# 실시간 CORBA의 우선순위 모델 설계 및 구현

°박순례*, 정선태**
* 숭실대학교 정보통신전자공학부 (TEL: 817-5987; E-Mail: psun@syscon.soongsil.ac.kr)
** 숭실대학교 정보통신전자공학부 (TEL: 820-0638; E-Mail: cst@syscon.soongsil.ac.kr)

# A Design and Implementation of Priority Model of Real-Time CORBA

°Sun-Rei Park*, Sun-Tae Chung**
* Dept. of Electronics Eng. SoongSil Univ. (TEL: 817-5987; E-Mail: psun@syscon.soongsil.ac.kr)
** Dept. of Electronics Eng. SoongSil Univ. (TEL: 820-0638; E-Mail: cst@syscon.soongsil.ac.kr)

## Abstract

The Current CORBA has many drawbacks to be deployed successfully in real-time system applications. Recently, OMG adopted Real-Time CORBA specification. In this paper, we report our efforts of a design and implementation of Priority Model of Real-Time CORBA spec., which is one of the most important components in Real-Time CORBA spec. The improvement of real-time performances of our implementation is verified by expereiments.

## 1. 서론

CORBA(Common Object Request Broker Architecture)는 OMG(Object Management Group)에서 규정한 미들웨어 표준안으로써, 객체 지향 방식(OOD/OOP)이 갖는 장점들을 지원하면서, 플랫폼과 사용 프로그래밍 언어에 독립적인 미들웨어 환경을 지원하므로, 현재 분산 컴퓨팅 환경의 미들웨어 표준으로 자리를 잡아가고 있다[1]. 현재 규약된 CORBA (현재 규약 버전은 3.0) 는 처음에 비즈니스환경의 클라이언트/서버 분산 컴퓨팅 환경 구축을 염두에 두고 주로 개발되어 왔기 때문에, 분산 제어 시스템 등 실시간성 지원이 필요한 분산 시스템 구축에 사용하기에는 부족한 점이 많다. 실시간 시스템은 태스크 처리가 제시간(real-time)에 수행을 마치지 않으면, 시스템의 성능에 영향을 미치는 시스템을 말하는 데, 이러한 실시간 시스템에서는 각 태스크가 제시간에 수행이 종료되는 것이 보장되기 위해서는 각 태스크 처리에 우선순위가 지원되어야 할 필요가 있다. 즉, 긴급하게 수행되어야 할 태스크는 우선적으로 수행되어야 한다. 그런데, 현재의 CORBA는 클라이언트 요청 처리에 우선순위를 지원하지 못하므로, 보다 긴급한 클라이언트 요청 처리가 먼저 요청된 낮은 우선순위의 요청처리에 의해 블록킹이 되는 우선순위 역전(priority inversion) 현상이 발생하게 되게 된다. 따라서, CORBA의 실시간성 개선을 위하여, 많은 연구가 진행되어 왔으며[4와 그곳에 언급된 참고문헌], 최근에는 OMG(Object Management Group)에서 '실시간 CORBA 명세'에 대해 발표하였다[2]. 본 논문에서는 본 연구원들의 연구실에서 진행하고 있는 '실시간 CORBA 설계 및 구현'에 관한 연구의 일환으로, 실시간 CORBA 사양중 가장 중요한 아이디어인 '우선순위 모델' 에 대해 설계하고 구현한 내용을 기술한다. 본 논문의 연구가 기존의 실시간 CORBA 연구와 다른 점은 기존의 실시간 CORBA 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 문맥을 따라 설계되고 구현된 것이라는 점이다. 현재, 실시간 CORBA 사양의 구현을 진행하고 있는 회사 및 연구 기관들이 있으나, 아직 그 구체적인 내용이 보고되어 있는 것은 없다. 본 논문에서의 구현은 오픈소스 ORB로 잘 알려진, 비실시간 ORB 인 omniORB3.0.0을 이용하여, 이에 실시간 CORBA의 '우선순위 모델' 지원기능을 확장한 형태로 하였다. 본 논문에서 설계하고 구현한 실시간 CORBA의 우선순위 모델의 실시간성 개선 결과는 간단한 실험을 통해 확인하였다.

## 2. CORBA 동작 구조

CORBA 구조에 대한 자세한 내용은 [1]를 참조하기로 하며, 여기서는 본 논문의 이해에 필요한 ORB(Object Request Broker), 객체 어댑터(POA; Portable Object Adapter), 객체 레퍼런스(IOR : Interoperable Object Reference) 등의 역할에 대해서 CORBA의 동작 메커니즘을 통해 간단히 기술한다. CORBA에서 지원하는 가장 중요한 기능은 'location transparency' 이다. 즉, (서버) 객체가 어디에 위치하던지 간에 객체가 제공하는 서비스를 클라이언트가 이용할 수 있게 하는 것이다. 이는 다음과 같은 메커니즘으로 달성된다. 클라이언트는 서버 객체의 레퍼런스(IOR; Interoperable Object Reference)를 얻어, 이를 통해 해당 서버 객체의 서비스(메소드)를 호출하게 된다. 클라이언트측 ORB는 해당 객체의 레퍼런스(IOR)로부터 서버 객체가 위치하는 서버측 ORB의 위치 정보(IIOP를 이용하는 경우, IP 주소 및 TCP 포트 주소), 서버측 ORB의 객체 어댑터 내에서의 '서버 객체의 식별정보(object key)'를 추출하고, 이를 이용하여 서버측 ORB 로 클라이언트의 서비스 요청을 전송하며, 서버측 ORB는 객체 어댑터와 협력하여, 전송된 서비스 요청 메시지에 포함된 서버 객체의 object key를 이용하여 해당 서버 객체를 찾아내고, 서비스를 제공하는 객체의 메소드를 호출한다. 이후 수행된 결과는 다시 클라이언트 ORB로 보내지고 클라이언트 ORB 는 최종적으로 이 결과가 클라이언트에게 전달되도록 한다. 이때, ORB는 다른 ORB와의 통신을 위해 GIOP(General Inter-ORB Protocol) 프로토콜을 사용하는 데, GIOP를 TCP/IP 로 매핑한 프로토콜을 IIOP(Internet Inter-ORB Protocol)이라 한다. GIOP 메시지의 포맷은 Request, Reply를 비롯한 8가지의 종류가 있다[1]. ORB 의 중요한 기능이 바로 GIOP 프로토콜 메시지를 생성하고 해석하는 GIOP 엔진 기능의 제공이다. 클라이언트가 서버 객체 메소드를 호출할 때, 클라이언트측 ORB의 GIOP 엔진은, object key 및 요청 메소드 이름 등의 정보를 담는 GIOP Request 메시지를 생성하여, 서버측으로 보

내고, 서버측 ORB의 GIOP 엔진은 이 GIOP Request 메시지를 해석하여, 해당 서버객체의 메소드가 수행되게 하며, 이후 수행결과를 담는 GIOP Reply 메시지를 만들어 다시 클라이언트 측에 보낸다.

## 3. Real-Time CORBA 사양 및 우선순위 모델

### 3.1 실시간 CORBA 사양 이해

Real-Time CORBA는 OMG에서 CORBA의 실시간성 개선을 위하여 제안하고 있는 규약으로, 기존 CORBA의 확장개념으로 실시간 특성을 지원하기 위해, RTCORBA ::RTORB(이하, RTORB), RTPortableServer::POA(이하, RTPOA), Real-Time Scheduling Service 등의 구조를 추가하였다. 실시간 CORBA 규약의 자세한 내용은 관련 문헌[2]을 참조하라.

### 3.2 실시간 CORBA의 우선순위 모델

1) 우선순위 모델 이해

실시간 CORBA 규약 가운데 실시간성 개선을 위해 선택한 가장 중요한 아이디어는, 클라이언트측에서 서비스를 호출할 때 와 서버측에서 클라이언트의 서비스 요청을 처리할 때, 보다 긴급한 호출 및 요청처리가 먼저 수행될 수 있도록 '우선순위 부여 구조'를 제공하고자 한 것이다. 실시간 CORBA 사양에서 지원하고 있는, 우선순위의 부여 구조에는 두가지가 있다. 하나는 '클라이언트 전달 우선순위 모델(Client Propagated Priority Model)' 이고, 다른 하나는 '서버 선언 우선순위 모델(Server Declared Priority Model)' 이다. '클라이언트 전달 모델은 클라이언트 측에서 서비스 요청할 때, 우선순위를 지정하며, 이 우선순위는 서버 측에도 전달되어 서버에서 서비스 요청을 처리할 때에 전달된 우선순위를 존중하여 처리되도록 하는 방식이다. '서버 선언 우선순위 모델'은 서버의 객체가 서버측 POA에 등록될 때, 우선순위를 지정할 수 있도록 한 방식이다. 이 모델의 경우, 이후 클라이언트 측에서 이 서버 객체의 서비스를 호출할 때, 지정된 우선순위가 존중되어 호출될 수 있도록 하며, 서버 측 ORB에서 이 객체의 서비스 요청을 처리할 때에도 해당 객체의 우선순위가 존중되어 처리되도록 지원한다. 실시간 CORBA 의 우선순위에 대해 분명히 알아야 할 중요한 점은, 실시간 CORBA가 규정하는 우선순위는 쓰레드의 우선순위를 말하며, 이에는 2가지가 있다는 것이다. 하나는 CORBA를 사용하는 분산 응용 전체에 걸쳐 유일한 CORBA 우선순위(CORBA priority)이며, 다른 하나는 응용의 운영체제가 관리하는 지역 우선순위(native priority)이다. 지역 우선순위는 이종의 운영체제 환경에서, 각 운영체제마다 다르므로, 시스템 전체에 걸쳐 통일적인 우선순위의 정의가 필요하다. 우선순위 모델에서 사용하는 우선순위는 CORBA 우선순위이며, 이는 쓰레드 스케쥴링을 위해 지역 운영체제가 지원하는 지역 우선순위로 변경되어야 한다. 실시간 CORBA 는 이를 위해 우선순위 매핑(priority mapping)을 규약하고 있다.

2) 우선순위 모델 정책 및 실시간 IOR

실시간 CORBA가 지원하는 우선순위 모델은 PriorityModelPolicy 라는 CORBA 객체를 통해 선택되고, 구성되도록 규약되어 있다. PriorityModelPolicy 객체는 우선순위 모델 정보 (클라이언트 전달 우선순위 모델 또는 서버선언 우선순위 모델) 와 우선순위 정보를 보유한다. 우선순위 모델 정책은 RTPOA 가 생성될 때, 서버측에 적용된다. PriorityModelPolicy 는 클라이언트에게 노출되는 정책이며, 이는 IOR를 통해 서버로부터 클라이언트로 전달된다. 보다 자세한 내용은 실시간 CORBA 사양[2] 참조. 기존 비실시간 ORB 가 사용하는 IOR 은 이러한 PriorityModelPolicy 정보를 지원하지 않기 때문에 실시간 ORB 구현시에는 기존 비실시간 IOR 에 PriorityModelPolicy 정보가 추가된 실시간 IOR (이하 RT-IOR)를 지원하도록 ORB 의 구조가 확장되어야 한다.

3) 우선순위 및 서비스 컨텍스트

타겟 객체가 클라이언트 전달 우선순위 모델을 지원하는 경우, 클라이언트는 서버 객체 메소드 호출의 우선순위를 지정할 수 있으며(RTCORBA::Current 를 이용하여), 이렇게 지정된 클라이언트의 우선순위는 서버 객체 메소드 호출시에 GIOP Requset 메시지 헤더의 서비스 컨텍스트로 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, 서버측에서 서버측 운영체제의 지역(native) 우선순위로 바뀌는 우선순위 매핑이 필요함에 유의해야 한다. GIOP Request 및 Reply 메시지 포맷의 서비스 컨텍스트는 GIOP 1.1 이후부터 정의되며, 정의된 서비스 컨텍스트의 구조는 실시간 CORBA 사양[2]을 참조하라.

## 4. 실시간 CORBA의 우선순위 모델 설계 및 구현

본 논문에서는 기존 ORB로는 소스코드가 공개되어 있고, 성능이 우수한 것으로 잘 알려져 있는 omniORB2 (v3.0.0) 를 사용하여 우선순위 모델을 구현하였다. 그런데, 기존 ORB를 이용하는 경우, 우선순위 모델의 설계 구조가 기존 ORB 에 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 기반되어 있음을 밝힌다.

4.1절에서는 우선순위 모델 구현을 위해서 필요한 CORBA의 구성요소에 대해 기술한다. 4.2절에서는 omniORB2 에 기반하여 확장 구현한 내용에 대해 간단히 기술한다.

### 4.1 우선순위 모델 구현을 위해 필요한 실시간 CORBA 요소 이해

1) RT-POA

RT-POA 는 기존 비실시간 POA(PortableServer::POA)의 자식 객체(계승)로 규정되며, 우선순위 모델 및 객체별 우선순위를 지정하여 RT-POA 에 등록하는 함수들을 규정한다. POA는 CORBA 객체의 object key와 이 CORBA 객체를 구현한 서버 객체에 관한 정보(서버 객체 위치 정보, 등록된 POA 위치 정보 등)와의 매핑을 제공하는 데, 이 매핑구조를 위해 CORBA는 객체 맵(active object map)을 규정하고 있다. RT-POA 에는 이 객체 맵에 우선순위 모델 정보와 우선순위 정보가 추가되어야 하며, 우선순위를 지원하는 객체를 지원하는 함수가 구현되어야 한다.

2) RT-IOR 및 실시간 객체레퍼런스

CORBA에서 CORBA 객체에 대한 정보(repository ID(객체 인터페이스 식별정보), profiles (서버 객체의 TCP/IP 주소, object key 등을 포함한 정보))는 IOR를 통해 클라이언트측에 전달된다. 클라이언트는 해당 객체의 IOR를 네이밍 서비스를 통하거나, 직접 서버로부터

입수하는 데, 이 IOR은 GIOP Reply 메시지의 body 에 실려 전달된다. 클라이언트 측 ORB의 GIOP 엔진은 GIOP Reply 메시지 처리시에, IOR를 꺼내고 이 IOR 로부터, repository ID 와 profiles 정보를 추출하여, 이 정보들을 보유하는 객체레퍼런스 타입의 스터브 객체를 생성한다. 앞 3.2절에서 언급한 바처럼, RT-IOR 은 비실시간 IOR 에 비해 우선순위 모델 정보 및 우선순위 정보를 포함한다. 클라이언트측 GIOP 엔진에서는 해당 객체의 RT-IOR을 입수하는 즉시, IOR 로부터 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-IOR 로부터 실시간 객체레퍼런스 타입 객체의 생성 등을 지원하여야 한다.

3) 실시간 ORB의 쓰레드 우선순위 변경 메카니즘 구조

클라이언트가 해당 객체의 메소드를 호출할 때, 클라이언트 ORB는 해당 메소드 호출을 GIOP Request 메시지로 변경하기 전에, 해당 객체와 연관된 PriorityModelPolicy 객체를 참조하여, 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 서버로 보낸다. 해당 객체가 서버 선언 모델인 경우, 호출 쓰레드의 우선순위를 PriorityModelPolicy 객체에 보관된 우선순위를 반영하도록 변경한 후, 해당 메소드의 GIOP Request 메시지 만들고 이를 서버측에 보낸다. 이때, 서버 선언의 경우는 우선순위를 싣는 서비스 컨텍스트를 만들지는 않는다.

서버측 ORB는 클라이언트 ORB 가 보낸 GIOP Request 메시지를 받게 되면, GIOP Request 메시지의 서비스 컨텍스트 부분을 점검하여 우선순위 정보가 있으면, 해당 작업 쓰레드의 우선순위를 이를 반영하여 변경한다. 이후, 서버측 ORB 의 클라이언트 요구처리는 변경된 쓰레드의 우선순위로 처리된다. 만약 우선순위 정보가 없으면, ORB는 GIOP Request 메시지에서 해당 서버 객체의 object key 값을 추출하여, 이를 이용하여 객체 맵 테이블에서 해당 객체를 찾게 된다. 이때, 등록된 해당 객체의 우선순위 모델 및 우선순위 정보를 점검하여, 서버 선언 모델로 등록되어 있으면, 등록된 우선순위를 반영하여 작업 쓰레드의 우선순위를 변경하며, 이후의 클라이언트 요구 처리는 변경된 쓰레드의 우선순위로 처리된다.

### 4.2 우선순위 모델 구현 내용

이 절에서는 omniORB2 에 기반하여 구현한, 본 논문의 우선순위 모델 지원 실시간 CORBA 의 구현 내용을 간단히 기술한다.

1) RT-POA

omniORB2에서 구현하고 있는 POA의 객체맵(active object map) 은 omniLocalIdentity 라는 객체의 포인터들의 해쉬 테이블로 구현하고 있다. omniLocalIdentity 객체는 구현 서버 객체의 포인터, 해당 POA 포인터 등의 정보를 멤버변수로 갖는다. 객체의 object key는 이 해쉬 테이블의 인덱스 키가 된다. 즉, 서버측 ORB는 object key를 이용하여 객체맵 내에서 해당 omniLocalIdentity 객체를 찾아내고, 이 객체에서 구현 서버 객체 포인터를 찾아, 이를 이용하여 서버 객체의 메소드를 호출하도록 되어 있다. 이러한 omniORB2 의 POA 구조를 확장하여, 본 논문의 RT-POA 는 PriorityModelPolicy 객체와 추가적으로 연관되도록 하였으며, omniLocalIdentity 객체에 우선순위 정보를 추가하였다.

2) RT-IOR 및 실시간 객체레퍼런스

omniORB2 에서는 실시간 IOR를 지원하지 않는다. 본 논문에서는 3.2 절에 설명한 RT-IOR를 설계하고 구현하였으며, 비실시간 IOR 도 구현된 ORB에서 동작되도록 하였다. 또한, 객체레퍼런스가 갖는 객체의 profiles 를, RT-IOR에서 확장된 우선순위 모델 및 우선순위 정보를 추가적 으로 보유할 수 있도록 변경하였다. 또한, 기존 비실시간의 IOR 생성, 객체레퍼런스 타입의 객체 생성을 지원하는 모듈을 확장하여, RT-IOR 생성, 실시간 객체레퍼런스 타입의 객체(스터브 등) 생성 등을 추가적으로 지원할 수있도록 변경하였다.

3) 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 (클라이언트측에서의 GIOP Request 메시지 생성 부분, 서버측에서의 GIOP Request 메시지 해석 부분, GIOP Reply 메시지 생성 부분, 쓰레드 우선순위 변경 부분 등)을 확장하여, 비실시간 GIOP 메시지 뿐만 아니라, 우선순위 모델 지원 실시간 GIOP 메시지 처리가 가능하도록 하였다.

## 5. 실험 및 결과 검토

### 5.1 실험환경

이 절에서는, 본 논문에서 설계하고 구현한 실시간 CORBA 가 기존의 비실시간 CORBA 에 비해, 확실하게 우선순위를 존중함을 보인 실험결과를 기술한다. 우선순위의 존중은 성능과 예측성으로 측정하였다. 성능은 클라이언트가 서버 객체의 메소드를 호출하였을 때 걸린, 평균 RTT지연시간 (Average Round Trip Time Latency)으로 측정하였으며, 예측성은 지터(jitter)로 측정하였다. 여기서, RTT 지연시간은 클라이언트가 서버 객체의 메소드를 호출을 시작하여 호출을 완료하기까지의 걸린 시간을 의미하며, 이 때 평균 RTT 지연시간은 동일 메소드를 m 번 반복 호출하여 각 호출의 RTT 지연시간을 평균한 값을 의미하며, 지터는 이 m 번 반복 측정 지연시간의 표준 편차를 의미한다. 본 논문의 실험에서 평균 RTT 및 지터 비교을 위해 사용한 서버 메소드는 클라이언트가 보낸 8KBytes 중 첫 1 바이트를 3제곱하여 다시 클라이언트측에 돌려주는 서비스(이하 cubic 서비스)를 제공하는 것을 이용하였다. 서버 객체들은 같은 메소드를 제공하나, 다른 객체 레퍼런스를 가진 다른 객체들로 서버 POA 에 등록되며, 비실시간 ORB 의 경우, 우선순위의 지정없이 등록되며, 서버 선언 우선순위 모델의 경우는 각 객체별로 우선순위가 등록되며, 클라이언트 전달 우선순위 모델의 경우는 클라이언트측에서 우선순위를 지정하였다. 본 논문의 실험에서는 클라이언트측의 n 개의 쓰레드가 각각 이 서버 객체의 메소드를 4000번 반복하여 호출하도록 하였다. 클라이언트 측의 n 개의 쓰레드중 (n-1) 개의 쓰레드의 우선순위는 모두 동일하며, 1 개의 쓰레드는 다른 쓰레드에 비해 우선순위가 높다. 서버선언 우선순위 모델 실험의 경우, 서버 객체는 호출하는 쓰레드의 우선순위를 반영한 우선순위가 등록되며, 클라이언트 전달 우선순위 모델의 경우, 호출되는 서버 객체의 메소드 처리가 클라이언트의 해당 호출 쓰레드의 우선순위를 반영하여, 처리된다. 이 절의 실험 결과는 낮은 우선순위 쓰레드들중 1, 높은 우선순위 쓰레드 1 의 2 개 쓰레드에 대해 이 메소드 호출의 평균 RTT 지연시간과

지터를 비실시간 CORBA 및 실시간 CORBA의 클라이언트 전개 우선순위 모델 및 서버 서언 우선순위 모델의 3가지 경우에 대해 측정하여 비교한 것이다.

본 논문에서 사용한 실험환경은 Pentium II 416Mhz, 128Mbyte(Memory)의 하드웨어 사양에 윈도우즈 NT를 탑재한 PC이다. 또한, LAN 사용시에 발생할 수 있는 다른 트래픽의 영향을 배제하기 위해, 클라이언트 및 서버 모두 같은 머신 상에서 동작시켰으며, 실험시에 운영체제의 시스템 프로세스의 영향을 배제하기 위해, 클라이언트 및 서버 프로세스는 "REAL_TIME_PRIORITY_CLASS" 라는 가장 높은 프로세스 우선순위를 사용하였다. 채택한 클라이언트측의 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).

### 5.2 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, ORB 의 성능을 보여주는 좋은 측도이다. 특히 우선순위가 높은 오퍼레이션의 마감시간 준수가 더 요구되는 실시간 분산 시스템의 환경에서, 높은 우선순위 쓰레드의 메소드 호출의 평균 RTT 지연시간의 개선은 매우 필요하다. 본 절의 실험 결과는 비실시간 ORB (기존의 omniORB)에 비해 본 논문에서 설계하고 구현한 실시간 ORB 의 클라이언트 전달 우선순위 모델 이나 서버 선언 우선순위 모델의 경우, 우선순위를 존중하여, 높은 우선순위 쓰레드의 (높은 우선순위로 등록된 객체)메소드 호출의 경우, 평균 RTT 지연시간이 비실시간 ORB 의 경우에 비해 개선되었음을 보여준다. 그림 1은 비실시간 ORB 와 클라이언트 전달 우선순위 모델의 실시간 ORB, 서버 선언 우선순위 모델의 실시간 ORB 에서의 높은 우선순위 쓰레드의 메소드 호출의 평균 RTT 지연시간을 비교한 것이다. 그림 1 에서 x 축은 낮은 우선순위 쓰레드 개수를 나타내며, y 축은 높은 우선순위 쓰레드의 메소드 호출 평균 지연시간이다. 부하가 작을 경우에는 비교적 차이가 적으나, 부하가 커지는 경우, 클라이언트 우선순위 모델의 실시간 ORB 의 경우와 서버 선언 우선순위 모델의 실시간 ORB 경우가 비실시간 ORB 에 비해 평균 RTT 지연시간의 개선의 차이가 큼을 볼 수있다.

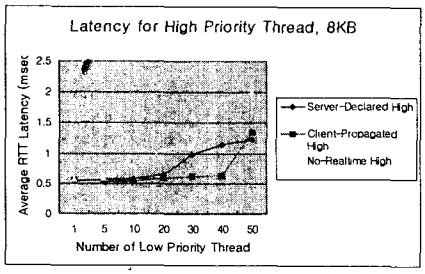


그림 1. 비실시간 ORB 와 우선순위 모델 지원 실시간 ORB의 평균 RTT 지연시간 비교

### 5.3 지터 비교

지터는 각 메소드 호출의 RTT 지연시간의 변동을 측정하는 측도로써, 지터가 작다는 것은 각 메소드 호출의 RTT 지연시간이 균일하다는 것을 보여 주게 되므로, 예측성이 요구되는 실시간 시스템 환경에서 필요한 특징이다. 특히 우선순위가 높은 쓰레드의 서버 객체 메소드의 호출에 있어서, 높은 예측성의 보장이 실시간 시스템 구축에 바람직하다. 본 절의 실험 결과는 비실시간 ORB (기존의 omniORB)에 비해 본 논문에서 설계하고 구현한 실시간 ORB 의 클라이언트 전달 우선순위 모델 이나 서버 선언 우선순위 모델의 경우, 우선순위를 존중하여, 높은 우선순위 쓰레드의 (높은 우선순위로 등록된 객체)메소드 호출의 경우, RTT 지연시간의 지터가 비실시간 ORB 의 경우에 비해 개선되었음을 보여준다. 그림 2는 비실시간 ORB 와 본 논문에서 구현한 클라이언트 전달 우선순위 모델, 서버선언 우선순위 모델에 있어서, 높은 우선순위 쓰레드의 서버 객체 메소드 호출의 RTT 지연시간을 비교한 것이다. 그림 2에서 보면, 클라이언트 전달 우선순위모델과 서버선언 우선순위 모델의 경우가 지터가 낮아 보다 높은 예측성을 지원함을 알 수 있다.

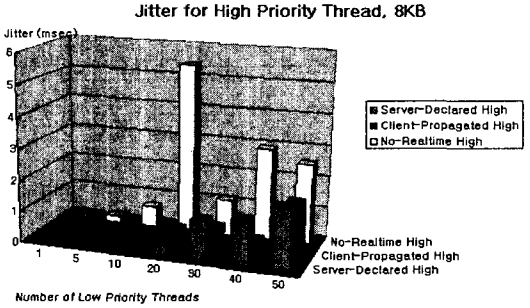


그림 2. 비실시간 ORB와 우선순위 모델 지원 실시간 ORB의 지터비교

## 6. 결론 및 향후 과제

본 논문에서는 실시간 CORBA 사양중 '우선순위 모델' 에 대해 기존 비실시간 오픈 소스 omniORB2 에 기반하여 설계하고 구현한 연구 내용을 보고하였다. 본 논문의 실험결과는 구현된 우선순위 모델의 실시간 ORB 의 실시간성 개선을 확인한다. 본 논문의 실험에서 사용한 cubic 서비스는 서버 구현 객체의 해당 메소드내에서 작업하는 양이 적다. 만약 서버 구현 객체내에서 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. 그밖의 실시간 CORBA의 사양에 대해서(특히 쓰레드 풀 및 이에 연관된 구조), 구현 작업도 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.

## 참고 문헌

[1]M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*, Addison-Wesley, 1998.
[2] OMG, *Real-Time CORBA Joint Revised Submission*, OMG TC Document, ptc/99-05-03.
[3] omniORB2, http://www.uk.research.att.com/omniORB2
[4] 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.