실시간 객체지향 캐싱 시스템

김 영 재[†] · 성 호 철[†] · 홍 성 준[††] · 한 선 영[†††]

요 약

기존의 캐싱 시스템은 다른 캐싱 서버간의 통신을 통해 분산환경을 고려하였다. 그러나 이러한 캐싱 시스템은 실시간성을 제공하지 못하고 한 서버에 몰리는 부하를 처리 하는데 문제가 있다. 이러한 단점을 보완하기 위해 본 논문은 RSVP와 TAO의 ORB를 통합하여 기존 CORBA에 없는 QoS보장 메커니즘을 제공하는 RIOP(Real-Time Inter-ORB Protocol) 구현하였다. 그리고 이러한 실시간CORBA를 기반으로 객체에 대한 복제서버(Replicated Server)를 두어 부하분담처리를 하였으며 객체의 접근에 있어서 발생되는 지역성(locality)을 이용 PS에 메인 메모리 캐시를 구현하여 좀더 빠른 XCSLS(Extended Caching System for Load Balance)을 설계 구현한다.

Real-Time Object-Oriented Caching System

Young-Jae Kim[†] · Ho-Chul Sung[†] · Sung-Joon Hong[††] · Sun-Young Han[†††]

ABSTRACT

Conventional caching system doesn't support Real-Time attributes and load balance. To solve these problems, this paper describes the design and implementation of the RIOP(Real-Time Inter-ORB Protocol) to provide QoS guarantees mechanism integrating RSVP and TAO ORB. Furthermore, it provides fast XCSLS(Extended Caching System for Load Balance) implementing main memory cache in Primary Server using locality of objects. In this paper, a key feature is presented : QoS enforcement, PS(Primary Server) and RS(Replicated Server).

1. 서 론

분산 환경하에서 다양한 하드웨어와 응용 소프트웨어, 운영체제, 데이터베이스 등을 일관되게 연결하기 위한 방법으로 OMG(Object Management Group)[1]에서 객체 기술에 근거한 통합 기술인 CORBA(Common Object Request Broker Architecture)[2]가 제안 되었다. 이것은 실제로 구현된 또는 구현되는 언어에 상관없이 각 시스템들을 연결·통합할 수 있는 방법을 제공하는 미들웨어이다. 그리고 그러한 OMG의 표준을 바탕으로 실시간을 제공하는 워싱턴 대학에서 TAO(The ACE ORB)[5]라는 실시간 ORB를 개발중이다.

CORBA 서비스와 더불어, 현재 인터넷에서는 데이타의 신속한 처리를 위하여 인터넷 캐싱 메카니즘이 도입되었다. 캐싱 시스템은 "Single point of failure"의 문제를 해결하기 위해서 Harvest 캐싱 시스템 또는 Squid 캐싱 시스템[9] 등이 등장하게 되었고 현재 운용 중에 있다. 그러나 이러한 캐싱 시스템은 멀티미디어와 같은 방대한 자료를 처리하기에는 대역폭과 지연 문제 때문에 적합하지 못하다. 그러므로 이런 문제를 해결하기 위해서 CgR(Caching goes Replication)[7]이라는 개념이 제시되었고 CgR구현을 위해서 WLIS(Web Location and Information Service)[8]라는 명명 서비스가 제안 되었다. 그리고, 상용으로 Cisco의 웹 캐싱 제품 등이 소

※ 본 논문은 1997년도 한국학술진흥재단의 대학교수 해외파견 연구지원에 의하여 연구되었음.
† 준 회 원 : 건국대학교 대학원 컴퓨터정보통신공학과
†† 준 회 원 : 여주대학 컴퓨터공학과 교수
††† 정 회 원 : 건국대학교 컴퓨터정보통신공학과 교수
논문접수 : 1999년 4월 20일, 심사완료 : 1999년 8월 9일

개되고 있다. 그러나, WLIS나 웹 캐싱 제품은 URL (Uniform Resource Location) 기반의 문제로 인해서 생기는 특정 웹 서버의 부하분담을 목적으로 개발되었지만, URL 기반에서 벗어나지 못하고 있다. 그러므로, 이러한 문제점을 해결하기 위해 분산된 PS(Primary Server) 또는 RS(Replicated Server)에 각각 객체가 등록되어 있고, 클라이언트가 원하는 객체의 서버(PS 또는 RS) 위치를 몰라도 원하는 객체를 검색할 수 있는 기능과 부하분담을 구현하는 캐싱 시스템 설계 및 구현하였다.

한편 Differentiated Service는 네트웍이 정체되었을 때 상이한 사용자에 대해 상이한 레벨의 서비스를 지원하는 것이다. Differentiated Service는 서비스 레벨을 2가지로 나누고 패킷의 헤더에 Best-effort와 Premium 두 가지로 지정하였다. Ipv4에서는 8비트의 Service Type이 있고, Ipv6에서는 8비트의 Traffic class가 있어 그 필드를 사용하여 구현한다. 기존의 RSVP등에서 모든 라우터에서 RSVP를 지원해야 한다는 규모성(Scalability)이 부족하고 복잡한 메커니즘을 가지고 있다. 그러므로 좀더 단순화되고 확장성을 가지는 Differentiated Service가 제안되고 지금 활발히 연구중이다.

본 논문은 실시간 CORBA상에서의 확장된 캐싱 시스템을 설계 및 구현하였으며, 두 가지 이슈로 나누어진다.

첫번째, 본 논문은 RSVP와 TAO의 ORB를 통합하여 기존 CORBA에 없는 QoS보장 메커니즘을 IIOP 프로토콜에서 지원할 수 있도록 설계 및 구현하였다. 본 논문에서는 IIOP/RSVP 통합함으로서 새로운 프로토콜 RIOP(Real-Time Inter-ORB Protocol)를 설계 및 구현하였다. 이러한 실시간 CORBA 환경은 RSVP가 사용자의 QoS요구를 입력 받아 RSVP를 지원하는 라우터에 전달하여 QoS보장 요구를 처리하는 메커니즘을 갖는다. IIOP와 RSVP통합을 위해서 QoS_Info라는 QoS 타입을 정의하였고, 이런 QoS정보를 ORB에 전송하기위한 QoS API를 개발하였다. 본 논문의 실시간 CORBA상의 CgR/명명 서비스는 URL이 아닌 URN방식의 접근이 가능하게 하여 객체 위치 투명성과 부하분담을 제공한다.

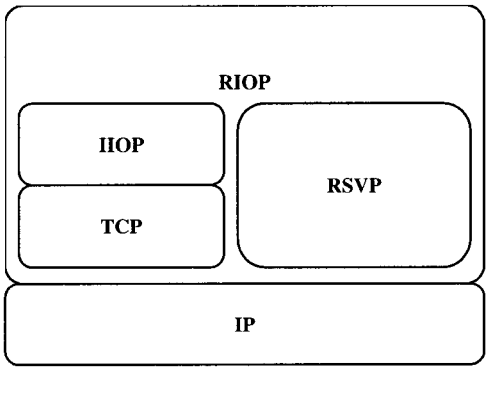
두번째, 실시간 CORBA상에서의 확장된 형태의 XCSLB(Extended Caching System for Load Balance)의 구현을 목적으로 한다. 기본개념은 RS에 통계와 클라이언트의 사용통계를 분석 많이 사용 되었던 것을 선별하여 PS의 메모리에 캐시해서 맨 처음 클라이언트가 PS에 접속했을 때 있으면 바로 처리해주는 확장된 캐싱 시스템의 구현을 목적으로 한다. 이러한 메모리 캐싱은 사용자가 요구하는 객체는 지역성(locality)을 가진다는 이론에서 출발하며, 그로인한 캐싱의 Hit 율을 높여 전체적인 시스템 속도의 향상을 도모한 것이다. 또한 이러한 XCSLB 시스템은 하부기반을 ORB 레벨의 QoS를 보장해주는 TAO를 사용 구현함으로 ORB에서 우선순위가 높은 객체를 먼저 처리해주는 실시간성을 지원한다.

본 논문은 2장에서 XCSLS(Extended Caching System for Load Balance) 설계에 대하여, 3장에서 구현에 대해 그리고 4장에서 결론을 맺는다.

2. 시스템 설계

2.1 실시간 CORBA를 위한 IIOP와 RSVP의 통합구조

본 논문은 (그림 2.1)에서 처럼 IIOP와 RSVP를 통합하여 공통된 프로토콜을 설계 및 구현하기 위해 RIOP(Real-Time Inter-ORB Protocol)이라는 통합된 프로토콜의 구조를 제안한다.



(그림 2.1) IIOP와 RSVP를 통합한 RIOP 프로토콜 스택

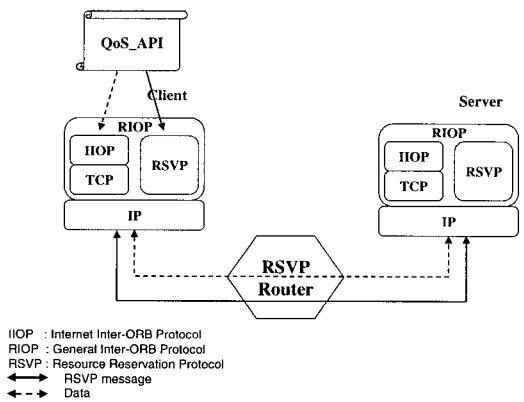
RIOP는 데이터전송 이외에 자원예약정보를 가지는 QoS정보를 전송하는 필드를 부분을 가지고 있다. 데이터전송 부분은 기존의 IIOP로 전송되고 QoS정보 부분은 RSVP와 연동된다. RIOP구조는 IIOP구조를 약간 확장하여 QoS정보를 전송하기 위한 필드를 추가하여 데이터와 함께 전송이 된다. QoS정보를 가진 필드는 service_context라는 필드이다.

```
Struct QoS_Info
{
    long bandwidth;
    long latency;
};
```

(그림 2.2) QoS 명세서

본 논문에서는 (그림 2.2)과 같이 QoS보장을 위한 대상을 대역폭과 지연성으로 정의하였다. 이런 QoS파라메터를 정의한 것은 IETF IntServ의 Guaranteed Service에 기초하고 있다.

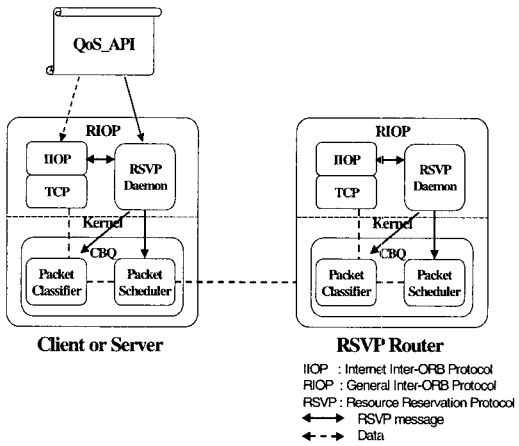
(그림 2.3)는 QoS 보장 메커니즘을 고려하여 IIOP와 RSVP 통합 구조를 보이고 있다. 사용자의 QoS 보장 요구는 CORBA응용으로부터 RIOP내의 RSVP로 전송된다. QoS 요구를 데이터 전송과는 별도로 자원 예약 시그널을 각 RSVP 지원 라우터에게 전송하게 한다. CORBA와 통합된 RSVP는 QoS 정보를 RSVP 지원 라우터에게 전달한다. RSVP 지원 라우터는 QoS 파라메터를 받아 자원을 예약한다. 이런 클라이언트와 라우터간의 자원예약 기능을 이용하여 사용자는 IIOP (Internet Inter-ORB Protocol)를 통해서 데이터가 전송되는 도중에라도 요구하는 QoS 파라메터 값을 조절할 수 있다.



(그림 2.3) IIOP와 RSVP를 통합한 QoS 보장 메카니즘 구조

(그림 2.4)은 (그림 2.3)의 클라이언트와 서버 등의 시스템과 RSVP 지원 라우터와의 관계를 보이고 있다. 본 논문에서 CORBA와 통합된 RSVP는 RSVP를 지원하는 라우터와 동작을 하면서 Admission Control 기능과 Policy Control 기능을 담당한다. 그리고 각 호스트 커널 부분과 RSVP를 지원하는 라우터의 커널에 CBQ (Class Based Queueing)라 불리는 패킷 분류자와 패킷 스케줄러 모듈이 설치되어 있어서 요구하는 QoS 보장을 달성하는 기능을 담당한다.

본 RSVP와 IIOP의 통합은 TAO의 ORB를 기반으로 RSVP를 통합한 것을 말한다. 본 IIOP와 통합된 RSVP는 클라언트로부터 서버의 객체에까지 RSVP를 지원하는 라우터를 경유해서 QoS 파라메터를 전송할 수 있게 한다. RSVP의 자원 예약 메세지는 QoS 파라메터(대역폭/지연시간)를 표현하는 속성을 포함한다.



(그림 2.4) 시스템과 라우터의 관계

본 논문에서 기반으로 한 TAO의 ORB에는 RIOP (Real-Time Inter-ORB Protocol)라 불리는 실시간 ORB 프로토콜이 있지만, TAO RIOP의 QoS 파라메터는 QoS 보장을 위해 각 종단 시스템에만 QoS보장을 지원하려고 하기 때문에 효과적인 메카니즘이 없이 전달된다.

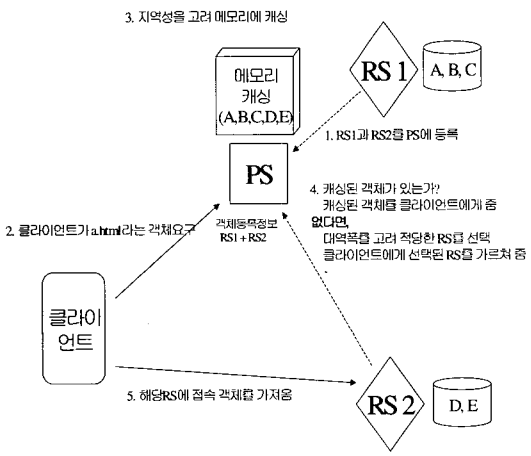
그러나 본 논문은 실시간성을 가지는 RIOP프로토콜을 제안한 것으로 기존CORBA와 통합된 RSVP의 QoS 파라메터는 QoS 보장 메카니즘을 가진 RSVP를 지원 라우터로 전송될 수 있다. 즉 각 종단 시스템 뿐만 아니라 망에서도 RSVP를 통해 QoS를 지원할 수 있는 메커니즘을 갖게 하는 것이다. 여기서 RSVP라우터는 호스트와

라우터간의 QoS 보장 메카니즘인 Admission Control 기능, 패켓 분류기능 그리고 패켓 스케쥴링 기능을 가지고 있다. 본 논문에서 정의된 QoS 파라메터 타입(대역폭과 지연성)은 정의된 타입을 실제로 수행시킬 수 있는 내부 메커니즘을 필요로 한다. 그러므로 본 논문은 이런 QoS 파라메터 타입에 대한 실제 수행 부와 연결시켜주는 QoS_API를 정의하였다.

CORBA 에게 QoS Specification을 제공할 수 있도록 하기 위해서 QoS_API는 QoS 타입정의와 QoS 보장을 지원할 수 있는 메커니즘을 연결해 주는 연결부 역할을 하는 것이다. QoS_API는 QoS_Info라 불리는 QoS 타입 정의 부분과 그 QoS 정보를 RSVP에 전송 해 주는 부분으로 구성되어있다. QoS_API내에 QoS_Info는 QoS 타입정의를 위해서 사용되었다. 그리고 RSVP는 내부적으로 QoS 타입에 대한 QoS 보장 메커니즘으로 사용된다. 즉 QoS_API는 CORBA환경에서 QoS를 RSVP에 넘겨주는 것이다.

2.2 XCSLB(Extended Caching System for Load Balance)의 설계

본 논문은 IIOP와 RSVP를 통합하여 분산환경에서 객체지향 개념을 적용한 실시간CORBA를 기반으로 하였다. 그리고 객체에 대한 복제 서버인 RS(Replicated Server)를 두어 부하분담을 처리해주고, 객체 접근에 있어 발생되는 지역성을 고려 PS(Primary Server)에 메인 메모리 캐싱을 구현한 확장된 XCSLS(Extended Caching System for Load Balance)을 설계하였다.



(그림 2.5) XCSLB설계

(그림 2.5)는 전체 시스템을 그림으로 나타낸 것으로, 각 기능을 살펴보면, PS(Primary Server)가 캐싱 서버 Squid가 저장하는 캐싱된 객체를 RS서버인 RS1 RS2등의 여러 복제서버(RS)에 복제(Replicate)하고 클라이언트의 요구에 대해 복제서버를 적절히 선택하여 부하분담(Load Balance)을 한다.

여기서 RS의 기능은 여러 클라이언트가 요구한 객체에 대한 정보를 access.info에 저장하고 통계를 내서 가장 많이 참조되었던 객체를 PS에 전달한다. PS는 RS들이 준 객체들을 대용량의 RAM에 캐시하여 두고 있다가 클라이언트가 요구한 객체가 메모리 캐시에 있으면 바로 주고 없으면 기존에 구현된 것처럼 RS에 연결해서 RS에게서 객체를 가져가는 구조를 가진다.

동작과정을 살펴보면, 웹 클라이언트가 PS에게 a.html이란 객체를 요구하면, PS 캐시서버는 해당 객체를 먼저 메모리 캐시에 저장되어 있는지 확인한다. 메모리 캐시에 저장되어 있으면 바로 클라이언트에게 돌려주고, 없으면 여러 RS중에 대역폭이 가장 좋은 것을 찾아낸 다음 클라이언트에게 찾은 RS의 정보 즉 RS의 ID를 준다. 클라이언트는 이런 RS의 정보를 받아 해당 RS에 접속한 다음 원래 원하던 객체를 요구한다.

RS는 클라이언트에게 자신에 등록되어있는 해당 객체의 참조(Reference)를 넘겨주면 클라이언트는 이 참조를 통해 해당 객체를 액세스한다.

이렇게 되면 CORBA의 구현 객체인 PS는 클라이언트의 요구에 대해 위치에 대한 투명성을 제공할 수 있게 된다. 이렇게 함으로서 본 실시간 캐싱시스템은 Squid가 갖고 있는 웹 캐싱에 대한 분산 계층화를 지원하였다. 또한 한 서버에 몰리는 경우에 여러 개의 RS서버를 두었고, PS의 부하 분담 기능을 좀더 확장하여 PS에 메모리 캐시를 해서 부하분담 기능을 효과적으로 확장한 XCSLB(Extended Caching System for Load Balance)를 설계를 목적으로 한다.

그리고 본 논문에서 설계를 크게 메모리 캐시를 가지고 있는 PS, 복제 서버인 RS 그리고 서비스를 요구하는 클라이언트 세 부분으로 나누었다.

2.2.1 Interface Definition

향상된 캐싱 시스템XCSLB를 구현을 위해 (그림 2.6)과 같이 PS와 RS 서버를 정의하였다.

```
interface PS{
    string Get_RS_IP();
    string Get_Mem_Obj();
    void Put_RS_IP(in string RS_IP);
    void Put_RS_Obj(in string RS_Obj);
};
interface RS {
   string Get_RS_Obj();
   void Ping();
};
```

(그림 2.6) 실시간 캐싱 시스템의 IDL

2.2.2 Primary Server의 메소드들

PS의 기능으로 다음 4개의 메소드들(Get_RS_IP(), Get_Mem_Obj(), Put_RS_IP(), Put_RS_Obj())을 정의한다.

(1) string Get_RS_IP()

선택된 RS의 IP를 반환하는 메소드로, .클라이언트가 자신이 접속할 RS의 IP를 얻기 위해서 호출한다.

(2) string Get_Mem_Obj()

요구된 객체를 반환하는 메소드로, 클라이언트가 PS의 메모리상에 캐시되어있는 객체를 가져올 때 호출한다.

(3) void Put_RS_IP(in string RS_IP)

인자로 전달된 IP(RS_IP)를 PS가 자신이 관리할 RS로 등록시키는 메소드로, RS가 처음 시작될 때 자신의 IP를 PS에 등록시키기 위해서 호출한다.

(4) void Put_RS_Obj(in string RS_Obj)

인자로 전달된 객체(RS_Obj)를 PS의 메모리에 캐싱하는 메소드이다. 클라이언트는 자신이 가지고 있는 객체가 요구되는 횟수에 대한 정보를 가지고 있다가 이 횟수가 특정 수치를 넘게 되면 해당 객체를 RS의 메모리에 등록시키기 위해서 이 메소드를 호출한다.

2.2.3 Replicate Server의 메소드

RS의 기능으로 다음 2개의 메소드들(Get_RS_Obj(), Ping())을 정의한다.

(1) string Get_RS_Obj()

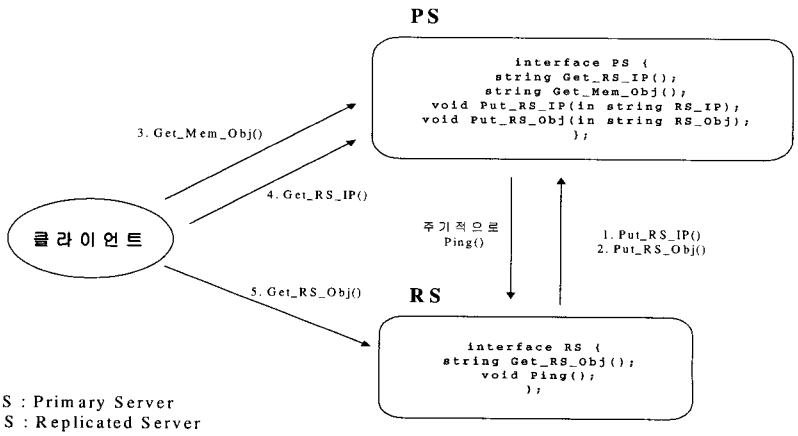
RS에 캐시 되어있는 객체들 중에서 요구된 객체를 반환하는 메소드이다. 클라이언트가 처음 PS에 객체를 요구했을 때, 해당하는 객체가 PS의 메모리에 없으면 클라이언트는 해당 객체를 RS에 요구하게 된다. 이때 클라이언트가 RS에 있는 객체를 요구할 때 이 메소드를 호출한다.

(2) void Ping()

PS는 클라이언트로부터 객체 요구가 들어왔을 때 해당 객체가 없으면, 자신이 관리하고 있는 RS들 중에서 통신상태가 가장 원활한 RS를 선택하여 클라이언트가 객체를 요구할 수 있도록 한다. 이를 위해 PS에서는 각 RS간의 대역폭 상태를 수시로 점검하여 이에 관한 정보를 가지고 있게 된다. 이때 PS가 각 RS간의 대역폭을 검사하기 위해서 이 메소드를 호출한다.

2.2.4 전체적인 개념도

(그림 2.7)는 XCSLB의 전체적인 IDL 개념도를 나타낸 것이다. 동작은 다음과 같다.



(그림 2.7) XCSLB IDL의 개념도

1. 각 RS는 자신의 IP를 Put_RS_IP() 메소드를 통해 PS에게 준다.
2. 그리고 RS가 가진 객체를 PS에게 Put_RS_Obj()를 통해 PS에게 준다.
3. 그리고 주기적으로 PS는 Ping() 메소드를 통해 각 RS와의 속도를 측정해 둔다.
4. 클라이언트가 PS의 Get_MemObj() 메소드를 호출함으로서 메모리에 객체가 캐시되어 있는지 본다.
5. 객체가 있다면 바로 클라이언트에게 해당 객체를 준다.
6. 해당 객체가 캐시되어있지 않으면 Get_RS_IP() 메소드를 통해 네트웍상에서 가장 빠른 RS를 알아낸다.
7. 마지막으로 클라이언트는 해당RS에 있는 Get_RS_Obj() 메소드를 호출하여 해당 객체의 참조(reference)를 가져온다.

3. XCSLB(Extended Caching System for Load Balance)의 구현

본 연구를 통해 우리는 PS에서의 메모리 캐싱 기능에 있어 좀더 빠르면서도 실시간 CORBA위에서 자원예약을 통해 멀티미디어 서비스가 가능한 XCSLB를 구현하였다.

3.1 구현

3.1.1 PS의 구현

```
CachingPS::CachingPS(void) {
        // this->RS_number 와 this->Obj_number를 0으로 초기화
}
void CachingPS::orb(CORBA::ORB_ptr o) {
        this->orb_ = CORBA::ORB::_duplicate(o);
}
char* CachingPS::Get_RS_IP(CORBA::Environment &env){
        // RS_Info[i].bandwidth의 값이 최소인 RS_Info[i].RS_IP를 return
}
char* CachingPS::Get_Mem_Obj(const char*name, CORBA::Environment &env){
        // this->Mem_Caching[i].Obj_name이 name이 같으면
        this->Mem_Caching[i].ref++;
        return this->Mem_Caching[i].Mem_Obj;
}
char* CachingPS::Put_RS_IP(const char*RS_IP, CORBA::Environment &env){
        // RS_number가 3보다 작을경우
        // RS_IP를 this->RS_Info[RS_number].RS_IP에 넣고,
        RS_number++;
}
char* CachingPS::Put_RS_Obj(const char*name, const char *RS_Obj,
CORBA::Environment &env){
        // if (Obj_number > 9 ) then
        // this->Mem_Caching[i].ref가 최소인 객체를 제거하고
        // this->Mem_Caching[i]에 새로운 객체를 저장
        // if (Obj_number < 10) then
        // this->Mem_Caching[Obj_number]에 새로운 객체를 저장하고,
         Obj_number ++;
}
```

(그림 3.1) PS의 구현

PS는 각 RS의 정보를 위한 RS_Info구조체와 메모리 캐시를 위한 Mem_Caching 구조체를 멤버 변수로 가지고 있다. RS_Info의 RS_IP필드는 각 RS들의 ip주소를 저장하고 있다. Mem_Obj의 name 필드는 캐시될 객체의 이름을 ref 필드는 참조된 횟수를 각각 저장하고 있다.

3.1.2 RS의 구현

```
void CachingRS::orb(CORBA::ORB_ptr o){
        this->orb_ = CORBA::ORB::_duplicate(o);
}

char* CachingRS::Get_RS_Obj(const char*name,
                        CORBA::Environment &env){
        // name에 해당하는 객체를 open 한 후
        // buff로 읽어 들인후
        return buff;
}

void CachingRS::Ping(CORBA::Environment &env){
        // 적당한 loop를 돈다
}
```

(그림 3.2) RS 구현

RS객체의 Ping메소드는 PS에 의해 호출되며, 이때 Ping 메소드는 단순히 loop를 돈다. PS쪽에서는 Ping을 호출하기 시작한 시간과 완료된 시간을 체크하여 각 RS와의 response time을 체크함으로써 대역폭을 체크한다.

3.1.3 클라이언트의 구현

```
// 클라이언트 객체를 orb에 등록
// PS와 RS객체 생성
CachingPS PS;
CachingRS RS;

if(buffer = PS::Get_Mem_Obj(name)) {
// buffer의 내용을 출력하고 종료
}

buffer = PS::Get_RS_IP;
// buffer에 있는 IP로 접속, 객체를 요구

if(buffer = RS::Get_RS_Obj(name)) {
// buffer의 내용을 출력하고 종료
}

// 종료
```

(그림 3.3) 클라이언트의 구현

3.2 실시간 캐싱 시스템 PS의 실행

본 캐싱 시스템은 (그림 3.4)에서 처럼 먼저 PS를 실행하여 각 RS들의 등록을 기다린다. 이렇게 함으로 중앙적인 관리를 할 수 있다.

```
[omega] PS &
Waiting the request
```

(그림 3.4) PS 실행

3.3 실시간 캐싱 시스템 RS 실행

캐싱 시스템 서버 프로그램을 (그림 3.5)과 같이 실행하면 지정된 디렉토리의 모든 파일들에 대한 정보(파일 이름, 파일이 있는 디렉토리)를 가진 객체들이 네이밍 서비스에 등록이 되고 클라이언트의 요구를 기다린다. 이렇게 RS가 실행되면 자동적으로 PS에 RS가 등록된다. (그림 3.5)은 이런 과정을 보여준다.

```
[cclab] RS &
***** Registering following objects. *****
0 : frog.jpg
1 : testing.avi
2 : biogra.avi
3 : planet.bmp
4 : michael.jpg
5 : a.html
6 : srv_img.jpg
7 : clt_img.jpg

****** Sever ready to serve ******
```

(그림 3.5) RS실행

(그림 3.6)에서 처럼 PS는 RS가 서버가 실행하면 자신을 PS에 등록하고 캐시된 객체를 등록하고 화면에 출력한다. 그리고 클라이언트쪽의 요구를 기다린다.

```
[omega] PS &
Waiting the request

RS1 was registered
RS2 was registered
RS3 was registered
```

(그림 3.6) RS가 등록할 때 PS의 반응

(그림 3.7)은 자바 클라이언트가 a.html 객체를 요구했을 때 캐싱 시스템 서버 프로그램의 반응이다.

```
Get a request from client
Activating the object that client had requested.
The name of object : ./a.html
```

(그림 3.7) a.html 객체 요구에 대한 서버처리 출력

(그림 3.7)는 (그림 3.5)에서 RS가 실행하고 난 뒤 클라이언트의 요구 즉 a.html 객체를 요구 했을 때 RS나 PS의 반응이다. 맨 먼저 클라이언트가 요구한 객체의 정보 즉 이름과 디렉토리를 출력한다.

3.4 캐싱 객체 서버의 클라이언트 실행 프로그램

(그림 3.8)은 클라이언트가 *a.html* 객체를 요구 했을 때 화면에 나타나는 결과를 보여준다. (그림 3.8)는 클라이언트가 객체 이름을 *a.html*라고 명시하면 해당 객체를 서버 프로그램에게 요구하고, 요구된 객체의 참조를 서버 프로그램으로부터 받아 그 참조를 가지고 화면에 출력하는 과정이다.

```
[kkucc]client a.html
Waiting the response

This is a.html
This is result of request from client
And the  object is returned by Servers
```

(그림 3.8) 클라이언트가 a.html 객체 요구

4. 결 론

본 연구는 기존의 캐싱 시스템에서의 문제점 즉 실시간성을 제공하지 못하고 한 서버에 몰리는 부하를 PS에 메모리 캐시를 두어 처리하는 것에 초점을 맞추었다. 이러한 향상된 부하분담을 적용하기위해 본 논문은 RSVP와 TAO의 ORB통합 RIOP를 구현하여 기존 CORBA에 없는 QoS보장 메커니즘을 제공하도록 실시간성을 제공하였다. 그리고 이러한 실시간 CORBA를 기

반으로 객체에 대한 복제서버(Replicated Server)를 두었고, 지역성(locality)를 이용하여 여러 RS에서의 클라이언트의 접속통계를 내어 가장 많이 사용되었던 객체를 미리 PS에 저장해 메모리 캐싱을 구현하여 좀더 빠른 XCSLS(Extended Caching System for Load Balance)을 설계 및 구현하였다. 이러한 XCSLB는 통계를 바탕으로 메모리 캐시를 PS에 추가함으로 Hit율을 높였고, 캐시 저장소가 HDD가 아닌 대용량의 RAM을 사용함으로 객체 접근속도를 향상시켰다.

참 고 문 헌

[1] CORBA OMG site, <URL : http://www.omg.org/>.

[2] Overview of CORBA, http://www.cs.wustl.edu/~schmidt/corba-overview.html.

[3] P.P. White, "RSVP and Integrated Service in the Internet : A tutorial," IEEE Communications Magazine, pp.100-106, May 1997.

[4] Real-Time CORBA, http://siesta.cs.wustl.edu/~schmidt/corba-research.html.

[5] Douglas C. Schmidt, Aniruddha Gophale, Tim Harrison, and Guru Parulker, "A high-performance end system architecture for real-time CORBA," IEEE Communication Magazine, Feb. 1997, pp.72-77, available at http://www.cs.wustl.edu/~schmidt/TAO.html.

[6] D.C. Schmidt, "ACE : an Object-Oriented Framework for Developing Distributed Applications," Proceeding of the 6th USENIX C++ Techical Conference, (Cambridge, Massachusetts), USENIX Association, April 1994.

[7] Michael Baentsch, Introducing Application-level Replication and Naming into today's Web, Fifth International World Wide Web Conference, Paris, France, 1996.

[8] WLIS(Web Location and Information System), Michael Baentsch, <URL : http://www.uni-kl.de/ AG-Nehmer/GeneSys/WLIS/>.

[9] Squid Caching Proxy, <URL : http://squid.nlanr.net/Squid/>.

[10] Jon Siegel, CORBA Fundamentals and Programming, John Wiley & Sons Inc, pp.1-111, 1996.

[11] Thomas J. Mowbray, CORBA Design Pattern, John Wiley & Sons Inc, 1997.

[12] Orfali Harkey, Client/Server Programming with JAVA and CORBA, Willy Computer Publishing, 1996.

[13] K. Sollins, L. Masinter, Functional Requirements for Uniform Resource Names : Internet RFC 1737, http://www.w3.org/pub/WWW/Addressing/rfc1737.txt, December 1994.

[14] C.Hui fema, Routing in the Internet, Prentice Hall, 1995, pp.279-310.

[15] S.J. Hong, et al , Object Oriented Real-Time CORBA Naming Service on Distributed Environment, The 12th International Conference on Information Networking(ICOIN-12), (Tokyo Japan), Jan. 1998, pp.637-640.

[16] S.J. Hong, et al, Real-Time Inter-ORB Protocol on Distributed Environment, International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98), Apr. 1998, pp.449-456.

김 영 재

e-mail : yjkim@cclab.konkuk.ac.kr

1996년 관동대학교 전자계산학과 졸업(공학사)

1998년 건국대학교 컴퓨터정보통신공학과 졸업(공학석사)

1998년~현재 건국대학교 컴퓨터정보통신공학과 박사과정 재학 중

1998년~1999년 6월 안산 공업대 시간강사

1999년~현재 세종대학교 시간 강사

1999년~현재 서울여대 시간 강사

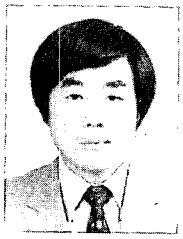
관심분야 : CORBA, Internet Caching, Mobile IP, Internet Protocol

성 호 철

e-mail : bullyboy@cclab.konkuk.ac.kr
1998년 건국대학교 전자계산학과 졸업(공학사)
1998년 건국대학교 컴퓨터정보통신공학과 석사과정 입학
관심분야 : 분산객체, Internet Caching, Mobile IP

홍 성 준

e-mail : sjhong@mail.yeojoo.ac.kr
1991년 경원대학교 전자계산학과 졸업(공학사)
1993년 건국대학교 전자계산학과 졸업(공학석사)
1998년 건국대학교 컴퓨터 정보통신공학과 졸업(공학박사)
현재 여주대학 컴퓨터공학과 교수
관심분야 : 컴퓨터 네트워크, 멀티미디어 시스템, 차세대 인터넷 프로토콜



한 선 영

e-mail : syhan@cclab.konkuk.ac.kr
1977년 서울대학교 계산통계학과 (학사)
1979년 한국과학기술원 전산학 석사
1988년 한국과학기술원 전산학 박사
1981년~현재 건국대학교 컴퓨터정보통신공학과 교수
1989년~1990년 미국 Maryland대 컴퓨터 과학과 객원부교수
1991년~1997년 금융결재원 자문교수
1990년~1997년 한국과학기술원 인공지능 연구센터 참여교수
1992년~1997년 개방형 컴퓨터 통신 연구회 TG-VT 의장
1991년~1993년 한국정보과학회 정보통신연구회 부위원장
1990년~1997년 ISO/IEC JIC1/SC21 WG8 위원장(국내 위원회)
1995년~1997년 개방형 컴퓨터 통신 연구회 TG-Web의장
1995년~1997년 건국대학교 산업기술연구소 정보통신연구센타 소장
1996년~1998년 개방형 컴퓨터 통신 연구회 총무이사
1997년~현재 한국 인터넷 협회 기술위원회 위원
1997년~1998년 건국대학교 정보통신원 교육지원 센터 소장
1998년~1999년 미국 Maryland 대학교 컴퓨터 과학과 객원교수
1998년~현재 개방형 컴퓨터 통신 연구회 이사
관심분야 : Real-Time CORBA, Internet Caching, 차세대 인터넷 프로토콜, Mobile IP