로.폭, Intersect_To_Buffer(가로등전선관,상수관로,2.0) FROM 가로등전선관,상수관로 WHERE CROSSES(가로등전선관,상수관로)

5.2 시스템 구조

본 논문에서 구현한 클라이언트 뷰 시스템의 전체 구조는 크게 클라이언트 모듈과 서버 모듈로 구성되고, 각각의 모듈은 클라이언트 뷰의 정의, 실체화 및 일관성 제어를 수행한다.

클라이언트 모듈은 그림 14와 같이 3계층으로 구성된다. 가장 하위 계층인 서비스 제공자(service provider)는 상위 계층에 공통적인 서비스를 제공하는 계층으로 실체화된 클라이언트 뷰 객체와 일관성 제어를 위한 추가 정보인 VDR의 접근을 위한 지역 데이타 접근 서비스(local data access service), 통신 패킷 및 클라이언트 뷰 객체를 생성하기 위한 팩토리 서비스(factory service), 서버와의 통신을 담당하는 통신 서비스(communication service)와 클라이언트 뷰 객체와 소스 객체간의 관련성을 유지하는 VDR관리 서비스(VDR manager service)로 구성된다.

서비스 제공자의 서비스를 이용하는 클라이언트 뷰 모듈은 클라이언트 뷰 정의기(CV builder), 클라이언트 뷰 실체화기(CV materializer), 클라이언트 뷰 일관성 관리기(CV consistency manager)로 구성된다. 클라이언트 뷰 정의기는 클라이언트 뷰를 정의하기 위한 인터페이스를 제공한다. 클라이언트 뷰 실체화기는 서버가 전송한 뷰-정의 질의의 결과를 이용하여 클라이언트 뷰 객체를 생성하는 작업과 클라이언트 뷰 객체와 소스 객체의 유도 관련성 정보인 VDR을 생성하는 작업을 수행한다. 클라이언트 뷰 일관성 관리기는 클라이언트에 저장된 뷰 객체들의 일관성을 담당하는 모듈로서 서버에 변경 정보를 요청하여 변경된 정보가 있는 경우에 VDR을 이용하여 점진적 변경 방법으로 일관성을 유지한다.

서버 모듈은 그림 15와 같이 3계층으로 구성된다. 가장 하위 계층은 상용 공간 데이타베이스인 GOTHIC[4]의 API를 이용하여 서비스를 제공하는 모듈들로서 공간 질의 처리를 위한 공간 질의 서비스(spatial query service), 통신 패킷 생성을 위한 팩토리 서비스(factory service), 통신 패킷을 처리하는 통신 서비스(communication service), 뷰 클래스와 대응하는 소스 클래스의 관련성인 CDR정보와 소스 객체의 변경 정보를 관리하는 CDR & Log 관리 서비스(CDR & Log manager service)로 구성된다.

서버의 클라이언트 뷰 모듈은 클라이언트에서 메시지를 통해 전송된 요청을 처리하게 된다. 클라이언트의 요청은 클라이언트 뷰 정의기(CV definer), 질의 관리기

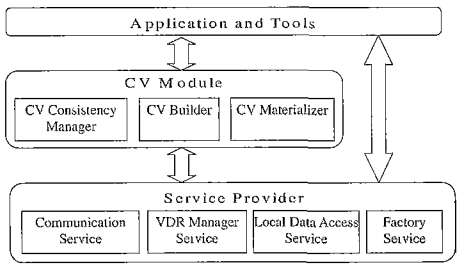


그림 14 클라이언트 구조

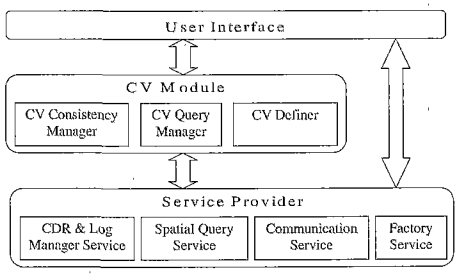


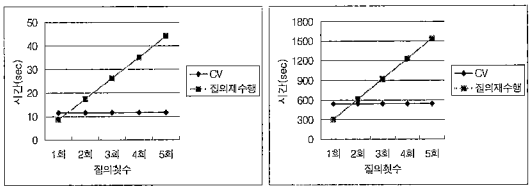
그림 15 서버 구조

(CV query manager), 그리고 클라이언트 뷰 일관성 관리기(CV consistency manager)에서 처리된다.

서버의 클라이언트 뷰 정의기는 클라이언트에서 요청한 뷰 정의문을 이용하여 소스 클래스와 클라이언트 뷰 클래스간의 유도 관련성 정보인 CDR을 생성시킨다. 질의 관리기는 클라이언트의 실체화 요청시 공간 연산을 포함하는 뷰 정의-질의의 처리를 담당한다. 클라이언트 뷰 일관성 관리기는 클라이언트의 일관성 제어를 위해 서버 작업을 담당하는 모듈로서 서버의 소스 객체의 변경시 변경 정보를 저장하고, 클라이언트에게 변경된 정보를 전송하는 역할을 한다.

5.3 질의 횟수에 따른 성능 평가

그림 16은 클라이언트 뷰를 이용한 경우와 이용하지 않고 질의 수행한 경우에 대한 누적시간을 나타낸다. 실험 평가에 사용된 질의는 [클라이언트 뷰1]과 [클라이언트 뷰2]를 사용한다. [클라이언트 뷰1]은 속성에 대한 질의로서 대상 객체 수는 13,567개이다. [클라이언트 뷰2]는 공간 연산자인 WITHIN연산자를 이용하여 공간 조인을 수행하는 공간 질의로서 질의 대상 객체 수는 11,567 X 11 개이다. 속성 질의와 공간 질의의 두 가지 성능 평가에서 클라이언트 뷰를 이용한 경우 최초 질의 이후의 동일 질의시는 클라이언트에 저장된 실체화된 클라이언트 뷰를 이용하여 질의 처리를 수행하기 때문에 사용하지 않는 경우보다 빠르다.



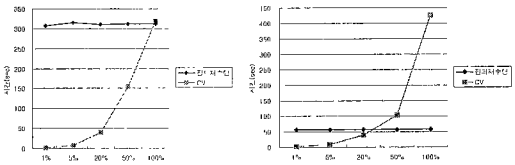
(a) 측정질의 (b) 공간질의

그림 16 클라이언트 뷰의 성능평가

반면 최초 수행시 클라이언트 뷰를 이용하는 경우에 클라이언트 뷰를 사용하지 않은 경우보다 평균 1.5배 느리다. 이와 같이 속도가 느린 이유는 뷰를 실체화하고 일관성 유지를 위해 부가 정보인 VDR을 생성하기 때문이다. 따라서 클라이언트 뷰 시스템은 사용자가 빈번히 사용하는 질의에 대해서 효율적이다.

5.4 소스 객체 변경에 따른 성능 평가

공간 데이타베이스에 저장된 소스 객체가 변경될 경우 실체화된 클라이언트 뷰의 일관성이 파괴된다. 이 경우 질의의 재수행 비용과 본 논문에서 제시한 일관성 제어 비용을 비교한다. 그림 17은 표 2의 [클라이언트 뷰2]와 [클라이언트 뷰3]에 대한 일관성 제어 비용과 질의 재수행 시간을 비교한다. 클라이언트의 실체화된 뷰 객체들은 소스 객체에 변경에 따라 점진적 변경을 하기 때문에 변경된 소스 객체가 증가할수록 비용이 선형적으로 증가한다. 그림 17(b)의 경우에는 소스 객체가 30% 이상 변경될 경우 일관성 제어 비용이 질의 재수행 비용보다 비효율적이다. 그러나 일반적인 GIS 데이타의 경우에는 소스 객체의 변경율이 5% 미만이므로 소스 객체의 변경율이 5% 미만인 부분에서는 실체화된 클라이언트 뷰의 일관성 제어 방법이 질의 재수행 방법보다 효율적이다.



(a) 클라이언트 뷰2 (b) 클라이언트 뷰3

그림 17 소스 객체 변경에 따른 성능 평가

5.5 일관성 제어 비용 분석

소스 객체의 변경으로 실체화된 클라이언트 뷰 객체의 일관성이 파괴된 경우 뷰 객체의 일관성 제어 비용을 서버와 클라이언트로 나눌 수 있다. 특히 서버의 비용은 클라이언트가 요청한 뷰의 변경여부를 검사하는 비용, 일관성이 파괴된 경우에 공간 관련성 연산을 수행하는 selection 비용, 그리고 후보 객체의 전송 비용으로 구성된다. 그리고 클라이언트 비용은 실체화된 클라이언트 뷰 객체를 점진적으로 변경하기 위한 시간이다. 그림 18은 [클라이언트 뷰3]에 대한 일관성 제어 비용을 클라이언트 뷰 객체의 변경율에 따라 분석한 것이다.

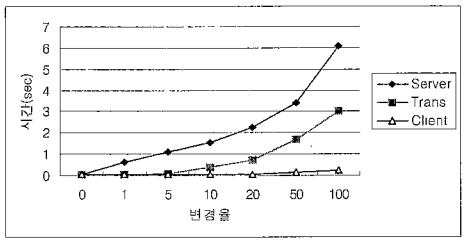


그림 18 일관성 제어 비용

서버의 비용중 뷰 유도 관련성을 이용하여 클라이언트 뷰의 변경 여부를 검사하는 비용은 공간관련성연산자를 처리하는 selection 비용에 비해 2% 이하의 비중을 차지하고 있으므로 그림 18에서 서버의 비용을 뷰-

정의 질의의 selection 처리비용으로 나타내었다. 그리고 selection작업을 수행하여 생성된 후보객체(쌍)를 클라이언트로 전송하기 위한 전송비용이 selection 처리비용의 10~50%를 차지하였다. 이것은 소켓을 이용한 전송 패킷을 생성하기 위한 비용으로 후보 객체(쌍)의 수에 비례하고 후보 객체(쌍)의 수는 뷰 객체의 변경 비율에 비례한다. 그리고 클라이언트에서 실체화된 클라이언트 뷰 객체의 점진적 변경 비용은 서버의 selection 비용의 4% 이하로 측정되었고, 변경된 객체 수에 비례함을 알 수 있다.

6. 결론 및 향후 연구

클라이언트/서버 환경에서 공간 데이타를 빠르게 검색하기 위한 방법으로 실체화된 공간뷰의 개념을 적용할 필요가 있다. 본 논문에서는 공간 객체에 대한 다양한 사용자 관점을 지원하는 공간뷰를 클라이언트/서버 환경에서 적용하기 위해 실체화할 수 있는 클라이언트 뷰의 유형을 분류하였다. 그리고 클라이언트 뷰의 실체화 및 일관성 제어를 위한 점진적 변경 알고리즘을 제시하고 객체지향 공간 데이타베이스에서 실체화된 클라이언트 뷰를 설계하고 구현하였다.

본 논문에서 공간뷰 개념을 클라이언트/서버 환경으로 확장한 클라이언트 뷰는 실체화과정에서 공간관련성 연산자를 포함하는 selection과 기하생성연산자를 포함하는 projection의 작업 분담을 기준으로 4가지로 분류된다. 특히 뷰-정의 질의에서 공간관련성연산자를 포함한 프레디킷의 처리를 서버에서 담당하고, 기하생성연산자를 포함하는 기하 생성 작업을 클라이언트에서 처리하는 유형 4에 대해 값 복사 방법을 적용하여 실체화하였다. 그리고, 소스 객체 변경에 따른 실체화된 클라이언트 뷰 객체의 일관성 제어를 위하여 뷰 객체에 대한 점진적 변경을 요구시에 처리할 경우 필요한 추가정보를 기술하고, 추가 정보를 이용한 일관성 제어 방법을 제시하였다.

마지막으로, 구현한 클라이언트 뷰 시스템의 성능을 검증하기 위해 실제 데이타를 사용하여 실험 평가를 수행하였다. 먼저 클라이언트에서 질의 수행시간과 실체화된 클라이언트 뷰의 질의 수행시간을 비교하였다. 또한 소스 객체의 변경시 점진적 변경과 재계산 시간을 비교하였다. 성능 평가를 통하여 본 논문의 클라이언트 뷰가 질의 재수행 방법 보다 효율적인 것을 증명하였고 클라이언트가 소스 객체의 변경이 빈번히 발생하지 않는 공간 데이타베이스를 검색할 경우 훨씬 효율적이라는 것을 알 수 있었다.

앞으로 공간뷰를 미들웨어를 사용하여 이질적인 공간 데이타베이스 서버에 접근하여 상호 운용성을 지원하는 공간뷰 서비스 제공자에 대한 연구가 필요할 것이다.

참 고 문 헌

[1] Nick Roussopoulos , Hyunchul Kang Principles and Techniques in the Design of ADMS±, IEEE Computer, pages 19-25, 1986.

[2] Nick Roussopoulos, An incremental access method for ViewCache : concept, algorithms, and cost analysis, ACM Trans.Database Syst. vol 16, no 3, pages 535-563, 1991.

[3] Alex. Delis and Nick Roussopoulos, Techniques for Update Handling in the Enhanced Client-Server DBMS, IEEE TOKD, pages 458-476, 1998.

[4] *Laser-Scan Ltd, GOTHIC CONCEPTS, Training* Course, 1995.

[5] Nick Roussopoulos, Chungmin M. Chen, Stephen Kelley, Alex Delis and Yannis Papakonstantinou, The ADMS Project: Views "R" Us, IEEE Data Engineering, pages 19-28, 1995.

[6] Latha S. Colby, Timothy Griffin, Leonid, and Libkin, Algorithms for Deferred View Maintenance, Proc. of ACM-SIGMOD Int'l Conf. on Management of Data, pages 469-480, 1996.

[7] Alfons Kemper and Guido Moerkotte, Object-Oriented Database Management : Applications in Engineering and Computer Science, Prentice-Hall, 1994.

[8] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian, Maintaining Views Incrementally, Proc. of ACM-SIGMOD Int'l Conf. on Management of Data, pages 157-166, 1993.

[9] Ashish Gupta and Inderpal Singh Mumick, Maintenance of Materialized Views: Problems, Techniques, and Applications, Proc. of Int'l Conf. on Data Engineering, pages 3-18, 1995.

[10] C. J. Date, "Views, An Introduction to Database System, Addison-Wesley Publishing Company, pages 466-496, 1995.

[11] C. Souza dos Santos, Design and Implementation of Object-Oriented Views, Proc. of Int'l Conf. and Workshop on Database and Expert Systems Applications (DEXA), pages 91-102, 1995.

[12] Elke A. Rundensteiner, MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases, Proc. of the Int'l Conf. on VLDB, pages 187-198, 1996.

[13] Harumi. A. Kuno and Elke A. Rundensteiner, *Materialized Object-Oriented Views in MultiView*, Int'l Workshop on Research Issues on Data

Engineering: Distributed Object Management, 1995.
[14] Harumi A. Kuno and Elke A. Rundensteiner, Using Object-Oriented Principles to Optimize Update Propagation to Materialized Views, Proc. of Int'l Conf. on Data Engineering, 1996.
[15] Harumi A. Kuno and Elke A. Rundensteiner, The MultiView OODB View System: Design and Implementation, Theory and Practice of Object Systems, vol. 2, no. 3, pages 203-225, 1996.
[16] Sang-Ho Moon and Bong-Hee Hong, Design and Implementation of Materialized Spatial Views, Proc. of Int'l Conf. on Object Oriented Information Systems (OOIS), pages 386-396, November 1997.
[17] Sang-Ho Moon and Bong-Hee Hong, Incremental Update Algorithms for Materialized Spatial Views by Using View Derivation Relationships, Proc. of Int'l Conf. and Workshop on Database and Expert Systems Applications (DEXA), pages 539-550, September, 1997.
[18] 문 상호,김 동우,반 재훈,홍 봉희, 객체지향 공간뷰의 설계 및 구현, 정보과학회 논문지(B), vol.26, pages 306-320, 1999.
[19] 임 덕성,반 재훈,문 상호,홍 봉희, 클라이언트 공간뷰의 실체화 방법, 정보과학회 봄 학술발표 논문집 vol.26, no.1, 1999.
[20] 임 덕성,반 재훈,문 상호,홍 봉희, 일관성 제어를 지원하는 클라이언트 공간뷰의 설계 및 구현, 정보과학회 가을 학술발표 논문집 vol.26, no.1, 1999.

임 덕 성
1998년 2월 동아대학교 컴퓨터공학과 공학사. 2000년 2월 부산대학교 컴퓨터공학과 공학 석사. 2000년 3월 ~ 현재 부산대학교 컴퓨터공학과 박사과정. 관심분야는 지리정보시스템(GIS), 공간 뷰, GIS 표준화, 모빌 GIS

반 재 훈
1997년 2월 부산대학교 컴퓨터공학과 공학사. 1998년 2월 부산대학교 컴퓨터공학과 공학석사. 1999년 3월 ~ 현재 부산대학교 컴퓨터공학과 박사과정. 관심분야는 지리정보시스템(GIS), 공간 뷰, GIS 표준화

문 상 호
1991년 2월 부산대학교 컴퓨터공학과 공학사. 1991년 ~ 1993년 한국기계연구원 전산시스템실 연구원. 1994년 2월 부산대학교 컴퓨터공학과 공학박사. 현재 위덕대학교 컴퓨터공학과 조교수. 관심분야는 지리정보시스템(GIS), 공간뷰, 객체지향데이타베이스, GIS 표준화

홍 봉 희
정보과학회논문지 : 데이타베이스
제 28 권 제 1 권 참조