객체 복제 기법에 의한 원격 접근 알고리즘

윤 동 식[†] · 이 병 관[††]

요 약

이 논문은 분산 컴퓨팅하에서의 객체 복제 클라이언트/서버 설계와 구현을 나타낸다 오늘날 많은 사용자들은 클라이언트/서버의 분산 시스템 내의 인터넷 컴퓨터 통신을 하고 있다 만약 많은 사용자들이 특정 원격 서비스를 요구하면, 그 서버는 서비스 처리에 대한 과부하를 가져오게 되고, 재시작을 위한 지연을 가지게 되며, 통신망에서는 병목현상이 발생하게 된다 그러므로 객체 복제 기법은 이 문제의 해를 제공해준다 분산 응용과 인터넷 작업들의 성장은 캐시메모리 복제시스템의 발전 필요성을 가져오게 되었다 그러나 현존하는 복제 프로토콜들은 주소, 알맞은 일역을 지정하지 못한다 게다가 현재의 응용 프로토콜은 다른 계층들의 일관성을 요구하게 된다. 그러므로 그들 내에서 일관성 유지 함수가 선택되어야만 된다. 그래서 각 서버은 특별한 시맨틱스를 가지고 있어야 한다 본 논문에서는 서버 객체 복제를 사용하여 원격 프로시쥬어 호출시 서버 과부하와 병목현상을 줄이고자 한다. 또한 접근 투명성이 공유 객체 복제에 의해 증가하게 된다 그리고 그것은 복제된 객체들의 일관성을 유지하게 된다

The Remote Access Algorithm by Object Replication

Dong-Sic Yun[†] · Byung-Kwan Lee[††]

ABSTRACT

In This paper, object replecation Client/server under distributed computing system is design and implementation Today many end-users have a computer communication by using internet in the distributed system of client/server If many users request services to a specific remote server, the server should have got a overhead for that service procesing, delayed the speed for replay, and brings a bottleneck in communication network Therefore object replication method was proposed to solve this problems. The growth of internet works and distributed applications has increased the need for large scale replicated systems however, existing replication protocols do not address scale and autonomy issues adequately. further, current application protocold require consistency of different levels, and therefor should be the selection function of consistency in them, in order to have particular semantics of each level In this paper, server overhead and bottleneck happening in remote procedure call be using server object replication Therfore access transparency can be improved by sharing object duplicately So it will Keep up with the consistency within the replicated objects

1. 서 론

대부분의 분산 컴퓨팅 시스템에서는 하나의 호스트(host)를 기준으로 여러 대의 소형 컴퓨터들을 네트워크로 구성하여 자원을 공유하고 관리하여 왔다. 이런 시스템에서 인터넷과 같이 수천 수만에 해당되는 사용자들이 자원을 요구하거나, 전송하고자 할 때 많은 지연 시간이 발생되고 혹은 노드의 연결이 매끄럽지 않게 되거나 단락 되는 경우가 흔히 발생한다 기존의 클라이언트/서버 모델은 여러개의 터미널로 연결되어 하나의 대형 서버에 다수의 소형 컴퓨터들로 구성되어

† 종신회원 안동과학대학 사무자동화과 교수
†† 종신회원 관동대학교 전자계산공학과 교수
논문접수 1998년 10월 7일, 심사완료 : 1999년 12월 30일

있다. 그러나 근래에는 정보통신 분야의 기술 향상으로 클라이언트/서버의 개념이 다소 변경되어가고 있다. 워크스테이션, 터미널 PC등의 클라이언트와 단일 서버가 아닌 다수개의 서버를 사용하여 각 서버의 기능과 성능에 맞게 서비스를 제공 할 수 있게 시스템이 변형되어가고 있다. 다수개의 서버를 두고 클라이언트가 서버로 접속하여 서비스를 받게된다면, 원거리의 단일 서버로의 접근보다 통신 지연속도를 줄일 수 있을 것이다. 그러나 원격지의 특정 서버로 다수의 클라이언트가 서비스를 요구 할 경우에는 해당 서버에서 처리할 수 있는 능력 이상의 오버 헤드가 발생 할 수 있다. 또한 통신 선로 상에 병목 현상이 발생하여 시스템의 접근 속도가 지연될 것이다. 본 연구에서는 통신상에서 발생되어지는 병목 현상과 자원의 복제에서 발생되어지는 오버 헤드를 객체 복제 프로토콜(Object replication protocol)을 이용하여 인접 서버로부터 서비스 받게 하여 시스템 서비 및 특성 서버의 오버헤드를 줄이고자 한다.

본 논문은 최근 객체 분산 환경에 ORB(Object Management Broker), OMG(Object Management Group)에 의해 표준화된 CORBA (Common Object Request Broker Architecture)로 객체 복제 클라이언트/서버 시스템을 구축 하였다.

2. 분산 컴퓨팅 시스템 환경

2.1 클라이언트/서버 환경

기존의 클라이언트/서버 모델은 몇 개의 터미널로 구성된 클라이언트와 하나의 서비에 네트워크로 연결된 형태를 가지고 있다. 분산 컴퓨팅 시스템에 구성되어지는 서버 및 클라이언트들은 이질적 형태를 지니게 된다. 이와 같은 시스템을 통합하기 위해서 1990년 OSF(Open Software Foundation)의 DCE(Distributed Computing Environment)가 개발되어 각종 어플리케이션을 개발, 사용 유지시켜 주었다 DCE는 통신용 RPC, 셀 디렉토리, 분산형 파일 서비스, 보안 서비스, 멀티스레드 소프트웨어 어플리케이션을 만들어 주는 스레드로 구성되어져있다. 일반적인 분산 컴퓨팅 시스템에서 클라이언트/서비의 형태는 (그림 1)과 같이 그룹 서버를 가진 클라이언트가 원격지의 특정 서버로 접근하여 서비스를 제공받는 형태의 네트워크를 형성하고 있다[3].

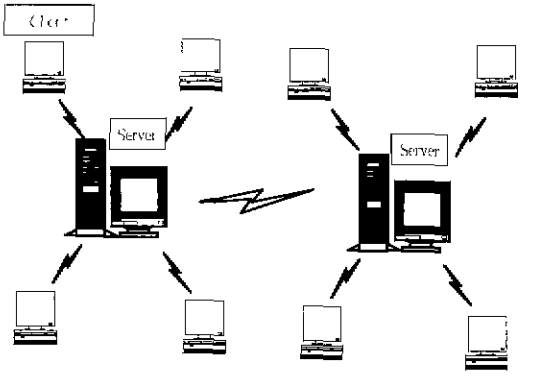
최근에 객체 지향 프로그래밍, 객체 지향 데이터베이스 등 객체 지향 개념의 도입으로 프로그램과 데이터베이스에 있어서 중복성을 배제하려는 노력을 기울이고 있다 그러나 하나의 특정된 서버로부터 서비스를 요구한다면 다음과 같은 문제점이 발생하게 될 것이다.

첫째 데이터 중복성의 제약 때문에 특정한 원격지 클라이언트들을 제외하고는 나른 모든 클라이언트들은 인접된 서버로부터 특정한 원격지의 서버가 위치한 거리만큼의 서비스 지연 시간을 갖게 된다.

둘째 각각의 클라이언트들은 특정된 하나의 서비로 서비스를 요청하므로 막대한 오버헤드를 불러일으키게 된다.

셋째. 각각의 클라이언트들은 특정된 하나의 서비로부터 서비스를 받기 위하여 특정 서버로 집중되기 때문에 이 서버 부근의 통신선로는 심한 병목현상을 가져온다. 이와 같은 문제점을 해결할 수 있는 방법으로는 객체 지향 개념을 적용한 데이터 중복성을 허용한 객체 복제 프로토콜을 제시하였다.

본 논문에서는 기존의 분산 시스템의 복제 프로토콜(replication protocol)을 각 계층으로 구분 지어서 각 클러스터로 구성하고 각각의 클러스터들을 객체로 연결하여 필요시에만 각 계층의 노드에 연결 지어 복제하여 사용하고자 한다. 또한 필요성을 표기하는 상태 노드를 두어 동기적인 신호에 따라 각각의 지역 클러스터에서 무작정 복제에 따른 메모리의 낭비를 막고자 한다. 그리므로 분산 시스템의 복제(replicated)데이터의 유용성과 시스템의 성능을 향상시켜 광역 통신망에서도 지역 통신망을 연결하여 사용하는 것처럼 하고자 한



(그림 1) 클라이언트/서버 모델의 일반적인 서비스 형태

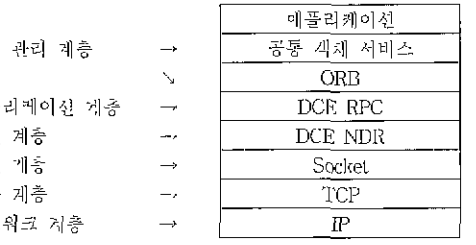
다. 또한 현재의 모든 자원의 관리는 막연한 network link protocol에 의존하고 있는 상황이며, 통신망을 이용하고자 하는 클라이언트 및 서비스를 제공하는 서버들 또한 내부적인 data 표현 및 link protocol에 의존하는 상황이며, 네트워크에 구성된 기계들의 속성에 따라 데이터의 표현 및 연결 상태가 자연스럽지가 않다. 또한 데이터의 공유에 있어서 동시성을 제공하지 못하므로 여러 클라이언트들이 동시에 데이터를 공유할 수 없다. 그러므로 본 연구에서는 각각의 노드들에 공유 가능한 클러스터를 구성하고 각각의 이기종간에 데이터의 공유 및 연결이 자연스럽고 동기신호를 이용하여 메모리의 낭비를 막는 진보된 분산 객체 복제 기법을 가진 Client/Server를 구축하고자 한다.

2.2 CORBA에 의한 객체 복제

최근에 분산 어플리케이션이나 분산 서비스와 관련된 일반적인 객체 지향 시스템에 적합한 산업 표준을 만들 목적으로 형성된 협회가 OMG(Object Management Group)이다[5]. 이 OMG 그룹에서는 소프트웨어 구성 요소들간에 원활한 운영을 할 수 있도록 표준화된 인터페이스를 위한 ORB(Object Request Broker)를 정의하였다. OMG에서는 ORB를 위한 산업적으로 표준화된 인터페이스를 제공하기 위하여 CORBA(Common Object Request Broker Architecture)명세를 만들었다. CORBA에 의해 IBM의 SOM(System Object model), HP사의 ORBplus와 Distributed Smalltalk, DEC사의 ObjectBroker, IONA의 Orbix, NCR의 Cooperation Frameworks, SunSoft의 DOE, Postmodern의 ORBeline 등 다수의 제품들이 개발 되거나 개발중에 있다[6, 7].

ORB코어에서는 객체들을 표현하고, 요청한 객체를 위한 통신을 제공한다 CORBA는 서로 다른 ORB코어들 사이에 마스크를 할 수 있는 인터페이스를 갖는 ORB코어 위에 적재요소들을 구성하게 함으로써 서로 다른 메커니즘을 갖는 객체들을 지원할 수 있다. 그러므로 ORB는 객체들 간에 요청이나 응답을 주고받는데 있어서 투명성을 제공할 수 있고 서로 다른 분산 환경이나 다중 객체 시스템등 이기종간의 어플리케이션 상에서도 상호간의 운영이 가능한 인터페이스를 제공하는 것이 가능하다.

OSI참조 모델에서 ORB와 공통 객체 서비스는 객체 관리계층에 존재하며 어플리케이션 계층의 상위 계층에 존재한다. ORB 부품은 이질적인 시스템 간에 네트워크를 통해 객체의 이식성과 상호연산(interoperation)을 지원한다. ORB의 구현을 위하여 클라이언트/서버 객체 상호 연결을 가능하게 해주는 IDL(Interface Definition Languag e)와 API를 정의하고 있다.



(그림 2) 객체 관리 계층

CORBA 2.0은 ORB들을 연결하는 브리지를 구축하는 방법을 정의하고 있다 하나의 ORB상의 클라이언트가 또 다른 ORB상의 서버를 호출하면, 새로운 DSI(Dynamic Skeleton Interface)가 목표 ORB에 요청하면, 동적 응답 인터페이스가 앞의 ORB에 있는 목표 객체를 불러낸다 이러한 동적 기술은 객체 버스로의 게이트웨어를 구축하는데 적합하다. 분산형 객체는 메소드 주문(Method Invocation)을 이용하여 원격 클라이언트에 액세스할 수 있는 이진 컴포넌트로 패키지화 할 수 있다

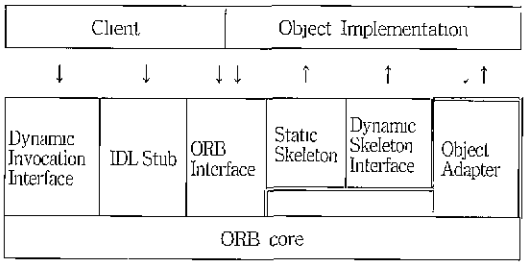
클라이언트는 서버 객체를 구축한 언어나 컴파일러의 종류와 네트워크 상의 객체의 상주 위치를 모른다. 단지 이들 클라이언트는 객체의 이름과 객체가 발생하는 인스턴스 만을 알면 된다. 결국, 클라이언트/서버 컴포넌트는 상호 작용과 협동 작업을 할 수 있는 자체 처리 능력이 필요하다.

클라이언트의 객체 호출 과정은 IDL 스터브를 통하거나 동적 호출 인터페이스인 DII(Dynamic Invocation Interface), 또는 직접 ORB 인터페이스를 이용할 수 있다.

객체 구현(object implementation)은 객체들의 메소드들을 위한 코드나 객체 속성을 위해 정의된 데이터에 의해서 객체의 의미를 부여한다 객체 구현은 클라이언트가 객체를 어떻게 호출하는 간에 ORB에 의존하지 않고 객체 아답터의 선택에 의해서 ORB에 따른 서비스를 받기 위해 인터페이스를 선택한다.

객체 아답터(object adopter)는 객체 구현이 ORB에 의해서 제공되는 서비스들로 접근할 수 있는 일반적인

방법이다. ORB는 객체 아답터를 통해서 객체 레퍼런스의 생성과 해석, 메소드 호출, 보안, 구현 객체의 활성화와 비활성화, 구현들을 위한 객체 매핑과 저장 등의 서비스를 받을 수가 있다.

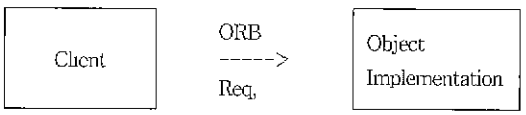


(그림 3) CORBA 인터페이스의 구조

인터페이스 저장소(Interface repository)는 런 타임시 IDL정보에 나타난 지속적인 객체에 대한 정보를 제공한다. 인터페이스 저장소의 정보를 사용하게 되면 객체에 허용해야 할 연산자가 무엇인지 결정되지 않은 프로그램이 컴파일 되어졌을 때 인터페이스를 할 수 있는 프로그램이 가능하다 또한 ORB 객체들의 인터페이스와 관련된 디버깅 정보, 스터브나 스켈톤의 라이브러리, 특정한 객체를 검색하거나 포맷 할 수 있는 루틴 등 인터페이스 저장소와 관련이 있는 부가적인 정보들까지도 저장하고 있다.

2.3 ORB의 동작

클라이언트가 객체 실행을 위해 ORB를 통하여 요청한다. 클라이언트는 객체상에서 오퍼레이션을 수행하며, 객체 구현의 주체는 코드와 데이터이다.



(그림 4) ORB 요청 절차

ORB는 요청을 받기 위하여 객체 구현을 준비하고, 요청 데이터와 통신하기 위해서 요청에 대한 객체 실행을 발견하는 메카니즘을 관장한다. ORB의 구조는 전반적으로 RPC구조와 유사하다. 차이점은 인수를 호출하는 대신 메소드를 호출한다 그리고 ORB는 동기적인 통신과 비동기적인 통신도 지원한다.

3. 객체 프로그래밍 개발 환경

객체 프로그래밍 개발과정을 모델링, 정의, 구현 단계로 구분 지을 수 있다. 객체 모델링 도구는 ORB에 의해 제공되어지지 않는다 그러므로 단순 정의 언어의 집합을 사용하여 객체 모델을 나타내는 것이다. 인터페이스 저장소는 모든 정의된 객체 인터페이스를 접근 할 수 있는 프로그램을 허용한다. 인터페이스 저장소는 정보 소스가 된다. 운영 요청을 하기 위해 클라이언트 stub 루틴을 생성하거나 메소드에 대한 skeleton 루틴을 생성한다.

첫째, 모델링 단계는 객체, 객체 클래스, 객체 운영, 객체 메소드 생성, 구문 객체 속성등의 방법론을 생성하는 단계로 업무 데이터를 기술하는 과정이다 둘째, 정의 단계에서는 모델링 단계에서 생성된 정보를 획득하는 단계로 인터페이스 정의 언어, 구현 사상 언어, 메소드 사상 언어, 구문 객체 언어를 정의하고 IDL과 MML화일은 정보 저장소에 적재시키고, IML화일은 객체 저장소로 COL화일은 구문 객체에 적재된다. 세 번째로는 구현단계로써 클라이언트와 서버로 구성된다. 이 과정이 완료되면, 애플리케이션 실행과 디버그과정을 거친다. (그림 8)은 ORB의 인터페이스를 나타낸다.

객체 인터페이스는 인터페이스 정의 언어에 의해 동적으로 정의된다. 클라이언트는 객체에 관한 객체 참조를 접근함으로써 요청을 수행한다 클라이언트는 초기화 요청에 의해 동적으로 객체에 특정한 stub루틴을 호출한다. ORB는 IDL skeleton을 통하여 객체 실행에 적당한 실행 코드, 전송 피라미터, 전송 통제를 위치시킨다. 객체 아답터를 통해서 객체 실행은 ORB로부터 서비스를 제공받는다.

3.1 ORB 클라이언트

클라이언트 IDL스터브은 IDL컴파일러로 만들어지며 클라이언트가 서버상의 해당 서비스를 어떻게 불러내는지를 정의한다. 또한 동적 주문(invocation)API의 경우에 사용자는 실행 시간으로 불러내고자 하는 서비스를 찾을 수 있고, 정의를 읽을 수 있으며, 이에 매개변수를 부여하여 호출을 발행 할 수 있고, 이에 응답을 받을 수 있다 API의 인터페이스 저장소를 이용하여 모든 등록된 컴퍼넌트 인터페이스들과 인터페이스를 지원하는 메소드들, 매개변수들의 명세를 얻을 수 있고, 이들을 갱신 시킬 수 있다. 사용자들의 프로그램은 인

터페이스 저장소의 API를 사용하여 이정보를 접근하고 갱신시킬 수 있다. CORBA에서는 전역 식별자(global identifier)를 통해서 저장소 상의 명명 일관성을 강화시켜준다. 기존 분산 체제의 원격 프로시져 호출(Remote Procedure Call:RPC)보다 더 강력하고, 유연하다.

3.2 ORB 서버

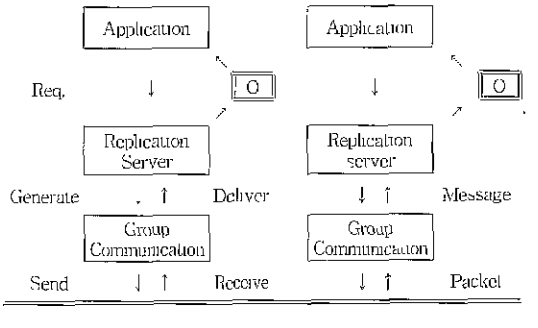
ORB는 객체 아답터를 위치시키고, 매개 변수를 보내고 IDL Stub를 통해서 객체 프로그램에 제어를 전송한다. 클라이언트상의 Stub들과 마찬가지로 이들 Stub들도 IDL 컴파일러를 사용하여 만든다. 정적 골격은 IDL로 정의된 특정 객체 클래스의 메소드들에 대하여 기계에 의해 처리되는 지원을 제공한다.

동적 skeleton 인터페이스는 메시지의 매개 변수를 점사하여 목표 객체 및 메소드를 결정하며, ORB 사이에 브리지를 구축하는데 용이하게 사용된다.

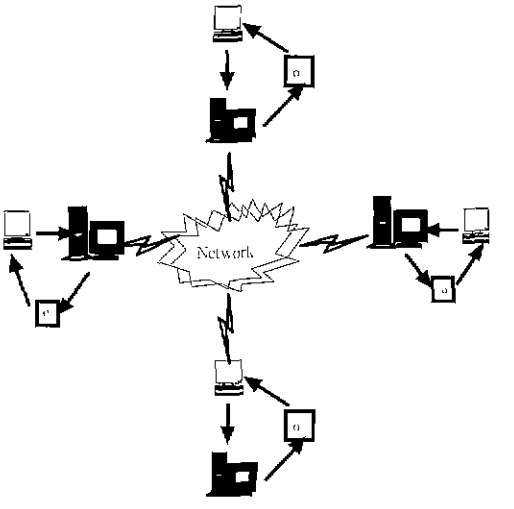
3.3 클라이언트/서버 모델에서의 객체 복제 방법

클라이언트/서버의 분산 객체 복제 프로토콜은 특정 서버의 집중으로 인한 서버의 오버헤드 문제와 통신 선로상에 발생되는 병목 현상을 방지하는 시스템이다.

(그림 5)의 그림은 데이터의 중복성을 가진 객체 복제 방법을 나타내고 있다. 각각의 클라이언트들은 각 인접 서버에게 서비스를 요청하고 각 인접 서버들은 특정 서버에게 서비스를 요청하게 되고 특정 서버는 각각의 인접 서버에게 서비스 객체를 복제(replication)하여 주게 되며, 각 클라이언트들은 인접 서버로부터 서비스를 제공받을 수 있다. 그러므로 각의 클라이언트들은 각의 인접 서버로부터 서비스를 제공받으므로 서비스 속도의 지연을 줄일 수 있고 특정 서버의 오버헤드를 줄일 수 있게되며, 병목 현상을 막을 수 있다[4]



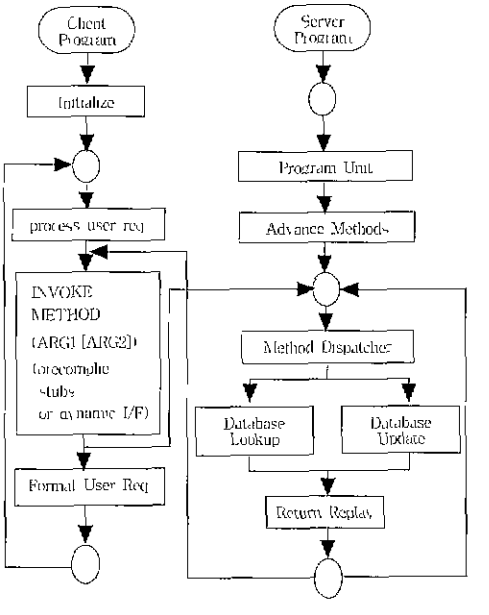
(그림 5) 클라이언트/서버 구조



(그림 6) 객체 복제 방법을 이용한 클라이언트/서버 서비스 모델

3.4 리얼타임 환경

ORB 리얼타임 환경은 (그림 7)과 같으며 클라이언트 프로세스는 ORB인터페이스 저장소 객체 모델에 근거한 논리적인 모델의 실행을 요청한다 클라이언트는 클라이언트나 서버 이름과 같이 구현에 관한 상세사항을 세분화 하지는 않지만, ORB가 올바른 서버, 네트워크 위치, 전송 프로토콜등을 결정한다.

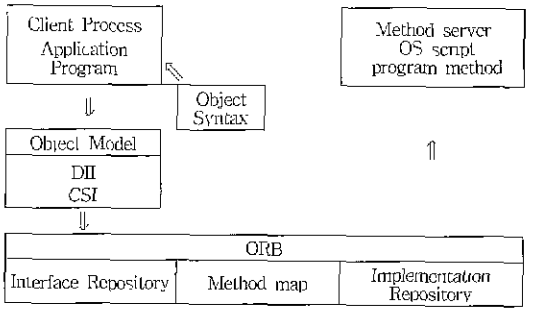


(그림 7) ORB의 흐름도

ORB는 요청 처리 서버에 요청을 적절하게 분배하며, 클라이언트에 응답을 보낸다.

4. CORBA객체의 IDL설계

CORBA의 특징 중에 가장 대표적인 것은 인터페이스 정의 언어를 사용하는 것이다. 클라이언트의 객체 호출과, 서버의 호출된 객체 수행을 정의하기 위해서 사용되어진다. IBM SOM 컴파일러를 사용하여 IDL를 작성하였으며, include부에서 메시지 클래스의 부모 클래스를 찾기 위한 파일 somcbj.idl의 위치를 알려 주고 메시지 클래스는 SOMobject 클래스로부터 파생되어진다. 인터페이스문에서는 인터페이스 이름과 상속 계층을 기술한다. 속성부에서는 인터페이스를 위한 속성들을 정의한다 메소드부에서는 이 인터페이스에 의해 받아들여진 각 메소드들을 위한 인터페이스를 정의한다. 구현부에서는 각 클래스들의 정보를 정의하며 SOM컴파일러가 클래스를 구축하기 위한 데이터 구조내의 클래스 메소드들을 위치 시키는 순서를 기술한다.



(그림 8) ORB 리얼타임 환경

```
#include <somobj.idl>
  interface Message:SOMObject
  {
     attribute string msg;
     void printMsg();
     #ifdef __SOMIDL__
     implementation
     {
        releaseorder _get_msg, _set_msg,
                     printmsg,
     };
     #endif
  };
```

(그림 9) CORBA IDL의 메시지 알고리즘

4.1 복제 객체의 Name

복제될 객체 이름을 설정하기 위해 SOMRNameable 클래스에서 제공되는 somrSetObjName과 somrGetObjName을 사용하였다

```
_somrSetObjName(receiver, ev, name);
_somrGetObjname(receiver, ev);
```

(그림 10) Objname

복제된 객체들 사이에 변경이 발생하게 되면 일관성을 유지하기 위하여 복제된 객체들에게 변경된 정보를 전달하여야 한다 이에 사용되는 Logging에는 Operation, Value를 사용한다. Logging의 초기화를 위하여 somrReplnit메소드를 사용한다. 복제된 객체는 해당 데이터를 갱신하기 전에 객체의 Lock을 수행해야만하고, 갱신 후에는 Lock을 풀어 주어야 한다. 본 논문에서 복제된 클래스는 갱신하기 전에 현재 복제 객체 상에서 Lock을 걸기 위한 메소드로 somrLockNlogOP메소드를 호출하고 갱신 작업이 수행 된 후에는 Lock을 풀기 위해서 somrReleaseNPropagateOperation메소드를 호출한다. 객체 갱신을 중지 시키기 위해서는 somrRelese-LockNabortOp메소드를 호출한다.

```
_somrLockNlogOp(receiver, ev, className,
                methodName, ap);
_somrReleaseNPropagateOperation(receiver,ev);
_somrReleaseLockNAbortOp(receiver,ev);
```

(그림 11) 객체 복제 갱신에 관한 메소드

4.2 복제 객체의 상태 설계

복제된 객체는 객체의 전체 상태를 얻고, 설정하기 위한 메소드가 필요하다. 이를 위하여 somrGETState와 somrSETState메소드가 SOMRLinearizable클래스에 의해서 제공되는데 이 메소드들은 복제된 클래스에 의해 오버라이드 되어야 한다. 복제 프레임에서는 복제된 객체 상에서 기존의 객체들과 함께 새로운 복제 객체를 동기화하기 위하여 이들 메소드를 호출한다 somrGETState메소드는 한 바이트 객체의 내부 상태를 암호화한다. 이 메소드는 기존의 복제 객체들에 의해서 호출되며 다음과 같이 표현된다. 여기에 사용되는 매개변수 buf는 객체의 내부 상태를 저장하기 위한 버퍼이다. 따라서 호출자는 문자열을 위한 저장장치를 개할당해야 하고, 처음 들어오는 4바이트는 문자열의

길이를 포함하고 있어야 한다.

```
_somrGETState(receiver, cv, buf),
_somrSETState(receiver, ev, buf),
```

(그림 12) 객체 복제 상태 메소드

somrSETState메소드는 내부 상태에 있는 바이트 문자열을 변환하기 위해서 사용되고, 새로 생성된 복제 객체들에 의해서 호출된다. somrSETState메소드에서 사용되는 매개변수 buf는 객체의 내부 상태를 위한 바이트 문자열을 포함하고 있으며, somrGETState에서와 같이 처음 4바이트의 문자열은 헤딩 문자열의 길이를 포함하게 된다.

복제된 객체는 데이터 복제 명령들을 받아들이고 응답하는 것이 가능해야 한다. 이 명령들은 비동기적으로 발생되는 임의의 상태들을 결정하기 위해서 복제 프레임으로부터 복제 객체로 전송되는 메시지이다.

```
_somrDoDirective(receiver, ev, directive),
```

(그림 13) 복제 객체의 관리 메소드

(그림 14)는 위의 알고리즘을 이용하여 Read/Write 시 의 응답시간(Response Time)과 도착 시간(Reach Time)을 얻은 결과이다. 이 실험은 EXP A, B, C로 나누어 실험하였으며, 이들에게 제공된 초기값은 다음과 같다

4.3 시뮬레이션

복제된 객체의 Read/Write에 소용되는 응답시간과 도착시간을 체크하기 위하여 실험 대상 세 가지의 경우를 설정하였다 성능 분석을 위하여 <표 1>의 A, B. C로 나누어서 각각 발생되어지는 상황 값을 설정하였다.

<표 1> 시뮬레이션 상황 설정 값

EXP	$slow_w$	opt_w	$fast_w$	$fast_{r0}$	$fast_{r1}$	$slow_r$
A	1	0	0	0-1	0	0-1
B	1	0	0	0-1	0-1	0-1
C	1	0	0	0-1	0-1	0-1

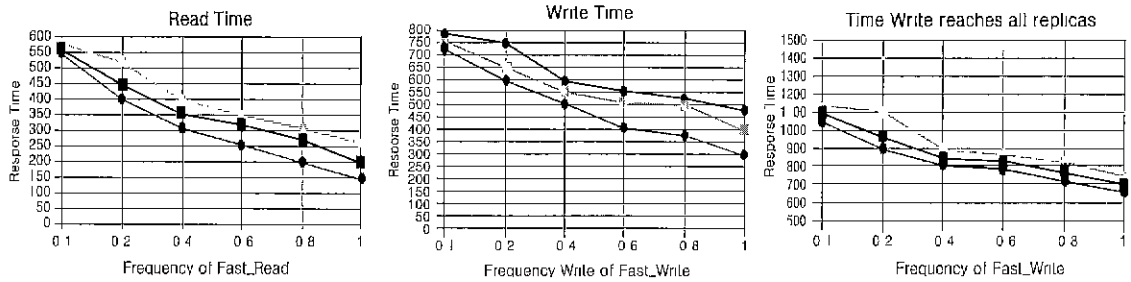
실험 A는 동적 호출 시나리오를 이용한 경우로서 read와 write에 같은 비중을 두어 read는 한번 실행하고 write 시에는 여러 번 반복 실행 될 수 있게 처리하였다 실험 B와 C는 read와 write에 대한 성능 평가치는 1997년 스탠포드 대학에서 실험한 Large Scale System의 읽기와 쓰기 방법에 제시되어 있다

복제된 객체의 정보가 최신의 정보가 아닐 때에는 환형 큐를 이용하여 정확한 객체 정보가 나타날 때까지 순환시키고자 하였다. 또한 무한정 객체 정보를 찾아다니는 무한 루프에서 빠져 나오기 위하서 환형 큐를 사용하였다

시뮬레이션 모델은 통신 네트워크에 의해 연결된 다수개의 노드들로 구성되며 각 노드들은 복제된 객체의 복사 본들을 가지고 각각의 version을 유지시킨다. 각 노드들은 비 휘발성 메모리인 디스크와 처리기를 가지고 있으며 이 처리기들은 단일 서비와 함께 단일 큐를 가지고 있다. 그러나 각 노드들은 호출 시 read/write의 우선권을 가지지 않으며 서버들은 FIFO 관리 체계로 설계되었다.

여기서 클라이언트가 요구하는 각 요구 신호는 단일 연산 Read와 Write로 구성되었으며 각 read와 write 연산에서 CPU의 평균값을 나타내는 *cpu_req*와 I/O 요구시간을 체크하는 *io_req*로 구성되어 있다.

성능평가에서 두 Fast_Read연산은 서로 다른 계층



(그림 14) READ, WRITE의 Response, Reach 시간

으로부터 읽은 결과를 평가하는데 첫 번째 연산은 Fast_Read_0을 나타내고 두 번째 연산은 Fast_Read_1을 나타낸다. 이는 현재의 계층 내에서 상위 계층의 노드로부터 복제 된 메시지를 읽어들인다.

상위 계층 노드는 부모 노드이고 파라메터 $fast_r0$, $fast_r1$ 그리고 $slow_r$는 Fast_Read_0, Fast_Read_1 연산의 평가 결과이다

실험에 사용된 각 설정 알고리즘에 제시된 0, 1, 1'의 값에 대하여 설명하면 0일 때는 복제를 수행하지 않고 상위 계층에서 읽기 작업과 쓰기 작업이 수행되어지는 경우이고 1일 때는 반대로 복제 작업을 수행하여 복제된 객체를 참조하여 상위 계층에 읽기와 쓰기 작업이 이루어진다. 마지막으로 1'는 객체가 존재하는 계층을 반식(propagation)알고리즘을 이용하여 여러 계층에 복제 작업을 반복 수행하여 가장 인접된 계층의 서버에 읽기와 쓰기 작업을 수행했다.

5. 결 론

본 논문에서 제안된 객체 복제 방법은 전통적인 분산 시스템의 형태에서 발전된 open client/sever 구조로 발전 시킬 수 있었다. 또한 애플리케이션간에 동기적인 방식의 통신을 제공하는 RPC와 동기적방식과 비동기적 방식을 통합한 ORB에 대하여 분석한 후 분산 객체 복제 기법을 설계하였다. 본 논문에서 제시한 ORB 오퍼레이션, 클라이언트/서버와 IDL을 이용한 객체 갱신에 필요한 메소드로써 통신상에서 발생되어지는 병목 현상과 자원의 복제에서 발생되어지는 오버 헤드를 객체 복제 프로토콜(Object replication protocol)을 이용하여 원격지의 서비스를 인접 서버로 객체를 복제하여 원격지의 서비스를 인접 서버로부터 서비스 받게 하여 시스템 서버 및 특정 서버의 오버헤드를 줄일 수 있었다

객체 복제를 이용하여 각 서버에서 데이터를 Read/Write시 Read에 많은 부하가 걸리는 것을 알 수 있으므로 read/write 비율을 6:4로 부여한 후 성능 분석을 시행하였다 read시에는 write보다 통신부하가 작기 때문에 read에 많은 비율을 할당해서 분석을 시행하였다. write의 비율이 read보다 높아지면 질수록 성능은 급격히 줄어든다.

본 논문에서 제시한 객체 복제 메카니즘을 이용하여 벤치마킹을 통한 성능 평가 결과를 보았을 때, 제시된 객체 복제 메카니즘에 따른 Read/Write 응답 시간 및 도착시간을 볼 때 기존의 복제 메카니즘에 비해 read 시에는 32m초와 write시에는 27m초 증가된 것으로 나타났다. 본 논문에서 제안된 객체 복제 메카니즘이 기존의 메카니즘에 비해 약 12%~14%정도 향상된 것을 알 수 있다.

Read/Write에 객체 복제 비율을 6:4로 부여 함으로 기존의 RPC에서의 액세스 효율보다 접근 서비스율을 높일 수 있었다

본 객체 복제 시스템의 연구 결과로 인하여 internet과 같이 큰 광역 통신망에서도 지역 통신망과 같이 빠르고 정확한 자원의 관리 및 공유를 가져올 수 있다

참 고 문 헌

[1] Paul J Fortier, Designs of distributed operating systems, McGraw-Hill, 1986

[2] Mark Roy, "*Buildding an ORB-based solution*," Netlinks technology, 1994

[3] Noha Adly, "Management of Replicated Data in Large scale System," Univ Cambridge, Aug 1995.

[4] Yair Amir, "Replication over a Partitioned Netwotwork," *The Hebrew Univ.Jerusalem. feb. 1995.*

[5] Jon Siegel, "CORBA Fundamentals and Programming," OBJECT MANAGEMENT GROUP, 1995.

[6] IBM, Experiences with SOMObjects : Distributed System Object Model , IBM Inc, 1993.

[7] OMG, *CORBA 2.0/Interoperability Universal network Object*, OMGTC Document 95.3 xx, mar. 1995

[8] M. B. Abbott and L. L. Peterson Increasing network thoughput by integarating protocol layers IEEE/ACM Transactions on networking, 1(1), 1993

[9] Alex berson, Client/Server Architecture, McGraw_Hill, 1991.

[10] AT&T, UNIX system V streams programming's guide. Prentice-Hill, 1987

[11] Robert Orfali and Dan Harkey, Client/Server Programming with JAVA and CORBA 2nd edition, john Wiely & Sons, Inc., 1998.

[12] Ronan Geraghty and Sean Joyce, COM-CORBA Interoperability, Prentice Hall, 1999.

윤 동 식

e-mail : yundos@mail.andong-c.ac.kr

1992년 관동대학교 정보처리학과 졸업(공학사)

1994년 관동대학교 대학원 전자계산공학과 졸업(공학석사)

2000년 관동대학교 대학원 전자계산공학과(공학박사)

1995년~1998년 아세아 항공 전문대학 정보통신학과 전임강사

1999년~현재 안동과학대학 사무자동화과 전임강사

관심분야 : 분산병렬처리, 컴퓨터네트워크, 운영체제

이 병 관

e-mail : bklee@mail.kwandong-ac.kr

1979년 부산대학교 졸업(학사)

1986년 중앙대학교 대학원 전자계산학과 졸업(이학석사)

1990년 중앙대학교 대학원 전자계산학과 졸업(이학박사)

1988년~현재 관동대학교 전자계산공학과 교수

관심분야 : 분산운영체제, CAD/CAM, 정보통신