論文2001-38CI-4-7

실시간 CORBA의 우선순위 모델 구현

(An Implementation of Priority Model of Real-Time CORBA)

朴 順 禮 *, 鄭 善 太 **

(Sun-Rei Park and Sun-Tae Chung)

요 약

현재의 CORBA는 처음부터 비즈니스 환경의 클라이언트/서버 분산 컴퓨팅 환경 구축을 염두에 두고 주로 개발되어 왔기 때문에, 실시간성 지원이 필요한 분산 제어 시스템 등의 분산 실시간 시스템 구축에 사용되기에는 부족한 점이 많다. CORBA의 실시간성 개선을 위한 많은 연구가 진행되어 왔으며, 이러한 연구에 힘입어 최근 OMG에서는 실시간 CORBA 사양을 발표하였다. 실시간 CORBA는 기존 CORBA의 확장으로 명세되고 있으며, '종단간 예측성' 를 지원하는 CORBA ORB 구현을 위한 표준을 제공하고자 한다. 종단간 예측성 지원을 위해, 실시간 CORBA는 우선순위 모델, 통신 프로토콜 구성, 쓰레드 관리 등을 지원하는 명세를 표준화하고 있는데, 이중 가장 중요한 요소는 클라이언트 요청 처리에 우선순위를 부여하여 처리할 수있도록 지원한 '우선순위 모델' 명세의 지원이다. 본 논문은 실시간 CORBA의 우선순위 모델을 설계하고 구현한 결과를 제시한다. 구현은 오픈소스인 비실시간 ORB 인 omniORB2 (v. 3.0.0)을 기반으로 확장한 형태로 이루어 졌다. 구현된 우선순위 모델의 실시간성 개선 결과의 분석은 지연시간과 지터의 측정으로 성능과 예측성을 각각 비실시간 ORB와 비교하였다. 실험결과, 구현된 실시간 CORBA 우선순위 모델 구현의 실시간성 개선을 확인할 수있었다.

Abstract

The Current CORBA shows some limitations for its successful deployment in real-time system applications. Recently, OMG adopted Real-Time CORBA specification, which is defined as an extension to CORBA. The goal of the Real-Time CORBA is to provide a standard for CORBA ORB implementations that support 'end-to-end predictability'. In order to support 'end-to-end predictability', Real-Time CORBA specifies many components such as priority model, communication protocol configuration, thread management, and etc. Among them, 'priority model' is the most important mechanism for avoiding or bounding priority inversion in CORBA invocations. In this paper, we present our efforts on a design and implementation of the Priority Model in Real-Time CORBA specification. The implementation is done as an extension of omniORB2(v.3.0.0), a popular open source non real-time ORB. Experiment results demonstrate that our priority model implementation shows better performance and predictability than the non real-time ORB.

* 正會員, 영보시스템
(Youngbo System)
** 正會員, 崇實大學校 情報通信電子工學部
(Dept. of Electronics Engineering, Soongsil University)
接受日字:2000年11月1日, 수정완료일:2001年6月4日

Ⅰ. 서 론

미들웨어는 운영체제와 응용 사이에 위치하여, 클라이언트의 요구를 서버에 전달하는 기초적인 통신 기능뿐만 아니라, 서버 위치 파악, 파라미터 마샬링/역마샬

링, 요구처리 역다중화(demultiplexing), 오류 회복, 안전(security) 등의 기능 등을 제공한다. 따라서, 소켓과 같은 저 수준 레벨의 메커니즘을 이용하여 분산 응용을 개발하는 경우 보다, 개발이 비교적 신속하며 에러가 덜 발생하기 쉬우며, 이식성이 좋다.[4] 또한, 미들웨어는 명명(naming) 서비스, 트랜잭션 서비스, 이벤트 서비스 등 분산응용 개발에 유용하게 사용되는 시스템적 기능을 추가적인 서비스로 제공하여, 분산 응용 개발이 보다 신속하고 편리하게 이루어지도록 지원해준다. 이미 알려진 여러 미들웨어 가운데, OMG(Object Management Group)가 규정한 CORBA(Common Object Request Broker Architecture)는 객체 지향 방식이 갖는 장점들을 지원하면서, 플랫폼에 독립적이며, 사용 프로그래밍 언어에 독립적인 미들웨어 환경을 지원하므로, 현재 분산 컴퓨팅 환경의 미들웨어 표준으로 자리를 잡아가고 있다.[2,4,9] 또한, CORBA가 제공하는 표준 통신 방식인 동기 호출은 '클라이언트/서버'의 '요구/응답' 문맥에 잘 부합되어 최선(best efforts)의 성능을 기대하는 분산 환경 응용에 성공적으로 이용되어 왔다.[2,4,9] 이에 따라, 산업용 네트워킹 시스템과 같은 분산 실시간 시스템에서도 CORBA를 활용하고자 하는 연구가 수행되어 왔다.[3,5]

그런데, 현재의 CORBA(버전 2.3)[11]는 비즈니스 환경의 클라이언트/서버 분산 컴퓨팅 환경 구축을 염두에 두고 주로 개발되어 왔기 때문에, 실시간성 지원에 대한 고려가 미약하여 실시간 시스템 응용 구축에 성공적으로 사용되기에는 부족한 점이 많다. 실시간 시스템은 태스크 처리가 제시간(real-time)에 수행이 끝마쳐지지 않으면, 시스템 수행 성능에 영향을 받는 시스템을 말하는 데, 이러한 실시간 시스템에서는 수행되어야 할 태스크(또는 메시지)가 제시간에 수행이 완료되는 것(또는 제시간에 보내지는 것)이 보장되기 위해서는 각 태스크 (메세지) 처리에 우선순위가 지원되어야 한다.[8] 실시간성 지원의 관점에서 현재 CORBA 규약에서 가장 부족한 것은 실시간 응용들이 요구 QoS를 ORB에 전달하고, 요구 QoS 달성을 위해 ORB가 자원을 적절하게 구성하고 제어할 수있도록 하는 표준 인터페이스가 없다는 점이다. 예를 들어, 현 CORBA에서는 클라이언트가 서버 객체 메소드를 호출할 때와 클라이언트의 요청을 서버에서 처리할 때 모두, 처리가 우선순위를 존중하여 수행되도록 요구할 수 있는 API 명세가 없으며, 이러한 QoS 처리 능력이 어떻게 ORB(Object Request Broker)에서 지원되어야 하는 가에 대한 명세가 없다. 현재 구현된 대부분의 ORB에서는 클라이언트 요청 및 서버에서의 요청 처리가 FIFO로 이루어진다. 따라서, 보다 긴급한 클라이언트 태스크의 처리가 먼저 요청된 낮은 우선순위 태스크 처리에 밀리게 되는 우선순위 역전(priority inversion) 현상이 발생하여, 긴급한 태스크나 통신이 우선적으로 처리되지 못하므로, CORBA를 분산 제어 시스템, 실시간 주식 거래 등의 분산 실시간 시스템의 미들웨어로 사용하는 경우에, 성능의 열화가 초래될 수있다. 이에 따라, CORBA의 실시간성 분석 및 개선에 대한 연구가 활발히 진행되어 왔다.[6,7,10] 최근 OMG에서는, 이러한 연구 결과에 힘입어, 분산 실시간 시스템 구축에 효과적으로 사용될 수 있는 실시간 CORBA 규약을 발표하였다.[13]

실시간 CORBA 규약은 (비실시간) CORBA의 확장으로 명세되며, 그 목표는 '종단간 예측성(end-to-end predictability)' 를 지원하는 CORBA ORB 구현을 위한 표준을 제공하는 것이다. 실시간 CORBA 규약에서 말하는 '종단간 예측성' 은 우선순위가 높은 클라이언트 호출의 처리 또는 우선순위가 높은 서버 객체에 대한 호출의 처리가 종단간에 먼저 처리되는 것을 의미한다.[13] 종단간 예측성 지원을 위해 실시간 CORBA는 우선순위 모델, 통신 프로토콜 구성, 쓰레드 관리 등을 지원하는 규약을 표준화하고 있는 데, 이 가운데 가장 중요한 요소는 '우선순위 모델' 의 지원이다. 실시간 CORBA가 규약하는 우선순위 모델에는 '클라이언트 전달 우선순위 모델' 과 '서버선언 우선순위 모델' 의 두 가지가 있다. 클라이언트 우선순위 모델에서는 클라이언트 측에서 지정한 우선순위가 서버 측에서도 존중되어 클라이언트의 요구가 처리되며, 서버선언 우선순위 모델에서는 서버에서 서버구현객체(servant)를 우선순위를 두어 등록할 수 있도록 지원하며, 이 서버 객체를 호출하는 클라이언트 요구 처리가 등록된 서버 객체의 우선순위를 존중하여 처리되도록 지원된다.

본 논문은 실시간 CORBA의 '우선순위 모델'을 설계하고 구현한 연구 결과를 제시한다. 우선순위 모델 구현은 오픈 소스이며, 성능이 우수한 것으로 잘 알려진 (비실시간) ORB인 omniORB2 (v3.0.0)[15]을 기반으로 이를 확장한 형태로 실현하였다. 구현된 실시간 우선순위 모델의 실시간성 개선 결과의 분석을 위해서, 서버 객체 메소드의 반복적인 호출을 수행하고, 1) 평균 지연시간(latency: 메소드 호출 처리 시간) 2) 지터

(jitter: 메소드 호출 처리시간의 표준편차) 등의 실시간 QoS를 측정하여 비실시간 ORB 의 경우와 비교하였다. 실험 분석 결과, 본 논문에서 구현된 우선순위 모델 실시간 ORB가 우선순위를 존중하여, 우선순위가 높은 클라이언트의 요구처리에 비실시간의 경우보다 나은 실시간 QoS (지연시간, 지터 등) 성능을 보여줌을 확인하였다.

본 논문의 연구가 기존 실시간 CORBA 연구[6,10]와 다른 점은 기존 연구는 실시간 CORBA 명세 발표 전에 주로 각 연구 그룹의 독자적인 명세 문맥에 따른 설계 및 구현에 관한 것이었으나, 본 논문의 연구는 실시간 CORBA 명세의 문맥에 따라 설계되고 구현된 것이라는 점이다. 현재 실시간 CORBA 명세는 미국 워싱턴 대학의 고성능 ORB인 TAO[6]에서 일부 구현하고 있으나, 우선순위 모델을 포함한 포괄적 구현은 아직 발표되어 있지 않다. 또한 미국 OIS사 가 실시간 ORB인 ORB Express를 출시하고 있으나,[16] 구조에 자세한 내용이 공개되어 있지 않아 실시간 CORBA 규약에 의한 실시간 ORB 구현인지는 아직 명확하지 않다.

Ⅱ. CORBA의 개요 및 실시간 환경에서의 CORBA

본 절에서는 CORBA의 실시간성 개선 연구에 이해가 필요한 연구 배경을 기술한다.

1. CORBA 구조

CORBA는 OMG에서 규정한 미들웨어 표준이다.[11] CORBA를 이용한 분산 환경에서; 클라이언트가 서버 객체가 제공하는 서비스를 이용하기 위해서는 서비스를 구현한 객체의 해당 메소드를 호출하도록 되어 있다.[2,4,9,11] CORBA에서, 객체는 잘 정의된 인터페이스를 갖는 실체(entity)이며, 객체로의 접근은 인터페이스를 통해 이루어진다. 인터페이스는 IDL(Interface Definition Language)로 기술된다.

CORBA의 구성요소 가운데 가장 중요한 것은 ORB이다. ORB는 클라이언트의 서버 메소드 호출을 중개하여, 해당 메소드를 구현한 객체를 찾아 주며, 해당 메소드 호출을 객체 어댑터와 협력하여 처리한다. ORB는 클라이언트가 목표 객체의 메소드의 호출(즉, 서버가 제공하는 기능이 구현된 함수를 호출하는 것으로 간주할 수 있음)을 목표 구현객체의 위치, 구현 프로그래밍 언어, OS 플랫폼, 통신 프로토콜, 네트워크, 하드웨어 등에 관계없이 가능하게 하므로, 분산 응용 개발자는 ORB 규약을 지키기만 하면, 클라이언트/서버 환경을 보다 쉽게 구현할 수있다. 또한, ORB는 클라이언트와 서버가 필요로 하는 기능들을 ORB Interface를 통해 제공한다. ORB 간의 통신을 위해 사용되는 프로토콜은 GIOP(General Inter ORB Protocol)로 규약 되어 있다. GIOP는 하부의 트랜스포트(transport) 계층의 프로토콜로 신뢰성 있는 프로토콜이기만 하면 되는 것으로 규정하고 있는 데, 이중 인터넷의 TCP/IP를 GIOP의 트랜스포트 및 네트워크 프로토콜로 사용하는 규약을 IIOP(Internet Inter ORB Protocol) 로 규정하고 있다.

객체 어댑터는 CORBA 객체에서 구현 객체로의 매핑을 제공하여, 클라이언트의 CORBA 객체 메소드에 대한 호출 요청을 해당 구현 객체의 메소드로 연결되어 처리되도록 하는 역할을 제공한다. 또한 CORBA 객체의 생성, 등록, 소멸 등의 객체 관리를 담당한다. 객체 어댑터로, 보다 분명하고, 확장된 기능을 갖는 POA(Portable Object Adapter)가 CORBA 2.2 부터 규정되어 있으며, 현재 대부분의 ORB들은 이를 지원한다.

그림 1은 클라이언트, 서버, ORB, 객체 어댑터 의 상호관계를 나타내는 CORBA 구조를 보여준다.

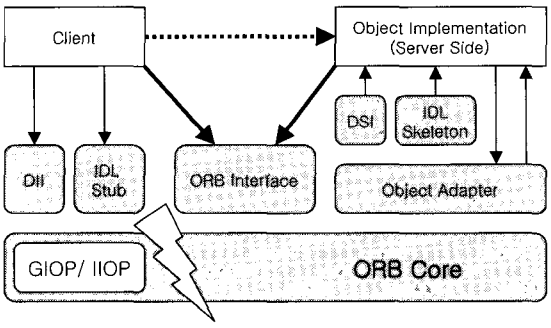

그림 1. CORBA 구조
Fig. 1. Corba Architecture.

또한, CORBA에서는 ORB 상위의 계층으로 CORBA 서비스를 규정하고 있다.[12] CORBA 서비스는 객체 지향 분산 응용 개발을 위해, 각 분산 응용에 필요한 시스템적인 기능을 분산 응용개발자가 따로 개발하지 않고 가져다 쓸 수 있도록, IDL 규정 인터페이스로 포장된 시스템 수준 서비스의 집합으로, 명명 서비스, 이벤트 서비스, 시큐리티 서비스, 동시제어 서비스, 트랙잭션 서비스 등이 제공된다. 각 서비스의 자세한 내용은

[12]를 참조하라.

2. CORBA 상호 운용 구조

CORBA는 분산환경에서 객체를 식별하기 위해 IOR(Interoperable Object Reference)를, ORB 간의 통신을 위해 IOP(Inter-ORB Protocol)를 규약하고 있다.[11]

(1) CORBA 객체 참조 (IOR)

IOR은 분산 환경에서 객체를 식별하기 위한 CORBA의 규약이다. IOR 은 repository ID, profiles 정보 등을 보유하는 데, repository ID는 CORBA 객체 인터페이스 타입을 식별하며, profiles 정보는 서버 위치 정보로 TCP/IP의 경우, TCP 포트 주소, IP 주소 및 서버 주소 공간에서의 객체의 식별 키(object key)를 포함한다.

(2) CORBA 프로토콜 모델

CORBA의 IOP는 ORB 들사이의 상호운용을 지원하는 통신 프로토콜 규약이다. IOP는 데이터 표현 포맷과 ORB 메시지 규격을 정의하는 데, 데이터 표현 포맷으로는 CDR(Common Data Representation)을 사용하고 있으며, ORB 메시지 규격으로 GIOP(General Inter-ORB Protocol)를 정의하고 있다. GIOP 메시지 포맷에는 'Request', 'Reply', 'CancelRequest', 'LocateRequest', 'LocateReply', 'CloseConnection', 'MessageError' 등 7개의 메시지 종류가 있으며, 각각 '메세지 헤더+메세지 몸체' 로 구성된다. 메시지 헤더 포맷은 메시지 종류마다 다르며, 'Request' 메시지의 경우, 서비스 컨텍스트, object key 등의 정보를 포함한다. 서비스 컨텍스트는 'Reply' 메시지의 헤더에서도 포함되는 데, 어떤 ORB 서비스에 요구되는 정보를 전달하는 데 사용된다. 참고로, GIOP 1.1 의 경우, Request 메시지 헤더는 다음과 같다.[2,11]

```
struct RequestHeader_1_1 {
        IOP::ServiceContextList service_context;
        unsigned long request_id;
        boolean response_expected;
        octet reserved[3]; // Added in GIOP 1.1, 따라
                        // 서 GIOP 1.0 에는 없음
        sequence <octet> object_key; //CORBA 객체
                        // 의 object_key
        string operation; //호출되는 CORBA 객체의
                        // 메소드를 나타내는 스트링
        Principal requesting_principal;
            };
```

또한, Request 메시지 헤더와 Reply 메시지 헤더에서 사용되는 서비스 컨텍스트의 구조는 다음과 같다(GIOP 1.1 기준).[2,11]

```
module IOP { // IDL
            ......
        typedef unsigned long ServiceId;
        struct ServiceContext {
            ServiceId context_id;
            sequence <octet> context_data;
                };
        typedef sequence <ServiceContext> o
        ServiceContextList;
                ......
                 };
```

3. CORBA 동작 메카니즘

CORBA에서 지원하는 가장 중요한 기능은 '객체 위치 투명성' 이다. 즉, 서버 객체가 어디에 위치하든 지(같은 주소공간에 위치, 같은 컴퓨터의 다른 주소 공간에 위치, 다른 컴퓨터에 위치 등)에 관계없이, 객체가 제공하는 서비스를 클라이언트가 동일한 문맥으로 이용할 수 있게 하는 것이다. 객체 위치 투명성을 중심으로 설명하는 CORBA의 동작 메커니즘은 다음과 같다. 클라이언트는 서버 객체의 레퍼런스(IOR)를 얻어, 이를 통해 해당 서버 객체의 서비스를 호출하게 된다. 클라이언트 측 ORB는 해당 객체의 레퍼런스 (IOR)로부터 서버 객체가 위치하는 서버 프로세스의 위치 정보(TCP/IP를 이용하는 경우, IP 주소 및 TCP 포트 주소), 서버측 ORB의 해당 객체 어댑터 정보, 객체 어댑터 내에서의 '서버 구현객체 (servant)'의 식별 정보(object key)를 추출하고, 이를 이용하여 해당 서버 객체가 위치하는 서버측 ORB로 클라이언트의 서비스 요청을 GIOP Request 메시지로 전송한다. 서버측 ORB는 객체 어댑터와 협력하여, 전송된 GIOP Request 메시지에 포함된 서버 객체의 식별자를 이용하여 해당 서버 구현 객체를 찾아내고, 그 객체의 서비스 메소드를 호출한다. 이후 수행된 결과는 GIOP Reply 메시지 형태(수행 결과 값은 GIOP Reply 메시지의 몸체에 실려 전달)로 다시 클라이언트 ORB로 보내지고 클라이언트 ORB는 최종적으로 이 결과가 클라이언트 응용에게 전

달되도록 한다.[1] 여기서, 객체 어댑터가 하는 역할을 보다 분명히 기술하면 다음과 같다. 객체 어댑터는 서버 구현 객체가 생성될 때, 이 서버 구현 객체가 서버의 주소공간에서 위치하는 위치 정보와 서버 객체의 식별 정보(object key)와의 매핑 구조를 제공한다. 즉, 서버 객체의 식별정보와 생성된 서버 구현 객체의 위치 정보를 '객체 맵(active object map)'에 기록하며, 이를 관리한다. 따라서, 서버측 ORB는 클라이언트의 서버 객체 서비스 요청이 들어오면, 객체 어댑터와 협력하여 요청 메시지에 포함된 객체 식별자(object key)를 인덱스로 이용하여 이 객체 맵에서 해당 서버 구현 객체의 위치 정보를 얻고 이를 통해 해당 구현 객체의 서비스 메소드를 호출할 수 있게 된다.

4. 실시간 시스템 환경에서의 CORBA

현재의 CORBA 규약은 여러 가지로 실시간성 지원이 미약하다. 가장 미약한 점은 실시간 응용이 요구 QoS를 ORB에 전달하고, 요구 QoS 달성을 위해 ORB가 자원을 적절하게 구성하고 제어할 수 있도록 하는 표준 인터페이스가 없다는 것으로 정리할 수 있다. 이러한 약점들은 최근에 도입된 CORBA 메세징 사양[14] 실시간 CORBA 규약 사양[13]에서 어느 정도 해결되고 있지만, 이러한 사양은 기본적으로 연성(soft) 실시간 시스템 달성을 목적으로 하고 있다. 또한, 클라이언트가 지연시간(latency) 및 지터(jitter)와 같은 타이밍 요구조건을 제시하고자 하는 경우, 이를 표현할 표준 인터페이스 규약이 없다. 게다가, 소켓과 같은 저 수준 프로그래밍 방식에 비교하여, 현재의 ORB 구현은 지연시간 증가, 처리능력(throughput) 감소, 비예측성 증가 등의 오버헤드를 갖고 있다.[7] 이러한 오버헤드들은 어느 정도 개선이 가능하지만, CORBA가 제공하는 장점(객체 위치 투명성, 객체 구현 투명성, 자동적인 마샬링/역마샬링 등)을 지원하기 위해 발생되는 오버헤드는 기본적으로 최소화하는 데에 한계가 있다. 이러한 약점들은 광범위하게 연구되었으며, 이러한 분석 결과에 기반 하여, CORBA의 실시간성을 개선하고자 하는 연구 또한 꾸준히 진행되었다.[6,7,10] 미국 워싱톤 대학의 연구자들이 개발한 TAO는 기존 ORB 구현의 실시간성을 철저히 분석하고, 이에 기반 하여, 실시간성을 개선한 ORB이다.[6] TAO 연구 그룹은 실시간 I/O 시스템, 실시간 GIOP 프로토콜 엔진, 실시간 객체 어댑터, 실시간 스케줄링, 디스패칭, 최적화된 IDL 컴파일러 등을 구현하여, 고성능의 ORB를 구현하였다. 이 방향의 연구는 기존 ORB 내부의 설계 및 구현을 개선하거나, 완전한 재설계를 요구한다. 반면, 미국 로드 아일랜드 대학 연구 그룹은 기존 ORB를 확장하여, 실시간 객체 서비스(실시간 스케줄링 서비스, 글로벌 우선순위 서비스, 실시간 이벤트 서비스, 글로벌 타임 서비스 등) 및 실시간 객체 관리자 등을 추가함으로써 실시간 CORBA 환경 구축을 제공하였다.[10] RT-Event 서비스[1]는 기존 CORBA 이벤트 서비스에 실시간 스케줄링, 실시간 디스패칭, 이벤트 필터링 등의 기능을 추가하여, 기존 이벤트 서비스의 실시간성 한계를 개선하였다. 이러한 방향의 연구는 무엇보다도 기존 ORB를 고치지 않고 이의 기반위에 실시간 환경 지원을 위한 확장 CORBA 서비스를 제공하고 있다는 점에서 기존 ORB를 사용하는 개발자에게는 적합하다.

최근, OMG에서는 이러한 기존의 CORBA 실시간성 개선에 대한 연구 결과에 힘입어 실시간 CORBA 사양을 발표하였다.[13] 실시간 CORBA는 기존 비실시간 ORB의 실시간 확장으로 크게 RTORB, RT POA, RT Scheduling 의 3가지에 대한 사양을 추가적으로 규약하고 있다.

Ⅲ. 실시간 CORBA 사양 및 우선순위 모델

이 절에서는 기존의 CORBA 규약에 실시간성 지원을 추가한 실시간 CORBA 사양[13]을 본 논문에서 구현하고자 하는 우선순위 모델을 중심으로 살펴본다.

1. 실시간 CORBA 구조

실시간 CORBA는 (비실시간) CORBA의 확장으로 규약된다. 통신 프로토콜 역시, 새로운 프로토콜이 규약된 것이 아니고, 기존 ORB 와의 통신이 가능하도록, CORBA 메세징 규약을 이용한 확장이다. 실시간 CORBA의 규약 내용은 크게, RTORB, RT POA, RT 스케줄링 서비스 등의 3가지이다. RTORB는 ORB의 실시간 확장으로 쓰레드 풀, 통신 프로토콜 구성, 통신 채널 연결 구성, 우선순위모델 등을 지원하는 API 규

1) 클라이언트와 서버가 같은 주소공간에 있는 경우, 클라이언트와 서버는 같은 ORB를 사용하게 되며, 이때 효율적인 메시지 처리를 위해 많은 ORB 에서는 서버 객체 메소드 호출이 GIOP 메시징을 경우하지 않고 직접 함수 호출 형태로 이루어지도록 하고 있다.

약이며, RT POA는 우선순위를 갖는 객체 생성 및 등록을 지원하는 실시간 POA에 대한 규약이다. RT 스케줄링 서비스는 CORBA 서비스로 객체 호출의 우선순위 지정, 서버객체의 우선순위 지정, 호출 처리 쓰레드 스케줄링 전략을 지원한다.

2. 실시간 CORBA 의 우선순위 모델

실시간 CORBA 규약 가운데 실시간성 개선을 위해 선택한 가장 중요한 아이디어는, 클라이언트 측에서 서비스를 호출할 때와 서버 측에서 클라이언트의 서비스 요청을 처리할 때, 보다 긴급한 호출 및 요청처리가 먼저 수행될 수 있도록 '우선순위 부여 구조'를 제공한 것이다. 실시간 CORBA 사양에서 지원하고 있는 우선순위의 부여 구조에는 두 가지가 있다. 하나는 '클라이언트 전달 우선순위 모델(Client Propagated Priority Model)' 이고, 또 다른 하나는 '서버 선언 우선순위 모델(Server Declared Priority Model)' 이다. 우선순위 모델에서의 우선순위는 CORBA를 사용하는 분산 응용 전체에 걸쳐 유일한 CORBA 우선순위(CORBA priority)를 말하며, 쓰레드의 수행 시에는 운영체제에서 지원하는 지역 우선순위(native priority)로의 매핑이 필요하다. 또한 다른 CORBA 객체로 우선순위를 전달하는 경우에는 역 매핑이 이루어져야 한다. 실시간 CORBA는 우선순위 매핑(priority mapping)을 규약하고 있다.

(1) 클라이언트 전달 우선순위 모델

클라이언트 측에서 서버 객체 메소드를 호출할 때 우선 순위를 지정하며, 이 우선순위는 서버 측에도 전달되어 서버 측에서 서비스 요청을 처리할 때에 전달된 우선순위를 존중하여 처리되도록 하는 방식이다. 호출 쓰레드의 우선순위 정보는 서버 객체 메소드 호출 메시지(GIOP Request 메시지) 헤더의 서비스 컨텍스트 필드(2.2절의 GIOP Request 메시지 헤더 포맷 참조)에 실어 보내지며, 서버측 ORB는 클라이언트 요청 메시지 헤더의 서비스 컨텍스트에서 우선정보를 추출하여, 이 우선순위에 따라 호출 처리 쓰레드의 우선순위를 변경한다.

(2) 서버 선언 우선순위 모델

서버 객체를 서버측 (RT) POA 에 등록할 때 우선순위를 지정할 수 있도록 하는 방식이다. 등록된 객체의 우선순위 정보는 IOR 를 통해 클라이언트 측에 전달되며, 클라이언트 측은 이 우선순위 정보를 이용할 수있다. 이후 클라이언트 측에서 이 서버 객체의 메소드를 호출할 때, 서버 측 ORB에서 이 객체의 메소드 요청을 처리할 때에, 해당 객체의 등록된 우선순위를 존중되어 처리되도록 지원한다.

(3) 우선순위 모델 정책 (Priority Model Policy)

실시간 CORBA가 지원하는 우선순위 모델은 'PriorityModelPolicy' 라는 CORBA 객체를 통해 선택되고, 구성된다. 이 'PriorityModelPolicy' 객체는 우선순위 모델 정보 (클라이언트 전달 우선순위 모델 또는 서버선언 우선순위 모델을 지정) 와 우선순위 정보를 보유한다. 다음은 실시간 CORBA 의 PriorityModelPolicy 에 대한 IDL 명세이다.

```
// IDL
module RTCORBA {
        .....
   // Priority Model Policy
   const CORBA::PolicyType
      PRIORITY_MODEL_POLICY_TYPE = ??;
      enum PriorityModel {
      CLIENT_PROPAGATED,
      SERVER_DECLARED
         };

interface PriorityModelPolicy : CORBA::Policy {
   readonly attribute PriorityModel priority_model;
   readonly attribute Priority server_priority;
     };
        .....
};
```

서버 선언 우선순위 모델이 선택된 경우, server_priority 는 RT POA 에 의해 관리되는 CORBA 객체의 기본 우선순위이다. 이는 RT POA 가 제공하는 우선순위를 갖는 객체 등록 시에 지정하는 API 함수의 매개변수로 주어지는 우선순위에 의해 변경될 수 있다. 클라이언트 전달 모델이 선정되는 경우, server_priority 는 비실시간 ORB의 클라이언트가 호출하는 경우의 객체의 기본 우선순위로 사용된다.

참고로, "PRIORITY_MODEL_POLICY_TYPE = ??" 에서 ?? 의미는 OMG에서 정의하지 않았기 때문에, OMG가 정의한 PolicyType과 겹치지 않는 상수로 사

용자가 정의해서 사용한다는 의미이다. 본 논문의 구현에서는 0xFFFF로 정하였다.

(4) 우선순위 모델 정책의 범위

우선순위 모델 정책은 RT POA가 생성될 때, 서버 측에서 적용되며, 이 우선순위 모델 정책 객체의 인스턴스는 RTORB의 'create_priority_model_policy' API 호출로 생성된다.

PriorityModelPolicy는 클라이언트에게 알려지는 정책이며, 이는 IOR을 통해 서버로부터 클라이언트로 전달된다.

Ⅳ. 실시간 CORBA의 우선순위 모델 설계 및 구현

실시간 CORBA의 우선순위 모델을 구현하는 데는 2가지의 접근을 고려할 수 있다.

- 기존 ORB의 확장으로 우선순위 모델 구현
- 처음부터 실시간 CORBA를 위한 실시간 ORB 설계 및 구현

두 번째 방법은 실시간 CORBA 구현에 맞는 최적의 설계가 가능하다는 점에서 이상적이나, ORB를 설계하고 구현하는 것은 방대한 작업이므로, 많은 시간과 노력이 소요된다. 첫 번째 방법은 최적의 설계가 아닐 지 모르지만 기존의 ORB의 기능을 이용할 수 있다는 점에서 짧은 시간에 적은 노력으로 완수가 가능하다. 따라서 본 논문에서는 기존의 ORB를 이용하여 우선순위 모델을 구현하였다. 기존 ORB로는 소스코드가 공개되어 있고, 성능이 우수한 것으로 잘 알려져 있는 omniORB2(v3.0.0)[15]를 사용하였다. 그런데, 기존 ORB를 이용하는 경우, 우선순위 모델의 설계 구조가 기존 ORB에 의존할 수밖에 없기 때문에, 기존 ORB 구현(즉 omniORB2)에 독립적인 설계가 불가능하다. 따라서, 본 논문에서 기술하는 우선순위 모델 설계 구조가 omniORB2 의 설계 구조에 기반 되어 있다. 또한, 본 논문에서는 우선순위 모델만을 구현하였으며, 우선순위 모델과 관계있는 CORBA 우선순위와 지역 우선순위와의 매핑, 우선순위 결정방법, 스케줄링 정책 등은 추후 계속 연구되어야 함을 밝힌다.

1. 우선순위 모델 구현을 위한 실시간 CORBA 요소

앞 절에서 살펴보았듯이, 실시간 CORBA가 지원하는 우선순위 모델을 구현하는 데에 있어서, GIOP 메시지 처리, IOR 처리, 객체 레퍼런스 생성, 이들과 연관된 구조 등의 이해가 필수적이다. 기존의 비실시간 ORB는 실시간 CORBA의 우선순위 모델 구현에 필요한 구조를 다 지원하지 못한다. 예를 들어, 클라이언트 전달 우선순위 모델에서 지원하여야 할 GIOP Request(Reply) 메시지의 우선순위 서비스 컨텍스트는 기존 ORB의 GIOP 엔진에서 지원하지 않는다. 더군다나, 기존 ORB는 실시간 IOR를 해석하는 데, 필요한 구조를 지원하지 않는다. 이 절에서는 우선순위 모델 구현을 위하여 필요한 실시간 CORBA 요소에 대해 살펴 본 결과를 기술한다.

(1) 실시간 IOR (RT_IOR)

실시간 CORBA가 지원하는 우선순위 모델은 PriorityModelPolicy라는 CORBA 객체를 통해 선택되고, 구성되도록 규약 되어 있다. PriorityModelPolicy 객체는 우선순위 모델 정보와 우선순위 정보를 보유한다. 우선순위 모델 정책은 RT POA 가 생성될 때, 서버에 적용된다. PriorityModelPolicy는 클라이언트에게 알려지는 정책으로, IOR를 통해 서버로부터 클라이언트로 전달된다.[13] 기존 비실시간 ORB가 사용하는 IOR은 이러한 PriorityModelPolicy 정보를 지원하지 않기 때문에 실시간 ORB 구현 시에는 기존 비실시간 IOR 에 PriorityModelPolicy 정보가 추가된 실시간 IOR (이하 RT_IOR)를 지원하여야 한다. (비실시간 CORBA) IOR 구조에서 우선순위 모델 정보를 추가할 수 있는 부분은 IOR 구조의 마지막의 필드인 IOP::TaggedComponent 부분이다. 실시간 CORBA가 기반하고 있는 CORBA 메세징 사양[14]을 참조하여, IOP::TaggedComponent 부분을 결정한 실시간 IOR 구조는 아래와 같다. RT_IOR에서, 굵은 글씨체 부분은 실시간 CORBA 지원을 위해 추가된 부분이다. PRIORITY_MODEL_POLICY_TYPE 는 현재, 그 값이 OMG에서 규정되어 있지 않으며, 구현자에 따라 정하면 된다. 본 논문에서는 0xFFFF 로 정하였다.

```
struct RT_IOR {
        string type_id; //IDL 인터페이스 respository
          //포맷
        ProfileId tag; //const로 TAG_INTERNET_IOP
          //이며 값은 0
        octet     major; //IIOP major version number
          //로 현재는 1
```

```
octet     minor; //IIOP minor version number
  //로 1.x 의 경우의, x값
string    host; //IP address
unsigned short port; //TCP port number
sequence <octet> object_key; //object의 key
componentId 2 ; //componentId 2는 policy를
  // 나타내며, unsigned long이므로 4 바이트
PRIORITY_MODEL_POLICY_TYPE 0xFFFF;
        // unsigned long 으로 4 바이트임에
        // 주의한다.
PriorityModel   0 or 1 ; //client propagated
model 은 0, server declared model 은 1
Priority  priority ; // 2 바이트
        };
```

(2) 우선순위 정보 전달을 위한 서비스 컨텍스트

타겟 서버 구현객체(servant)가 클라이언트 전달 우선순위 모델을 지원하는 경우, 클라이언트는 서버 객체 메소드 호출의 우선순위를 지정할 수 있으며 (RTCORBA::Current 이용), 이렇게 지정된 클라이언트의 우선순위는 서버 객체 메소드 호출 시에 GIOP Request 메시지 헤더의 '서비스 컨텍스트' 필드에 실려 전달된다. 서버측 GIOP 엔진은 이 GIOP Request 메시지를 받은 후, 서비스 컨텍스트를 해석하여 우선순위를 추출하고, 이후의 처리를 담당하는 쓰레드의 우선순위가 이 우선순위를 반영하여 바뀌게 한다. 만약 타겟 객체가 서버 선언 우선순위 모델로 등록된 경우, 서버 객체 메소드 호출시의 GIOP 메시지의 서비스 컨텍스트에 우선순위 정보는 실리지 않는다. 여기서, 서비스 컨텍스트로 전달되는 우선순위는 CORBA 우선순위이다. 따라서, 서버 측에서 서버측 운영체제의 지역(native) 우선순위로 바뀌는 우선순위 매핑이 필요함에 유의해야 한다. 2.2절에 소개된 서비스 컨텍스트 구조에서, 우선순위 정보 전달을 위한 서비스 컨텍스트 ID는 CORBA에서는 추후 그 값을 결정하도록 하고 있다. 본 논문에서는 그 값을 임의로 0x000000FF (255)로 정하였으며, 우선순위 정보는 1 octet (0-255)으로 전달되도록 하였다.

(3) RT POA

RT POA는 기존 비실시간 POA(PortableServer::POA)의 계승으로 규정되며, 우선순위 모델 및 객체별 우선순위를 지정하여 RT POA에 등록하는 함수들을 규정한다. POA는 CORBA 객체의 object key와 이 CORBA 객체를 구현한 서버 객체와의 매핑을 제공하는 데, 이 매핑구조를 위해 CORBA는 '객체 맵'을 규정하고 있다. RT POA 에는 이 객체 맵에 우선순위 모델 정보와 우선순위 정보가 추가되어야 하며, 우선순위를 지원하는 객체를 지원하는 함수가 구현되어야 한다.

(4) 실시간 객체 레퍼런스 타입 객체

CORBA에서 CORBA 객체에 대한 정보(repository ID 및 profiles)는 IOR를 통해 클라이언트 측에 전달된다. 클라이언트는 해당 객체의 IOR를 명명 서비스를 통하거나, 직접 서버로부터 입수하는 데, 이 경우 IOR은 GIOP Reply 메시지의 몸체에 실려 전달된다. 클라이언트 측 ORB의 GIOP 엔진은 GIOP Reply 메시지 처리 시에, IOR를 꺼내고 이 IOR 로부터, repository ID 와 profiles 정보를 추출하여, 이 정보들을 보유하는 객체 레퍼런스 타입의 스터브 객체(프록시)를 생성한다. 앞에서 언급한 바처럼, RT_IOR은 비실시간 IOR에 비해 우선순위 모델 정보 및 우선순위 정보를 포함한다. 클라이언트 측 GIOP 엔진에서는 해당 객체의 RT_IOR을 입수하는 즉시, 이 RT_IOR로부터 우선순위 모델 정보 와 우선순위 정보를 추출하여, 추후의 사용을 위해, PriorityModelPolicy 객체를 생성하고, 이 정보들을 여기에 보관한다. 따라서, 실시간 ORB 는 RT_IOR 생성, RT_IOR로부터 실시간 객체 레퍼런스 타입 객체의 생성 등을 지원하여야 한다.

(5) 실시간 ORB의 쓰레드 우선순위 변경 메커니즘 구조

클라이언트가 해당 객체의 메소드를 호출할 때, 클라이언트 ORB는 해당 메소드 호출을 GIOP Request 메시지로 변경하기 전에, 해당 객체와 연관된 PriorityModelPolicy 객체를 참조한다. 해당 객체가 클라이언트 모델인 경우, RTCORBA::Current 에 보관된 우선순위를 추출하여, 현재 (서비스 요청) 호출 쓰레드의 우선순위를 이 우선순위를 반영하여 변경하고, 이 우선순위는 Request 메시지의 서비스 컨텍스트에 실어 서버측으로 보낸다. 해당 객체가 서버 선언 모델인 경우, 호출 쓰레드의 우선순위를 PriorityModelPolicy 객체에 보관된 우선순위를 반영하도록 변경할 수있다. 서버 선언의 경우는 우선순위를 싣는 서비스 컨텍스트를 만들지는 않는다.

서버측 ORB에서 해당 작업 쓰레드의 우선순위 변경은 다음과 같이 이루어 진다. 먼저, 클라이언트 전달 모델의 경우, 서버측 ORB가 GIOP Request 메시지를 수

신한 후, GIOP Request 메시지의 서비스 컨텍스트 부분의 우선순위 정보를 점검하여, 이 때 해당 작업 쓰레드의 우선순위를 이를 반영하여 변경한다. 이후의 클라이언트 요구처리는 변경된 쓰레드의 우선순위로 처리된다. 서버 선언 모델의 경우, 해당 작업 쓰레드의 우선순위 변경은 이보다 후에 이루어진다. 즉, 서버측 ORB는 GIOP Request 메시지에서 해당 서버 객체의 object key 값을 추출하여, 이를 이용하여 POA와 협동하여 객체 맵에서 해당 객체를 찾게 되는 데, 이때, 등록된 해당 객체의 우선순위 모델 및 우선순위 정보를 점검하여, 서버 선언 모델로 등록되어 있으면, 등록된 우선순위를 반영하여 작업 쓰레드의 우선순위를 변경한다. 이후의 클라이언트 요구 처리는 변경된 쓰레드의 우선순위로 처리된다.

2. 우선순위 모델 구현 내용

이 절에서는 omniORB2에 기반하여 구현된, 본 논문의 실시간 CORBA 의 우선순위 모델 구현 내용을 기술한다.

(1) 서버측 ; ORB 와 RT POA

omniORB2에서 구현하고 있는 POA의 객체 맵은 omniLocalIdentity 라는 객체의 포인터들의 해쉬 테이블로 구현하고 있다. omniLocalIdentity 객체는 서버 구현객체(servant)의 포인터, 해당 POA 포인터 등의 정보를 멤버변수로 갖는다. 객체의 object key는 이 해쉬 테이블의 인덱스 키가 된다. 즉, 서버측 ORB는 GIOP Request 메시지 헤더에서 object key를 추출하고 이를 이용하여 객체맵 내에서 해당 omniLocalIdentity 객체를 찾아내고, 이 객체에서 구현 서버 객체 포인터를 찾아, 이를 이용하여 서버 구현객체의 메소드를 호출하도록 되어 있다. 이러한 omniORB2 의 POA 구조를 확장하여, 본 논문의 RT POA 는 PriorityModelPolicy 객체

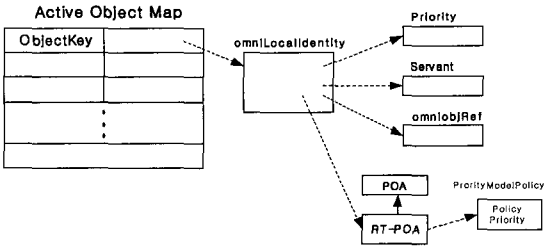

그림 2. 서버측 RTORB 및 RT POA 구현구조
Fig. 2. Implementation Arch tecture of Server Side RTORB and RT POA.

와 추가적으로 연관되도록 하였으며, omniLocalIdentity 객체에 우선순위 정보를 추가하였다.

(2) 클라이언트측 ORB 구조

클라이언트측 ORB는 서버 객체 IOR이 주소공간에 들어오는 즉시, 서버객체 프록시(스터브)를 만든다. 이 스터브에는 서버객체의 repository ID와 profiles 정보(서버의 TCP/IP 주소, 서버 객체의 object key 정보를 포함)가 포함되어 있다. 비실시간 ORB에서는 클라이언트 호출시에, 스터브의 이 정보를 참조하여, GIOP Request 메시지를 만들고, 이를 추출된 해당 서버 객체의 TCP/IP 정보를 이용하여 해당 서버로 보내게 된다. 실시간 ORB의 경우, 4.1절에서 설명한 바대로, 클라이언트 전달 우선순위 모델의 경우, 해당 호출 쓰레드의 우선순위를 변경하고 또 이 우선순위 정보를 GIOP Request 메시지에 실어야 하기 때문에 이 우선순위 정보를 RTCORBA::Current에서 얻어 와야 한다. 따라서, 스터브에 PriorityModelPolicy 포인터와 RTCORBA::Current 포인터 정보를 추가하였다. 다음 그림 3은 이 구조를 보여준다.

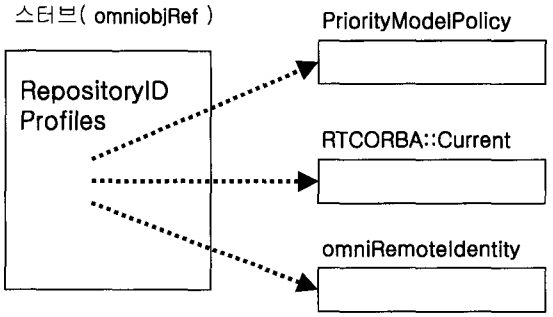

그림 3. 클라이언트 측 RTORB 구현 구조
Fig. 3. Implementation Arch tecture of Client Side RTORB.

(3) RT_IOR 및 실시간 객체레퍼런스

omniORB2에서는 실시간 IOR를 지원하지 않는다. 본 논문에서는 4.1 절에 설명한 RT_IOR를 설계하고 구현하였으며, 비실시간 IOR 도 구현된 ORB에서 동작되도록 하였다. 또한, 객체 레퍼런스가 갖는 객체의 profiles를, RT_IOR 에서 확장된 우선순위 모델 및 우선순위 정보를 추가적으로 보유할 수 있도록 변경하였다. 또한, 기존 비실시간의 IOR 생성, 객체 레퍼런스 타입의 객체 생성을 지원하는 모듈을 확장하여, RT_IOR 생성, 실시간 객체 레퍼런스 타입의 객체(스터브 등) 생성 등

을 추가적으로 지원할 수있도록 변경하였다.

(4) 우선순위 지원 GIOP 엔진

우선순위 모델을 지원하기 위해서, GIOP 는 서비스 컨텍스트를 지원하여야 한다. 현재 omniORB2는 GIOP 1.0 만을 지원하는 데, 이 GIOP 1.0 은 서비스 컨텍스트를 지원하지 않는다. 따라서, 서비스 컨텍스트를 지원하는 GIOP 1.1 규격에 맞추어, 우선순위 모델 서비스 컨텍스트를 지원하도록 GIOP 1.0 메시지 포맷을 확장하였으며, 이에 관련된 GIOP 엔진 부분들 (클라이언트 측에서의 GIOP Request 메시지 생성 부분, 서버 측에서의 GIOP Request 메시지 해석 부분, GIOP Reply 메시지 생성 부분, 쓰레드 우선순위 변경 부분 등)을 확장하여, 비실시간 GIOP 메시지뿐만 아니라, 우선순위 모델 지원 실시간 GIOP 메시지 처리가 가능하도록 하였다.

V. 실험 및 결과 검토

이 절에서는, 본 논문에서 설계하고 구현한 실시간 CORBA의 우선순위 모델이 기존의 비실시간 CORBA에 비해, 확실하게 우선순위를 존중하여 클라이언트 요청을 처리하는 것을 보인 실험 결과를 기술한다.

1. 실험방법

본 논문에서는 구현된 우선순위 모델 지원 실시간 ORB가 비실시간 ORB에 비해, 클라이언트 요청 처리를 우선순위를 존중하여 수행하는 것을 보이기 위해, 평균 RTT(Round Trip Time) 지연시간과 지터를 실시간 ORB와 비실시간 ORB에 대해 각각 측정하였다. 여기서, RTT 지연시간은 클라이언트가 서버 객체의 메소드의 호출을 시작하여 호출을 완료하기까지의 걸린 시간을 의미하며, 이 때 평균 RTT 지연시간은 동일한 메소드를 m 번 반복 호출하여 각 호출의 RTT 지연시간을 평균한 값을 의미한다.

지터는 m 번 반복 측정했을 때, 지연시간들의 표준편차를 의미한다. 평균 RTT 지연시간의 측정은 ORB의 성능을 보여준다. 따라서, ORB A 가 ORB B 보다 우선순위가 높은 호출에 대해 평균 RTT 지연시간이 낮으면, ORB A 가 ORB B 보다 메소드 호출 처리에 우선순위를 존중한다는 것을 나타낸다. 지터가 낮다는 것은 매 메소드 호출처리 시간의 변동이 적다는 것이므로, 메소드 호출 처리의 예측성을 보여준다. 따라서, 낮은 지터를 보이는 ORB에서는 우선순위를 존중하는 메소드 처리가 안정적으로 수행됨을 나타낸다.

본 논문의 실험에서 평균 RTT 및 지터 비교를 위해 사용한 서버 객체 메소드로는 클라이언트가 보낸 8KBytes 중 첫 1 바이트를 3제곱하여 다시 클라이언트 측에 돌려주는 서비스(이하 cubic 서비스)를 제공하는 것을 사용하였다. 서버 객체들은 같은 메소드를 제공하나, 각기 다른 객체 레퍼런스를 가진 별도의 객체들로 서버 POA 에 등록된다. 비실시간 ORB 의 경우, 우선순위의 지정없이 등록되며, 서버 선언 우선순위 모델의 경우는 각 객체별로 우선순위가 등록된다. 본 논문의 실험에서는 클라이언트 측의 n 개의 쓰레드가 각각 해당 서버 객체의 메소드를 반복하여 호출하도록 하였다. 클라이언트 전달 우선순위 모델의 경우의 실험에서는 클라이언트에서 서버 객체 메소드 호출시에, 해당 클라이언트 쓰레드의 우선순위가 반영되어 호출되도록 하며, 이 우선순위가 또한 GIOP Request 메시지에 전달되도록 하였다. 서버 선언 우선순위 모델의 경우, 클라이언트 쓰레드는 해당 클라이언트 쓰레드의 우선순위로 등록된 객체의 메소드를 호출한다. 호출의 주기는 높은 우선순위의 경우는 50msec, 낮은 우선순위의 경우는 100msec 이며, 호출의 횟수로 높은 우선순위 호출의 경우에 8000번, 낮은 우선순위의 경우에 4000번이 수행되도록 하였다. 클라이언트 측의 n 개의 쓰레드중 (n-1) 개의 쓰레드의 우선순위는 모두 동일하며, 1 개의 쓰레드는 다른 쓰레드에 비해 우선순위가 높다. 호출 쓰레드의 우선순위를 1개의 높은 것과 나머지 동일한 낮은 것의 2가지로만 결정하여 실험한 것은 높은 우선순위 호출의 처리가 더 낮은 우선순위 호출의 처리에 비해 확실히 우선순위를 존중받아 처리되는 것을 보기 위해, 여러 종류의 낮은 우선순위 호출들을 사용함으로써 고려되어야 하는 실험 변수를 배제하기를 원하기 때문이다. 서버선언 우선순위 모델 실험의 경우, 서버 객체는 호출하는 쓰레드의 우선순위를 반영한 우선순위로 등록된다. 클라이언트 전달 우선순위 모델의 경우, 호출되는 서버 객체의 메소드 처리가 클라이언트의 해당 호출 쓰레드의 우선순위를 반영하여 처리된다.

이 절의 실험 결과는 높은 우선순위 쓰레드에 대해 메소드 호출의 평균 RTT 지연시간과 지터를, 비실시간 CORBA, 실시간 CORBA의 클라이언트 전개 우선순위 모델 및 서버 서언 우선순위 모델 의 3가지 경우에 대해 측정하여 비교한 것이다. 참고로, 본 논문에서 선택한 우선순위는 우선순위 매핑 구조가 구현되어 있지

않았기 때문에('우선순위 매핑' 은 본 논문의 주제인 우선순위 모델과 깊은 관련은 있으나, 별도의 구현 주제임), 윈도우즈 NT에서 지원하는 우선순위만을 이용하였다. 즉, 낮은 우선순위 와 높은 우선순위는 윈도우즈 NT에서 지원하는 'THREAD_PRIORITY_NORMAL' 에, 높은 우선순위는 'THREAD_PRIORITY_ABOVE_NORMAL' 에 각각 매핑되도록 선택하였다.

2. 실험환경

본 논문에서 사용한 실험환경은 Pentium II 416MHz, 128Mbyte(Memory)의 하드웨어 사양에 윈도우즈 NT 4.0 workstation을 탑재한 PC이다. 따라서, 우선순위에 따른 스케줄링은 윈도우즈 NT 운영체제의 스케줄링 전략에 의존하게 되는 데, 윈도우즈 NT는 우선순위를 존중한 스케줄링을 지원한다. 또한, LAN 사용 시에 발생할 수 있는 다른 트래픽의 영향을 배제하기 위해, 클라이언트 및 서버 모두 같은 컴퓨터 상에서 동작시켰으며, 실험 시에 윈도우즈 NT의 다른 시스템 프로세스들의 영향을 배제하기 위해, 클라이언트 및 서버 프로세스는 'REAL_TIME_PRIORITY_CLASS' 라는 가장 높은 프로세스 우선순위 클래스를 사용하였다. 채택한 클라이언트 측의 쓰레드 개수 n 은 2, 6, 11, 21, 31, 41, 51 이다 (따라서, 낮은 우선순위 쓰레드 개수는 1, 5, 10, 20, 30, 40, 50 이다).

3. 평균 RTT 지연시간 비교

평균 RTT 지연시간은 서버 객체 메소드 호출 수행 완료에 걸리는 시간을 측정하는 것으로 이 값이 작을수록 메소드 호출 수행에 걸리는 시간이 적게 걸리는 것을 의미하므로, ORB의 성능을 보여주는 좋은 측도이다. 특히 우선순위가 높은 오퍼레이션의 마감시간 준수가 더 요구되는 실시간 분산 시스템의 환경에서, 높은 우선순위 쓰레드의 메소드 호출 경우에, RTT 지연시간을 개선하는 것은 매우 필요하다. 본 절의 실험 결과는 본 논문에서 설계하고 구현한 클라이언트 전달 우선순위 모델과 서버 선언 우선순위 모델 ORB가 비실시간 ORB(omniORB2)에 비해, 높은 우선순위 메소드 호출의 평균 RTT 지연시간에 대해서 더 나은 성능을 가지고 있음을 보여준다.

그림 4 는 비실시간 ORB와 클라이언트 전달 우선순위 모델의 실시간 ORB, 서버 선언 우선순위 모델의 실시간 ORB 각각에 대해, 높은 우선순위 쓰레드의 메소드 호출 평균 RTT 지연시간들을 비교한 것이다. 그림 4 에서 x 축은 낮은 우선순위 쓰레드 개수를 나타내며, y 축은 높은 우선순위 쓰레드의 메소드 호출 평균 RTT 지연시간이다. 부하가 작을 경우에는 비교적 차이가 적으나, 부하가 커지는 경우, 클라이언트 우선순위 모델의 실시간 ORB의 경우와 서버 선언 우선순위 모델의 실시간 ORB 경우가 비실시간 ORB에 비해 평균 RTT 지연시간의 개선의 차이가 큼을 볼 수있다. 4.1절 5)에서 설명한 바와 같이, 클라이언트 전달 우선순위 모델의 경우, 클라이언트의 메소드 호출 처리를 담당하는 서버 ORB 측의 해당 쓰레드의 우선순위는 클라이언트 GIOP Request 메시지가 서버측 ORB에 도달하자마자, ORB 가 이 메시지의 서버스 컨텍스트 필드에 실린 우선순위를 반영하여 변경된다. 이에 반하여, 서버선언 우선순위 모델에서는 이후, POA가 객체 맵에서 등록된 객체를 찾고, 이때 등록된 객체의 우선순위를 반영하여 해당 쓰레드의 우선순위가 바뀌게 되므로, 우선순위 반영이 늦다. 따라서, 클라이언트 전달 우선순위 모델의 경우가 대체적으로 평균 RTT 지연시간이 적을 것으로 예측할 수있으며, 그림 4는 이를 확인하여 준다.

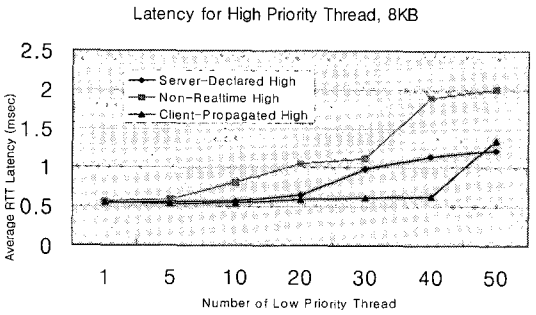

그림 4. 비실시간 ORB 와 우선순위 모델 지원 실시간 ORB 와의 평균 RTT 지연시간 비교

Fig. 4. Comparison of Averige RTT Latency between non real-time ORB.

그림 4에서 낮은 우선순위 쓰레드의 수가 증가함에 따라, 어느 시점에서 평균 RTT 지연시간의 급격한 변화를 볼 수있다. 이는 낮은 우선순위 쓰레드의 수가 증가함에 따라, 시스템이 처리하여야 할 처리 부하가 시스템의 처리능력에 비해 과중하게 되기 때문인 것이 중요한 요인으로 보인다. 더구나, 클라이언트와 서버가 모두 한 컴퓨터에서 처리됨으로, 클라이언트 호출 쓰레드 수의 증가는 클라이언트 측의 처리 부하뿐만 아니라, 이를 처리하는 서버측의 처리부하의 증가도 초래한다. 또한, 이러한 급격한 변화는 omniORB2의 동작구조

에도 일부분의 원인이 있는 것으로 추정되나, 이에 대한 보다 자세한 분석이 필요하다.

4. 지터 비교

지터는 각 메소드 호출의 RTT 지연시간의 변동을 측정하는 측도로써, 낮은 지터는 각 메소드 호출의 RTT 지연시간이 비교적 균일하다는 것을 의미하므로, 예측성이 요구되는 실시간 시스템 환경에서 필요한 QoS이다. 특히 우선순위가 높은 쓰레드의 서버 객체 메소드 호출에 있어서, 높은 예측성의 보장이 실시간 시스템 구축에 바람직하다. 본 절의 실험 결과는 본 논문에서 설계하고 구현한 클라이언트 전달 우선순위 모델과 서버 선언 우선순위 모델의 경우가 비실시간 ORB의 경우에 비해, 높은 우선순위 메소드 호출의 경우에 대한 지터가 개선되었음을 보여준다.

그림 5는 비실시간 ORB와 본 논문에서 구현한 클라이언트 전달 우선순위 모델, 서버선언 우선순위 모델에 있어서, 높은 우선순위 쓰레드의 서버 객체 메소드 호출의 RTT 지연시간을 비교한 것이다. 그림 5 에서 보면, 클라이언트 전달 우선순위모델과 서버선언 우선순위 모델의 경우가 지터가 낮아, 보다 높은 예측성을 지원함을 알 수있다. 이는 비실시간 ORB에서는 클라이언트에서 서버의 메소드 호출이 우선순위가 높게 호출되었어도, 서버측에서는 우선순위가 존중되지 않기 때문에, 서버에서의 각각의 메소드 호출 처리 쓰레드 사이에서 자원 경합시에 어느 호출 처리가 먼저 수행될지에 대한 제어가 없어 처리시간에 대한 변동이 커지게 되는 반면, 클라이언트 전달 우선순위 모델이나 서버 선언 우선순위 모델의 경우는 우선순위가 존중되므로, 우선순위가 높은 요청의 처리가 자원경합에서 우선권을 가지고 수행되므로, 우선순위가 높은 요청처리는 처리시간에 대한 변동, 즉 지터가 적게 되기 때문이다.

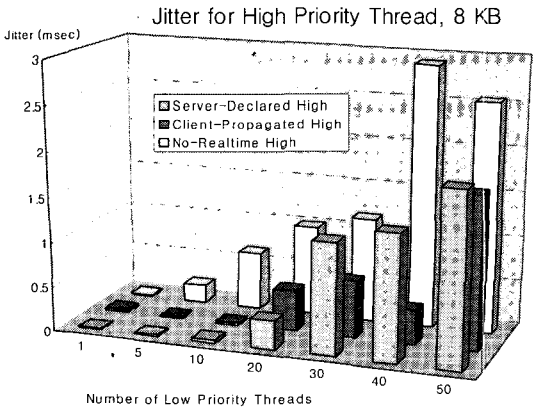

그림 5. 비실시간 ORB와 우선순위 모델 지원 실시간 ORB와의 지터비교

Fig. 5. Comparison of Jitter between non real-time ORB and real-time ORB implementing priority Model.

VI. 결론 및 향후 과제

본 논문에서는 실시간 CORBA 사양중 '우선순위 모델' 에 대해 기존 비실시간 오픈 소스 omniORB2 에 기반하여 설계하고 구현한 연구 결과를 기술하였다. 수행한 실험결과는 본 논문에서 구현된 실시간 CORBA 우선순위 모델 ORB가 더 나은 실시간 특성 (평균 RTT 지연시간 및 지터)을 나타냄을 확인하여 주었다. 본 논문의 실험에서 사용한 서버 객체 메소드(cubic 서비스)는 서버 구현 객체의 해당 메소드 내에서 작업하는 양이 적다. 만약 서버 구현 객체 내에서 작업해야 할 내용이 많은 서비스를 호출하는 실험의 경우, 우선순위 모델의 실시간성 개선 효과는 더욱 두드러질 것이다. 우선순위 모델을 완전히 지원하기 위해서는 우선순위 매핑이 지원되어야 하는 데, 현재 이에 대해 작업을 진행하고 있다. 본 논문의 실시간 CORBA 의 우선순위 모델 구현은 '실시간 ORB 설계 및 구현' 의 일환이다. 현재 그 밖의 실시간 CORBA의 사양의 구현에 대해서, 작업을 진행하고 있다. 향후 이에 대한 보고가 이루어 질 것이다. 본 논문에서 설계하고 구현한 우선순위 모델 지원 실시간 ORB 의 설계 및 구현 구조는 보다 광범위한 성능 평가를 통한 꾸준한 개선 작업이 필요하다.

참 고 문 헌

[1] T. Harrison, L. Levine and D. C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," *Proc. of OOPSLA '97*, ACM, Atlanta, GA, Oct., 1997.

[2] M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*, Addison-Wesley, 1998.

[3] K. Kusunoki, I. Imai, H. Ohtani, T. Nakakawaji, K. Ushijima, M. Oshima, "A CORBA-Based Remote Monitoring System for Factory

Automation,", 2nd Int'l conf. on Object-Oriented Distributed Real-time Systems, Tokyo, 1988.

[4] R. Orfali, D. Harkey, and J. Edwards, *The Essential Client/Server Survival Guide*, John Wiley and Sons, New York, 1995.

[5] A. Polze, D. Plakosh, and K. C. Wallnau, "Study in the Use of CORBA in Real-Time Settings: Model Problems for the Manufacturing Domain," Technical Report, CMU/SEI-97-TR-011, Software Engineering Institute, Carnegie Mellon University.

[6] D. C. Schmidt, D. Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers," Computer Communications, 21(4), pp. 294~324, April, 1998.

[7] D. C. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale, "Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers," to appear, *The International Journal on Time-Critical Computing Systems*, 2000, Kluwer Academic Publishers.

[8] J. Stankovic, K. Ramamritham (Eds.), *Advances in Real-Time Systems*, IEEE Computer Society Press, Washington (DC), 1993.

[9] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Hetergeneous IEEE Communications Magazine, vol. 14, February, 1997.

[10] V. Wolfe, L. Dipippo, R. Ginis, M. Squadrito, S. Wohlever, I. Zykh and R.Johnston, "Expressing and Enforcing Timing Constraints on a Dynamic Real-Time CORBA System," *The Iternational Journal on Time-Critical Computing Systems*, 1999, Kluwer Academic Publishers.

[11] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OMG Document formal/98-12-01.

[12] Object Management Group, *CORBAservices: Common Object Services Specification, OMG Document formal/98-07-05.*

[13] Object Management Group, *Real-Time CORBA Joint Revised Submission, OMG Document* ptc/99-05-03.

[14] Object Management Group, *CORBA Messaging Specification, OMG TC Document, OMG Document orbos/98-05-05*

[15] omniORB2, http://www.uk.research.att.com/omniORB2

[16] ORB Express, http://www.ois.com

저 자 소 개

朴 順 禮(正會員)

1998년 2월 한국 방송 통신 대학교 졸업(학사). 2000년 8월 숭실대학교 대학원 졸업(석사). 2000년 9월~현재 영보시스템 근무. 연구 관심분야 : 분산 시스템, 실시간 시스템

鄭 善 太(正會員)

1983년 2월 서울대학교 전자공학과 졸업(학사). 1985년 12월 미국 미시간 대학교(앤아버) 전기 및 컴퓨터 공학과 석사. 1990년 12월 미국 미시간 대학교(앤아버) 전기 및 컴퓨터공학과 박사. 1999년 9월~2000년 7월 미국 매사츄세츠 주립대학(앰허스트) 전산과 방문 연구원. 1991년 3월~현재 숭실대학교 정보통신전자공학부 부교수. 연구 관심분야 : 실시간 시스템, 분산 시스템, 멀티미디어 시스템, 임베디드 시스템